

浙江大学



题 目： MiniSQL 数据库系统设计报告

授课老师： 孙建伶

课 程： 数据库系统设计

组员姓名	学号	专业
丘颖悦	3130100723	计算机科学与技术
叶俊利	3130100672	计算机科学与技术
郑濡樟	3130100670	计算机科学与技术

MiniSQL 数据库系统设计报告.....	3
1. 引言.....	3
1.1 项目名称.....	3
1.2 项目背景和内容概要.....	3
2. 系统结构.....	3
2.1 系统功能.....	3
2.2 系统结构图.....	4
2.3 系统目录结构.....	4
2.4 基本设计概念.....	5
2.5 程序模块说明.....	6
Interpreter.....	6
词法解析.....	6
语法和语义分析.....	6
语法分析.....	7
语义分析.....	8
API.....	9
函数接口介绍.....	9
Catalog Manager.....	10
index 结构.....	10
table 结构.....	10
attribute 结构.....	11
CatalogManager 类.....	11
Buffer Manager.....	12
Block 类.....	12
Buffer Manager 类.....	13
Record Manager.....	15
tuple 类.....	16
conditionNode 类.....	16
RecordManager 类.....	18
File Manager.....	21
IndexManager.....	21
IndexManager 类.....	21
BPlusTree 类.....	22
B+树结构.....	26
3. 测试结果.....	27
A 功能测试.....	27
Create table 功能.....	27
Insert 功能.....	27
Select 功能.....	28
B 语法错误测试.....	31
C 语义错误测试.....	32
4. 成员分工.....	34

MiniSQL 数据库系统设计报告

1. 引言

1.1 项目名称

MiniSQL 数据库系统设计与实现。

1.2 项目背景和内容概要

数据库系统设计与实现实验。

主要目的：

设计并实现一个精简型单用户 SQL 引擎(DBMS)MiniSQL, 允许用户通过字符界面输入 SQL 语句实现表的建立/删除; 索引的建立/删除以及表记录的插入/删除/查找。

通过对 MiniSQL 的设计与实现, 提高学生的系统编程能力, 加深对数据库系统原理的理解。

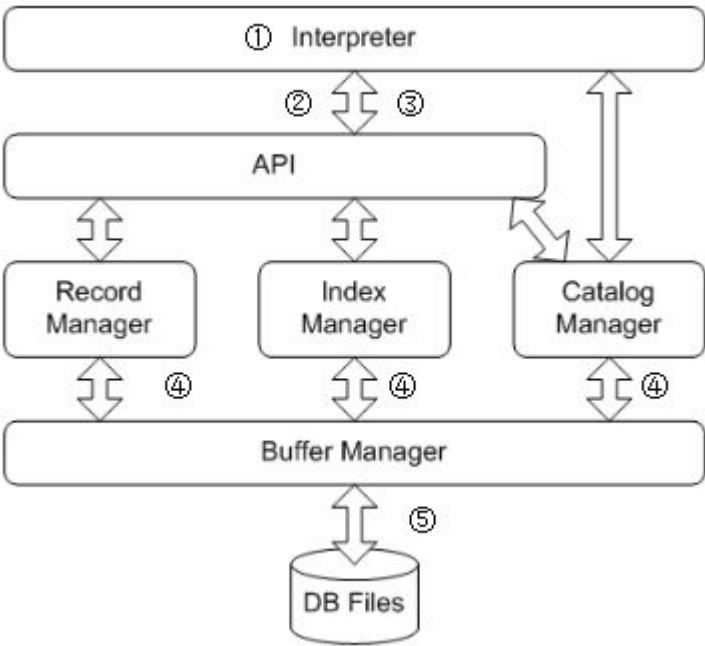
2. 系统结构

2.1 系统功能

最终设计出来的 MiniSQL 除了支持基本的数据库系统功能以外, 还拓展设计了一些附加功能, 具体支持的功能列表如下:

- ①Table 操作: 包括表定义、表新建与表删除
- ②Insert 操作: 单条数据的插入
- ③Select 基本操作: 基本的表内容查询功能, 显示所有的记录。
- ④【BONUS 功能】Projection 功能: 支持查询结果的 Projection。
- ⑤查询结果条件判断功能:
 - (a) 普通单属性条件语句判断。
 - (b) 带 and 的条件语句: 支持与条件查询。
 - (c)【BONUS 功能】带有 or 的条件语句: 支持或条件查询。
 - (d)【BONUS 功能】带有括号的条件语句: 支持带括号的条件有限查询功能。
- ⑤【BONUS 功能】Order by 排序功能: 支持结果的排序功能。
- ⑥【BONUS 功能】Join 功能: 支持表的连接功能。
- ⑦ Delete 操作: 数据的条件删除, 支持多条。
- ⑧ Index 操作: 支持 unique 属性的索引建立、删除以及数据更新时的维护。

2.2 系统结构图



- 图中各标号简明解释：
- ①判断并接受用户字符输入，使做为解释器的输入。
 - ②解释器对用户输入进行翻译，调用 API 接口。
 - ③执行选定的 API，返回用户所需的输出。
 - ④BPlus、Record、Catalog 类调用 Buffer 类的方法实现自己各自的方法。
 - ⑤Buffer 类方法对数据库文件进行直接操作。

2.3 系统目录结构

MinSQL 的文件目录主要包括了 bin 和 src 两个文件夹。

bin 目录下存放 MiniSQL 的 class 类文件，主要执行的都是这里的文件；

src 目录下存放 MiniSQL 的程序源文件，所有的功能更改与添加都通过修改其中的模块来完整，里面的程序清单如下表所示：

模块名		程序文件名	语言	简要描述
用户接口	解释器	Interpreter.java	java	解释器模块实现文件
	API 集成	API.java	java	API 实现文件
系统	IndexManager	BPlusTree.java	java	B+树实现文件
		IndexManager.java	java	索引模块实现文件
		offsetInfo	java	索引结果文件
	RECORDMANAGER	conditionNode.java	java	查询条件类实现文件

内 核		tuple.java	java	记录类实现文件
		RecordManager.java	java	RecordManager 实现文件
	CATALOGMANAGER	attribute.java	java	表属性类文件
		index.java	java	索引信息类文件
		table.java	java	表信息类文件
		CatalogManager.java	java	CatalogManager 实现文件
	BUFFERMANAGER	Block.java	java	块信息类文件
		BufferManager.java	java	BufferManager 实现文件
	FILEMANAGER	FileManager.java	java	FILEMANAGER 实现文件
	lexer	词法分析器		

其余的数据文件类型主要由以下几种：

Catalog 文件：主要有 index catalog.txt 以及 table catalog.txt，主要是为整个 MiniSQL 的数据库管理服务，保存了建立的表以及表信息、索引信息等总的数据库资料。

.index 文件：主要保存了相关的 B+树索引文件结构。

Table 文件：保存表中插入的相关 record 的信息。

2.4 基本设计概念

1. 系统目标

设计并实现一个精简单用户 SQL engine，并在其中实现表定义、索引、表记录操作功能。其中：

（1）表定义中列（属性）的类型至少支持三种：integer、char、float（其中 char(n)满足 $1 \leq n \leq 255$ ）；

（2）一个表最多可以定义 32 个属性，各属性可以指定是否为 unique；支持单属性的主键定义。

（3）对于表的主属性自动建立 B+树索引，对于声明为 unique 的属性可以通过 SQL 语句由用户指定建立/删除 B+树索引（因此，所有的 B+树索引都是单属性单值的）；

（3）支持每次一条记录的插入操作；支持每次一条或多条记录的删除操作，并能即时更新相应的索引；

（4）记录的搜索至少实现按主键查找，支持主键上的范围查找，包含遍历。可以通过指定用 and 连接的多个条件进行查询，支持等值查询和区间查询。

2. 结构清晰

（1）用户模块和内核模块完全分开。

（2）内核部分 IndexManager、RecordManager、CatalogManager 三个模块不能访问物理文件，而由 BufferManager 模块实现物理文件操作的所有细节。（CatalogManager 模块在系统初始化以及退出时要读取和更新 Catalog 相关文件，这部分操作由于比较特殊所以独立于 BufferManager）。

（3）API 根据 IndexManager、RecordManager、CatalogManager 三个模块的方法整合生成，适合用户模块调用的 API。

2.5 程序模块说明

Interpreter

基本功能：将用户的输入命令进行语法分析和语义解析并得到需要的命令参数,最后将该命令参数封装成对应命令的参数类对象,传到 API 模块; 同时对于 API 返回的操作结果进行输出显示。

词法解析

工具：Lexer 类

	名称	功能描述
成员变量	char peek	下一个读入字符
	Hashtable<String, Word> words	关键字存储
	BufferedReader reader	
	Boolean isReaderEnd	判断当前是否读取到了文件的结尾
外部接口	Lexer(BufferedReader reader)	构造函数，初始化关键字，将关键字存进哈希表
	Boolean getReaderState()	是否读取到输入流的结尾
	Token scan()	读取字节流，返回 token，可通过 token 标签进行判断单词类型判断

相关数据结构

类名称	作用
Token	符文，用标签 tag 分类
Comparison 继承 Token	操作符（<,<=,>,>=,=,<>）
Num 继承 Token	数字
Word 继承 Token	单词
Tag	Token的标签（所有关键字、 STR——字符串 INTNUM ——整数 FLOATNUM——浮点数 TYPE——字段类型 OP——操作符 ID——表名、索引名或字段名）

语法和语义分析

工具:Interpreter 类

	名称	功能描述
类变量	Token thetoken	下一个读入符
	boolean isSynCorrect=true;	标记当前语句是否语法正确，初始化为真

量	<code>boolean isSemaCorrect=true;</code>	标记当前语句是否语义正确，初始化为真
	<code>String synErrMsg;</code>	记录当前语法错误信息
	<code>String semaErrMsg;</code>	记录当前语义错误信息

语法分析

判断方法：通过状态机来实现状态跳转。

在读取到 `lexer.getReaderState()==false`(输入流末尾)前，循环读取 `lexer.scan()`函数返回每个 `token` 标签进行判断，来决定下一个要进入的状态。符合正确语法的跳转到下一状态，不符合正确语法时，记录语法错误信息，将语法标记为假，同时跳过本轮循环。在下一轮循环开始前，将会输出语法错误信息，并将语法标记重新置为真。

支持语句

1. 创建表语句

```
create table 表名 (
    列名 类型 ,
    列名 类型 ,

    列名 类型 ,
    primary key ( 列名 )
);
```

2. 删除表语句

```
drop table 表名 ;
```

3. 创建索引语句

```
create index 索引名 on 表名 ( 列名 );
```

4. 删除索引语句

```
drop index 索引名 ;
```

5. 选择语句

```
select *(列名*) from 表名 ;
```

或:

```
select *(列名*) from 表名 where 条件 ;
```

或:

```
select *(列名*) from 表名 where 条件 order by 列名;
```

或:

```
select * from 表名 1,表名 2 where 表名 1.列名=表名 2.列名;
```

其中“条件”具有以下格式: 列 `op` 值 `and/or` 列 `op` 值 `...` `and/or` 列 `op` 值

或: 列 `op` 值 `...` `and/or` (列 `op` 值 `and/or` 列 `op` 值)

支持括号优先级

其中 `op` 可以为: `<>`、`<`、`<=`、`>`、`>=`、`=`

6. 插入记录语句

```
insert into 表名 values ( 值 1 , 值 2 , ... , 值 n );
```

7. 删除记录语句

```
delete from 表名 ;
```

或:

delete from 表名 **where** 条件 ;

其中“条件”具有以下格式: 列 op 值 and/or 列 op 值 ... and/or 列 op 值

或: 列 op 值 ... and/or (列 op 值 and/or 列 op 值)

支持括号优先级

其中 op 可以为: <>、<、<=、>、>=、=

8. 退出系统语句

quit;

9. 执行 SQL 脚本语句

execfile 文件名

语义分析

判断方法: 在当前状态为语法正确的基础上调用 **CatalogManager** 的接口对表和索引信息进行查询并进行判断。判断为错误时, 记录当前语义错误信息, 将语义标记为假。在本条语句语法完全基础上, 判断语法标记, 若为真, 调用执行命令语句, 若为假, 输出语义错误信息, 并将语义标记重新置为假。

1. create table 语义错误种类

- 1) table name 已存在
- 2) primary key 所指字段不存在
- 3) 出现重复的 attribute 字段
- 4) char(n) 的 n 越界

2. create index 语义错误种类

- 1) index name 已存在
- 2) table name 不存在
- 3) attribute 不存在
- 4) attribute 已经是索引
- 5) attribute 不是 unique

3. drop table 语义错误种类

- 1) table name 不存在

4. drop index 语义错误种类

- 1) 该 index name 不存在
- 2) 该 index name 是主键不能删除

5. insert into 语义错误种类

- 1) table 不存在
- 2) 插入的 tuple 数量不对
- 3) 插入的 tuple 类型 (及长度) 不对
- 4) 对于 unique key 字段有重复插入记录

6. delete 语义错误种类

- 1) table 不存在
- 2) where 条件有误: 字段名不存在; value 属性与字段属性不匹配

7. select 语义错误种类

- 1) table 不存在

- 2) `where` 条件有误：字段名不存在；`value` 属性与字段属性不匹配
- 3) `select` 或 `order` 的字段名不存在*
- 4) 两字段属性不同无法比较*

API

函数接口介绍

	名称	功能描述	内部实现
数据	<code>void Initialize()</code>	Minisql 的初始化与退出	BufferManager 和 CatalogManager 从文件中读取或数据库
	<code>void close()</code>		
	<code>void showCatalog()</code> <code>void showTableCatalog()</code> <code>void showIndexCatalog()</code>	显示 Catalog 信息	调用 CatalogManager 接口即可
SQL 语句操作	<code>boolean createTable(String tableName, table newTable)</code>	创建表格	CatalogManager 添加表格和主键索引的定义信息；Recordmanager 创建存储记录的文件；IndexManager 对主键创建索引
	<code>boolean dropTable(String tableName)</code>	删除表格	CatalogManager 删除表格和表中所有索引的定义信息；Recordmanager 删除存储记录的文件；IndexManager 删除表中所有索引
	<code>boolean createIndex(index newIndex)</code>	创建索引	CatalogManager 添加索引定义信息；IndexManager 创建索引
	<code>boolean dropIndex(String indexName)</code>	删除索引	CatalogManager 删除索引定义信息；IndexManager 删除索引
	<code>boolean insertTuples(String tableName, tuple theTuple)</code>	插入记录	CatalogManager 更新表和索引信息；Recordmanager 添加记录；IndexManager 对 B+树中做添加
	<code>int deleteTuples(String tableName, conditionNode conditionNodes)</code>	删除记录	CatalogManager 更新表和索引信息；Recordmanager 删除记录，返回删除记录数；IndexManager 在 B+树中做删除

Vector<tuple> selectTuples(String tableName, Vector<String> attriNames, conditionNode conditionNodes)	查询记录	CatalogManager 更新表信息（记录数改变） Recordmanager 返回查找结果
Vector<tuple> selectTuples(String tableName, Vector<String> attriNames, conditionNode conditionNodes, String orderAttri, boolean ins)	查询记录（含 order 指令）	
Vector<tuple> join(String tableName1,String attributeName1,String tableName2,String attributeName2)	查询记录（含 join 指令）	

Catalog Manager

Catalog Manager 负责管理数据库的所有模式信息，包括：

1. 数据库中所有表的定义信息，包括表的名称、表中字段（列）数、主键、定义在该表上的索引。
2. 表中每个字段的定义信息，包括字段类型、是否唯一等。
3. 数据库中所有索引的定义，包括所属表、索引建立在那个字段上等。

index 结构

功能描述：用于索引中的所有定义信息以及索引文件信息。

其内部成员为：

```
public class index{
    public String indexName;//索引名，唯一标记索引
    public String tableName;//表名
    public String attriName;//字段名
    public int column;        //字段列数
    public int columnLength;//字段长度
    public int rootNum;       //根节点数目
    public int blockNum=0;    //index_name.table 占用 block 数
}
```

table 结构

功能描述：用于存储表的所有定义信息，包括表的名称、表中字段（列）数、主键、定义在该表上的索引。

其内部成员为：

```
public class table{
    String tableName;        //表名
    String primaryKey;       //主键名
    Vector<attribute> attributes;//以vector方式存放字段
    Vector<index> indexes;    //以vector方式存放表上的索引
}
```

```

int indexNum;           //索引数量
int attriNum;           //属性数量
int tupleNum;           //记录条数
int tupleLength;        //单条记录总字节数
}

```

attribute 结构

功能描述：用于存储Table中的每个字段信息

其内部成员为：

```

public class attribute{
    String attriName; //字段名称
    String type;      //字段类型：int/float/char
    int length;       //字段字节数
    boolean isUnique; //字段是否为布尔型
}

```

CatalogManager 类

	名称	功能描述	作用方法
成员变量	Hashtable<String,table> tables	表和索引的对象	哈希表容器存放表和索引，实现主键名和实例的一一对应关系
	Hashtable<String,index> indexes		
	String tableFilename="table catalog"	表和索引	
	String indexFilename="index catalog"	文件名	
函数接口	信息管理接口		
	void InitialCatalog() void InitialIndexCatalog() void InitialTableCatalog()	初始化 Catalog	minisql 初始化时调用从文件中读取 Catalog 信息
	void storeCatalog() void storeIndexCatalog() void storeTableCatalog()	存储 Catalog	退出 minisql 时调用将内存中的 Catalog 信息写入文件中
	void showCatalog() void showIndexCatalog() void showTableCatalog()	显示 Catalog 信息	方便 Interpreter 调用查看当前 Catalog 信息
	数据库操作接口		
	boolean createTable(table newTable)	添加表格	Create table 操作调用
	boolean dropTable(String tableName)	删除表格	Drop table 操作调用
	boolean createIndex(index newIndex)	添加索引	Create index 操作以及 Create table 的自动调用
	boolean dropIndex(String indexName)	删除索引	Drop index 操作以及 Drop table 的自动调用
	void addTupleNum(String tableName)	增加记录数	insert 操作调用更新表格记录数信息

		void deleteTupleNum(String tableName,int num)	减少记录数	delete 操作调用更新表格记录数信息
		boolean updateIndexTable(String indexName,index indexinfo)	更新索引信息	Insert 和 delete 操作要更新索引信息(block 数等)
信息交互接口		boolean isTableExist(String tableName)	获取表的定义信息	主要用于 Interpreter 的语义判断以及 API、RecordManager 和 IndexManager 对 Catalog 信息的访问
		int getAttriNum(String tableName)		
		int getTupleLength(String tableName)		
		int getTupleNum(String tableName)		
		boolean isIndexExist(String indexName)	获取索引的定义信息	
		index getIndex(String indexName)		
		String getIndexName(String tableName,String attriName)		
		boolean isAttributeExist(String tableName,String attriName)	获取表中的字段定义信息	
		boolean inUniqueKey(String tableName,String attriName)		
		boolean isIndexKey(String tableName,String attriName)		
		int getAttriOffest(String tableName,String attriName)		
		String getType(String tableName,String attriName)		
		int getLength(String tableName,String attriName)		
		String getAttriName(String tableName,int i)		
		String getType(String tableName,int i)		
		int getLength(String tableName,int i)		

Buffer Manager

代码结构：

BUFFER MANAGER 包中包含两个类。Block 类以及 Buffer Manager 类。

Block 是用来记录块内 4KB 大小的数据以及标记位的类。

Buffer Manager 类是用于管理 buffer 的类。具体见下文。

Block 类

功能：作为直接返回给 Record Manager 或 Index Manager 使用的数据块。记录了块内 4KB 大小的数据以及标记位（脏数据位、有效位、reference 位、锁定位）。

类内各部分的功能描述及实现原理如下表所示：

名称	功能描述	实现原理
----	------	------

成员变量	byte[] data	数据区，4KB 大小	
	String filename	记录块所属文件名	
	int blockoffset	记录这个块属于这个文件的第几个块	
	boolean dirty	是否脏数据	
	boolean valid	有效位	
	boolean fixed	是否被锁定	
	boolean reference_bit	引用位，用于 LRU 算法	
外部接口	byte[] readData()	用于读出 4KB 数据	返回内部成员 data，将引用位置 1
	boolean writeData(int byteoffset, byte inputdata[], int size)	用于将 inputdata[] 中的数据写入块中	写入数据后，将引用位置 1，dirty 位置 1 以标记为脏数据
	boolean writeData()	用于在直接修改 data 之后发出脏数据信号及引用位信号	写入数据后，将引用位置 1，dirty 位置 1 以标记为脏数据
	void fix()	把块锁定在缓冲区	fix 位置 1 以锁定
	void unfix()	把块从缓冲区解锁	fix 位置 0 以解锁
	int readInt(int offset)	从块中的指定位置读出一个整数	读出数据后，将引用位置 1
	void writeInt(int offset, int num)	从块中的指定位置写入一个整数	写入数据后，将引用位置 1，dirty 位置 1 以标记为脏数据
	float readFloat(int offset)	从块中的指定位置读出一个 float	读出数据后，将引用位置 1
	void writeFloat(int offset, float num)	从块中的指定位置写入一个 float	写入数据后，将引用位置 1，dirty 位置 1 以标记为脏数据
	String readString(int offset, int length)	从块中的指定位置读出一个长度为 length 的 String	读出数据后，将引用位置 1
	void writeString(int offset, String num, int length)	从块中的指定位置写入一个长度为 length 的 String	写入数据后，将引用位置 1，dirty 位置 1 以标记为脏数据

Buffer Manager 类

功能概述：

1. 根据需要，读取指定的数据(块)到系统缓冲区或将缓冲区中的数据写出到文件
2. 实现缓冲区的替换算法（LRU），当缓冲区满时选择合适的页进行替换

3. 记录缓冲区中各页的状态，如是否是脏数据等
4. 提供缓冲区页的 fix 功能，及锁定缓冲区的页，不允许替换出去，以提高效率。

类内各部分的功能描述及实现原理如下表所示：

	名称	功能描述	实现原理
成员变量	Block[] blocks	用一个 block 类数组代表 buffer,总共占用 80k 的空间	
	int pointer	用于实现时钟算法，用块在 buffer 中的下标代替指针	
外部接口	void initialize()	初始化。在使用 Buffer Manager 前调用一次	为 buffer 申请内存空间
	void close()	关闭 Buffer Manager,在退出程序之前调用	把 Buffer 中的脏数据写回文件
	Block getBlock(String filename, int blockoffset)	给定文件名和块编号，返回一个 block	调用 findBlock 搜索这个块是否在 buffer 中，是则返回这个 block。否则，调用 getFreeBlockNum 得到 buffer 中可用的一个块下标，把文件中的数据读入这个块，并返回这个块。
内部函数	int findBlock(String filename, int blockoffset)	给定文件名和块编号，返回一个 block 在 buffer 中的下标，如果不在 buffer 中则返回 -1	对 buffer 中的所有块进行遍历搜索。
	int getFreeBlockNum()	返回一个可被替换出去的 block 的下标。过程中使用时钟算法进行选择，并且跳过被锁定在 buffer 中的块。	将 pointer 指向 buffer 中下一个块。如果且 fixed 为 1，将 pointer 指向下一个块并进入下一个循环。否则，如果指向的块 reference_bit 为 1，则把 reference_bit 位置 1；如果指向的块 reference_bit 为 0，则把这个块写回文件，并返回 pointer 的值。具体算法见后文。
	boolean readFromDisk(String filename, int blockoffset, int num)	给定文件名，块偏移，把数据从文件读取到 buffer 中下标为 num 的块中，并对标记位进行初始化（有效位、reference_bit 置 1，dirty、fixed 位置 0）	通过 RandomAccessFile 执行文件读取
	void writeToDisk(int num)	把 buffer 中下标为 num 的	如果 dirty 位为 0，则不执

		块写回到文件中，并把块的有效位置 0	行写操作，否则执行写操作。
--	--	--------------------	---------------

以下对重要模块做更为详细的说明

时钟算法具体说明(getFreeBlockNum())

为实现近似 LRU 的的替换算法，使用时钟算法来进行 Buffer 中的块替换：
基本原理如下图所示。

- Arrange block into a cycle, store one **reference_bit** per block
- When **pin_count** reduces to 0, set **reference_bit=1**
- **reference_bit** as the 2nd chance bit



为实现能把块锁定在缓冲区的功能，给块增加 **fixed** 位，**fixed** 位为 1 的块的 **reference_bit** 在时钟算法中不会被置为 1。

具体实现代码如下：

```
private static int getFreeBlockNum() {
    do {
        pointer = (pointer + 1) % NUMOFBLOCKS;
        if (blocks[pointer].reference_bit == true
            && blocks[pointer].fixed == false)
            blocks[pointer].reference_bit = false;
        else if (blocks[pointer].reference_bit == false) {
            writeToDisk(pointer);
            return pointer;
        }
    } while (true);
}
```

Record Manager

代码结构：

RECORD MANAGER 包中包含三个类。tuple 类, ConditionNode 类以及 recordManager 类。

tuple 是用来存储单条记录的类。

ConditionNode 类是存储条件语句的类。

recordManager 类是 Record Manager 的核心，用于管理表。

tuple 类

功能描述：用于存储Table中的一条记录

实现原理：用一个 String 的 Vector 以字符串格式一个一个地存储每个属性对应的值。

其内部成员为：

```
public class tuple {  
    public Vector<String> units;  
}
```

conditionNode 类

整体功能描述：以二叉树结构记录并计算 sql 语句中的 where 条件语句。计算时输入一条 tuple,返回 true 表示符合条件,false 表示不符合条件。

支持以下基础功能：

条件语句中指定属性与常数比较

用 AND 连接条件语句。

支持以下扩展功能：

用 OR 连接语句。如：

```
select * from student2 where score>90 and id<=1080100003 or name='申辉幸';
```

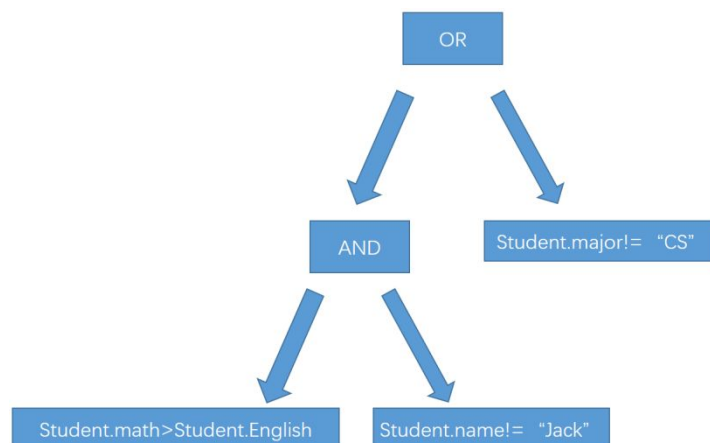
用括号指定优先级。如：

```
select * from student2 where score>90 and (id<=1080100003 or name='申辉幸');
```

同一个表中两个不同属性之间的数据进行比较。如：

```
select * from student2 where ChineseScore>MathScore;
```

实现原理概述：将 sql 语句的条件语句按照执行优先级存入二叉树，优先级越高的深度越大。如下图所示：



这棵二叉树所表示的是

where Student.major!= “CS” OR Student.math>Student.English AND Student.name!= “Jack”

计算时从根节点开始递归地计算出结果。详细计算方法见后文。

类内各部分的功能描述及实现原理如下表所示：

	名称	功能描述	实现原理
成员变量	String tablename	在非叶节点中为空。在叶节点中表示运算符左侧涉及的属性属于哪个表。	
	String attriName	在非叶节点中为空。在叶节点中表示运算符左侧涉及的属性的名字。	
	String tablename2	在非叶节点中为空。在叶节点中表示运算符右侧涉及的属性属于哪个表。如果运算符右侧为常数时该值为空。	
	String attriName2	在非叶节点中为空。在叶节点中表示运算符右侧涉及的属性的名字。如果运算符右侧为常数时该值为空。	
	String conjunction	在叶节点中为空。表示多个条件之间用 and 还是 or 来进行连接。	
	Comparison op	在非叶节点为空。在叶节点中表示运算符。	
	String value	在非叶节点中为空。在叶节点中如果运算符右侧为常数时，记录在该值中。如果运算符右侧涉及的是属性数据而不是常数时该值为空。	
	conditionNode left	左儿子	
	conditionNode right	右儿子	
	boolean constantFlag	如果是和常数比较则为 true，如果是和另一个 attribute 比较则置 false	
外部接口	conditionNode(String attriName, Comparison op, String value,boolean constantFlag)	用于构造属性与常量比较的叶节点	根据输入进行赋值
	conditionNode(String conjunction)	用于构造属性与属性之间比较的叶节点	根据输入进行赋值
	conditionNode linkChildNode(conditionNode l, conditionNode r) {	用于把一个父节点链接到两个叶节点	根据输入进行赋值
	boolean calc(String tablename)	输入一个 Tuple，判断这条	从根节点开始，递归地进

	ame, tuple T)	记录是否满足这个条件。	行计算。具体算法详见后文。
--	---------------	-------------	---------------

以下对重要模块做更为详细的说明

计算方法具体说明 (`boolean calc(String tablename, tuple T)`)

由于条件已按照优先级存储在树中，可用递归的方法进行计算。

对于非叶节点，计算方法如下：

```
if (conjunction.equals("and"))
    return (left.calc(tablename, T) & right.calc(tablename, T));
else if (conjunction.equals("or"))
    return (left.calc(tablename, T) | right.calc(tablename, T));
```

对于叶节点，根据运算符右侧是否是常量，可分为两类处理：

1. 属性值与常量比较：
根据属性类型把 tuple 中的对应数据从 String 转化为 int 或 float 或保持 String 类型。再把常量也从 String 转化为对应类型。再根据运算符进行比较。
2. 同一个表中两个不同属性值之间比较。
根据属性类型把 tuple 中的两个对应数据都从 String 转化为 int 或 float 或保持 String 类型。再根据运算符进行比较。

RecordManager 类

功能概述：

基于 **Free List**，对一个表中的所有记录按块进行存取。每个表存储在一个独立的文件中。支持以下基础功能：

select, delete, drop, create table, insert 的基本功能。

支持以下扩展功能：

排序功能: Order by 如：

```
select * from student2 order by score;
```

连接功能: Join。 如：

```
select * from student2 join student_department where
student2.id=student_department.id;
```

Projection 功能。如：

```
select name from student2;
```

整体存储结构：

表中第一条记录为表头。每条记录在真实数据开始前的 4 位存储一个 int 作为伪指针来实现 Free List。因为 JAVA 中没有指针，所以用 tupleOffset 的值来替代指针。表头指向的是第一个被删除后留下的空余位置，后面空余位置的伪指针指向下一个空余位置，以此类推。最后一个空余位置伪指针置 0 表示接地。非空余位置的伪指针置-1 表示该条记录有效。其结构示意图如下：



块内存储结构:

在每个块中存储尽可能多的记录。每个块能存储的记录个数计算公式如下:

$$\text{tupleNumInABlock} = \text{BLOCKSIZE} / (\text{SIZEINT} + \text{TupleLength})$$

其中 **SIZEINT** 代表伪指针占的4位。

根据 **tupleNumInABlock**, 给定 **tupleOffset** 即可定位出一条记录在文件中的块号及块内字节偏移。计算公式如下:

$$\begin{aligned} \text{tupleNumInABlock} &= \text{tupleOffset} / \text{tupleNumInABlock}; \\ \text{byteoffset} &= (\text{SIZEINT} + \text{TupleLength} * (\text{tupleOffset} \% \text{tupleNumInABlock})); \end{aligned}$$

类内各部分的功能描述及实现原理如下表所示:

名称	功能描述	实现原理
boolean createTable(String tableName)	给定表名, 创建表文件, 初始化表头。	调用 FileManager 创建文件。调用 bufferManager 在表头指针处写 0。
boolean dropTable(String tableName)	给定表名, 删除表文件。	调用 FileManager 删除文件。调用 bufferManager 把该表的脏数据清除掉。
int insert(String tablename, tuple Tuple)	给定表名, 及一个 tuple, 插入表中, 并返回所插入位置的 tupleOffset。	先从表头查询 FreeList, 如果有被删除后留下来的空位, 则插入表头指向的空位, 并让表头指向下一个空位。如果 FreeList 中没有空余, 则通过 Catalog Manager 获得表当前的 tuple 数量, 以此计算出表末尾的块号 blockOffset 及字节偏移 byteOffset, 插入对应位置。
Vector<tuple> project(Vector<tuple> res, String tablename, Vector<String> attriNames)	用于做 select 时的 projection。给定表名, 及要选出来属性名称, 输入 select 函数返回的结果, 返回经 projection 后的结果。	根据要选出来属性名称, 把 tuple 中没有被选择的属性值删去。

Vector<tuple> select(String tablename, conditionNode condition)	用于普通 select 语句。输入表名和判断条件，选出符合条件的记录存储在 Vector<tuple>中并返回	对表中的有效记录进行逐个遍历，读出记录至 tuple T 并使用 condition.calc(tuple T)去判断每条记录是否符合条件，符合则放入 Vector<tuple>中，遍历完后返回结果。
Vector<tuple> select(String tablename, conditionNode condition, String orderAttrName, boolean isInc)	用于带有排序功能 order 的 select 语句。输入表名、判断条件、排序依据属性名、升序排序或降序排序，选出符合条件的记录存储在 Vector<tuple>中，排序后返回	对表中的有效记录进行逐个遍历，读出记录至 tuple T 并使用 condition.calc(tuple T)去判断每条记录是否符合条件，符合则放入 Vector<tuple>中，遍历完后，调用 vector 中的 sort 函数进行排序，返回结果。
int delete(String tablename, conditionNode condition)	用于 delete 语句。输入表名和判断条件，删除符合条件的记录，返回被删除的记录数	对表中的有效记录进行逐个遍历，读出记录至 tuple T 并使用 condition.calc(tuple T)去判断每条记录是否符合条件，符合则把该记录的标记为标为已删除。
Vector<tuple> join(String tableName1, String attributeName1, String tableName2, String attributeName2)	用于 select * join from A.a=B.b 指令，指定两个表及其对应属性进行 join，返回对应结果。	用 select 函数获得两个表中的所有记录，用两重循环进行两两匹配，如果指定的属性值相等则加入结果中。
Vector<tuple> getTuple(String tablename, Vector<Integer>tupleOffsets)	给定表名和多个 tupleOffsets 所构成的 vector，返回对应多个 tuple 所构成的 vector。如果给定 tupleOffset 处数据为空或数据已被删除，则对应结果中为 null。	先根据 tupleOffset 和每个记录的长度，计算出对应记录在文件对应的块号 blockoffset 以及块内字节偏移 byteoffset，从对应块中读出数据，再根据每个 attribute 的类型进行转换，逐个存入 tuple 中。
tuple getTuple(String tablename, int tupleOffset)	给定表名和多个 tupleOffsets 所构成的 vector，返回对应多个 tuple 所构成的 vector。如果给定 tupleOffset 处数据为空或数据已被删除，则对应结果中为 null。	先根据 tupleOffset 和每个记录的长度，计算出对应记录在文件对应的块号 blockoffset 以及块内字节偏移 byteoffset，从对应块中读出数据，再根据每个 attribute 的类型进行转换，逐个存入 tuple 中。

内部类	<code>static class MyCompare implements Comparator<tuple></code>	用于带有排序功能 <code>order</code> 的 <code>select</code> 语句，通过指定要比较属性名称来进行比较。	通过属性名称取出数值进行比较。
-----	--	--	-----------------

File Manager

功能概述：用于创建、删除、管理文件。

	名称	功能描述	实现原理
对外接口	<code>void creatFile(String filename)</code>	给定文件名，创建文件	调用 <code>File.createNewFile()</code>
	<code>void dropFile(String filename)</code>	给定文件名，删除文件	调用 <code>File.delete()</code>
	<code>boolean findFile(String filename)</code>	给定文件名，判断文件是否存在	调用 <code>File.exist</code> 函数

IndexManager

代码模块：

`IndexManger` 包中包含三个类。`BPlusTree`、`offsetInfo` 类以及 `IndexManager` 类。

`BPlusTree` 类是用来包装新建或者已存在的索引结构的 `B+` 树结构类。并且为 `IndexManager` 提供 `B+` 树的更删改查等维护功能。

`IndexManager` 类是用来给 `API` 提供相应的索引函数接口，比如索引的建立、删除、插入等操作。

`offsetInfo` 是用来记录 `B+` 树中的键值的数据结构。

下面详细介绍各个模块内的功能。

IndexManager 类

功能：作为封装好的对索引进行操作的模块，通过 `API` 提供给 `Interpreter` 进行调用。类内各部分的功能描述及实现原理如下表所示：

	名称	功能描述	实现原理
方法描述	<code>boolean createIndex(index indexInfo)</code>	用于针对一个表及指定属性创建索引	返回布尔变量，表示创建是否成功
	<code>boolean dropIndex(String filename)</code>	用于针对文件名字删除指定索引	返回布尔变量，表示创建是否成功

	<code>void deleteKey(index indexInfo,String deleteKey)</code>	用于删除指定索引中的指定键值	无返回值
	<code>Integer searchEqual(index indexInfo, byte[] key)</code>	用于等值的查找，通过索引查找指定键值的位置	返回所要找的 tuple 在文件中的偏移量
	<code>Vector<Integer> searchRange(index indexInfo,String startkey, String endkey)</code>	用于范围的查找，通过索引范围来查找在范围内的一系列键值的位置	返回所要找的一系列 tuple 在文件中的偏移量数组
	<code>void insertKey(index indexInfo,String key,int blockOffset,int offset)</code>	用于将键值插入指定的索引中。	无返回值
	<code>byte[] StringInttoByte(String num)</code>	将用字符串表示的整数转换为 Byte 型数组	返回转换后的 Byte 型数组
	<code>byte[] StringFloattoByte(String num)</code>	将用字符串表示的浮点数转换为 Byte 型数组	返回转换后的 Byte 型数组

BPlusTree 类

功能：作为对硬盘上的索引文件抽象成 B+树结构的类供 IndexManager 调用，同时可以对物理存储上的索引文件进行维护：

	名称	功能描述	实现原理
成员变量	<code>final int POINTERLENGTH = 4</code>	常量，用于记录指针的长度，由于使用的是整形所以长度是 4	
	<code>final double BLOCKSIZE = 4096.0</code>	常量，用于记录一个树节点的大小，默认是 4KB	
	<code>int MIN_CHILDREN_FOR_INTERNAL;</code> <code>int MAX_CHILDREN_FOR_INTERNAL;</code> <code>int MIN_FOR_LEAF;</code> <code>int MAX_FOR_LEAF;</code>	常量，用于记录中间节点的最小最大路标个数，以及叶子节点的最小最大索引个数。	
	<code>String filename;</code>	索引文件名	
	<code>Block myRootBlock;</code>	包装好的根块 Block 信息体	
	<code>index myindexInfo;</code>	索引的信息体，由外部传入，可实时更新	
方	<code>BPlusTree(index index</code>	构造函数，用于从无到有	①通过 FileManager 新建索

法 描 述	<code>Info)</code>	地生成新的一颗 B+树。	引文件, 并通过 BufferManager 获取第一块节点大小的 Block ; ②实时计算树的叉数; ③初始化根节点。
	<code>BPlusTree(index index Info,int rootBlockNum)</code>	构造函数, 用于读取已存在的索引并包装成 B+树。	①计算树的叉数; ②通过 BufferManager 以及根块的位置获取根节点的 Block ; ③初始化根节点。
	<code>void insert(byte[] originalkey,int blockOffset, int offset)</code>	插入函数, 以树为单位的插入。	①判断根块的属性并用中间节点和叶子节点的构造方法进行包装; ②调动节点的插入方法进行插入操作; ③判断节点的返回值, 如果有返回值说明根块有更新, 要更新树的 rootNum 变量。
	<code>offsetInfo searchKey (byte[] originalkey)</code>	等值查找函数, 以树为单位的范围查找。返回一个保存返回信息的类结构。	①判断根块的属性并用中间节点和叶子节点的构造方法进行包装; ②调动节点的查找方法进行查找。
	<code>offsetInfo searchKey (byte[] originalkey,byte[] endkey)</code>	范围查找函数, 以树为单位的范围查找。返回一个保存返回信息的类结构。	①判断根块的属性并用中间节点和叶子节点的构造方法进行包装; ②调动节点的查找方法进行查找。
	<code>void delete(byte[] originalkey)</code>	删除函数, 以树为单位的删除。删除一个树中与给定值相等的索引值。	①判断根块的属性并用中间节点和叶子节点的构造方法进行包装; ②调动节点的删除方法进行插入操作; ③判断节点的返回值, 如果有返回值说明根块有更新, 要更新树的 rootNum 变量。

其中 BPlusTree 类还包含了一个 Node 的节点虚类, 它有 InternalNode 和 LeafNode 两个子类。它们的功能为对节点为单位的更删改查以及节点调整中的删除、合并、增添的操作。具体的函数如下所示:

名称	功能描述	实现原理
抽象类 Node		

成员变量	<code>Block block;</code>	存放该节点对应磁盘中的 Block 块。	
方法描述	<code>Node createNode(Block blk)</code>	初始化成员变量 block。	直接通过参数赋值。
	<code>abstract Block insert(byte[] inserKey, int blockOffset, int offset)</code>	抽象方法：插入函数	
	<code>abstract Block delete(byte[] deleteKey)</code>	抽象方法：删除函数	
	<code>abstract offsetInfo searchKey(byte[] Key)</code>	抽象方法：等值查找函数	
	<code>abstract offsetInfo searchKey(byte[] skey, byte[] ekey)</code>	抽象方法：范围查找函数	
	<code>int compareTo(byte[] buffer1, byte[] buffer2)</code>	比较函数，定义统一的根据 Byte 型数组互相之间的比较，能够适配所有的树内索引排序。	和字典排序类似，只需要从开始比较数组中两个数的大小关系，并在不等的时候返回大小关系。
class InternalNode extends Node 中间节点类			
方法描述	<code>InternalNode(Block blk)</code>	构造函数，申请新建一个新的块。	对新块进行信息头的标记。具体可以看后面 B+树的数据结构。
	<code>InternalNode(Block blk, boolean t)</code>	构造函数，已有的块进行包装。	对参数 block 赋值。
	<code>Block insert(byte[] insertKey, int blockOffset, int offset)</code>	索引插入函数，对指定的键值进行插入。	查找键值所在的路标，然后递归到下一个子节点调用它的插入函数。如果根块有更新，会返回一个指针，如果没有返回 null。
	<code>Block branchInsert(byte[] branchKey, Node leftChild, Node rightChild)</code>	路标插入函数，供子节点分裂后调用插入路标。插入过程中有时需要对中间节点进行分支，branchKey 为要新插入的路标，leftChild 为新路标的左子节点(也就是已经存在的节点)，rightChild 为新路标的右子节点	①如果要分裂的情况下要申请新的节点空间，并分配好路标，同时继续往上调用 branchInsert 进行插入更新； ②对该节点的插入位置键值、插入左右子节点、节点总路标数更新。
	<code>offsetInfo searchKey</code>	以中间节点为单位的等值	遍历整个节点，找到该键

	<code>(byte[] key)</code>	查找	值后递归调用子节点的 <code>searchKey</code> 方法。
	<code>offsetInfo searchKey (byte[] skey, byte[] ekey)</code>	以中间节点为单位的范围查找	只是提供给范围查找调用，方法与上一个完全一样。
	<code>Block delete(byte[] deleteKey)</code>	以中间节点为单位的删除	与查找类似，递归调用子节点删除方法。
	<code>Block union(byte[] unionKey, Block afterBlock)</code>	删除过程中产生的节点合并， <code>this</code> 块和 <code>after</code> 块以及它们之间的 <code>unionKey</code>	合并两个节点，并找到合并的路标，调用父节点进行块删除。
	<code>byte[] rearrangeAfter (Block siblingBlock, byte[] InternalKey)</code>	删除过程中产生的兄弟块内容重排， <code>this</code> 块和 <code>after</code> 块以及它们之间的 <code>internalKey</code> ，返回的 <code>changeKey</code> 是为了更新父块中它们两指针中间的键值。这是兄弟节点在其后的方法。	①找到兄弟节点要转移的一条指针内容； ②将 <code>internalKey</code> 和兄弟块的第一条指针复制到 <code>this</code> 块的尾部，路标数加 1； ③兄弟块的路标数减 1，获取兄弟块的第一条键值作为更新父块的键值，再将兄弟块后面的信息调整
方法描述	<code>byte[] rearrangeBefore (Block siblingBlock, byte[] internalKey)</code>	同上，是兄弟节点在其前的方法。	同上。
	<code>void exchange(byte[] changeKey, int posBlockNum)</code>	修改 <code>posBlockNum</code> 标号后面的路标。	直接修改对应位置的键值。
	<code>Block delete (Block blk)</code>	在中间节点中删除一个子块信息（以及它前面的那条路标）	删除后查看是否需要调整中间节点进行合并，如果需要则向父亲查找前后兄弟节点进行 <code>rearrange</code> ；不需要合并则直接结束。
class LeafNode extends Node 叶子节点类			
方法描述	<code>LeafNode (Block blk)</code>	构造函数，申请新建一个新的块。	对新块进行信息头的标记。具体可以看后面 B+树的数据结构。
	<code>LeafNode (Block blk, boolean t)</code>	构造函数，已有的块进行包装。	对参数 <code>block</code> 赋值。
	<code>Block insert (byte[] insertKey, int blockOffset, int offset)</code>	索引插入函数，对指定的键值进行插入。	查找键值所在的路标并插入，同时判断是否需要分裂叶子节点。如果需要更新，会返回一个根块指针，如果没有返回 <code>null</code> 。
	<code>offsetInfo searchKey (byte[] key)</code>	以叶子节点为单位的等值查找。	遍历整个节点，找到该键值后返回该记录的偏移

			量，如果需要合并或借值再分别判断完成。
offsetInfo searchKey (byte[] skey, byte[] ek ey)	以叶子节点为单位的范围查找。		只是提供给范围查找调用，方法基本与上一个完全一样。找到 skey 后往后遍历找到最后不满足为止，返回记录偏移量的数组。
Block delete(byte[] deleteKey)	以叶子节点为单位的删除		与查找类似，递归调用子节点删除方法。
Block union(byte[] unionKey, Block afterBlock)	删除过程中产生的节点合并，this 块和 after 块以及它们之间的 unionKey		合并两个节点，并找到合并的路标，调用父节点进行块删除。
byte[] rearrangeAfter (Block siblingBlock, byte[] InternalKey)	删除过程中产生的兄弟块内容重排，this 块和 after 块以及它们之间的 internal Key, 返回的 changeKey 是为了更新父块中它们两指针中间的键值。这是兄弟节点在其后的方法。		①找到兄弟节点要转移的一条指针内容； ②将 internalKey 和兄弟块的第一条指针复制到 this 块的尾部，路标数加 1； ③兄弟块的路标数减 1，获取兄弟块的第一条键值作为更新父块的键值，再将兄弟块后面的信息调整
byte[] rearrangeBefore (Block siblingBlock, byte[] internalKey)	同上，是兄弟节点在其前的方法。		同上。

B+树结构

B+树采用实时计算的特点得出最终每个节点的最大最小路标数和最大最小索引树。其中每个节点的结构如下所示：

长度	1	4	4	4	4	记录长度	...	4
类型	标记	个数	偏移	偏移	偏移	键值	...	偏移
Internal Node	I	路标个数	父亲节点偏移	子块偏移		路标值	...	子块偏移
Leaf Node	L	索引个数	父亲节点偏移	记录文件偏移量	记录块内偏移量	索引键值	...	尾指针(兄弟节点)

对应的每个节点都用 Byte[] 型数组来进行存储，长度均为 Byte 型，这样可以保证适配所有类型属性的索引建立。同时使用文件内的偏移量来代表数组，这样保证每次搜寻时都需要通过 BufferManager 去 Disk 里寻找新的节点块，保证了内存方面的管理。

3. 测试结果

A 功能测试

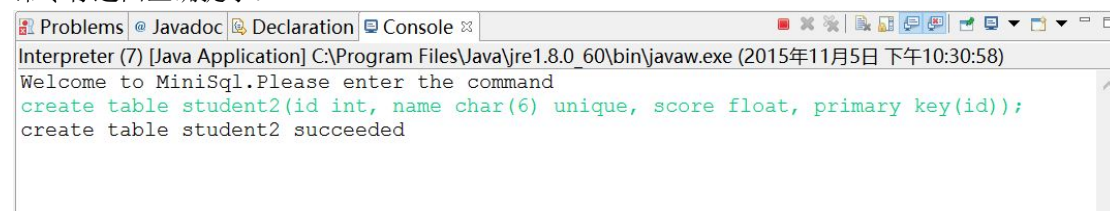
Create table 功能

测试语句：



```
create table student2(id int, name char(6) unique, score float, primary  
key(id));
```

测试结果：

命令行返回正确提示：



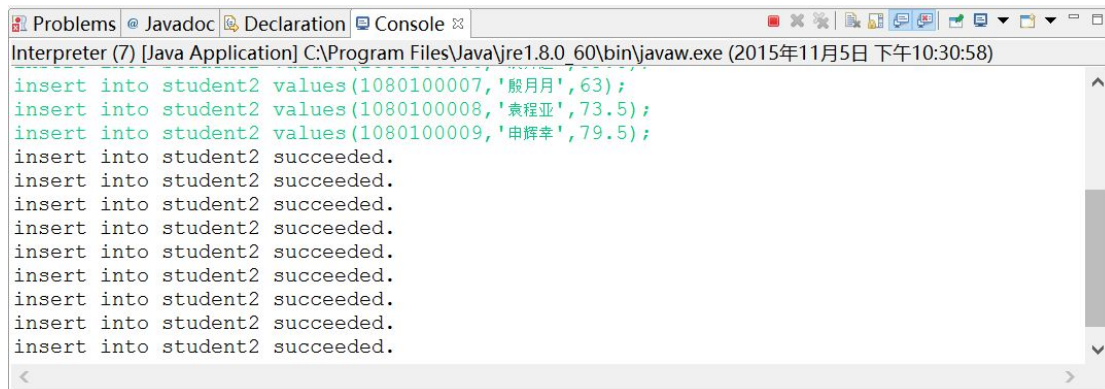
记录文件及索引文件被创建

 student2	2015/11/5 22:31	文件	0 KB
 student2_prikey.index	2015/11/5 22:31	INDEX 文件	0 KB

Insert 功能

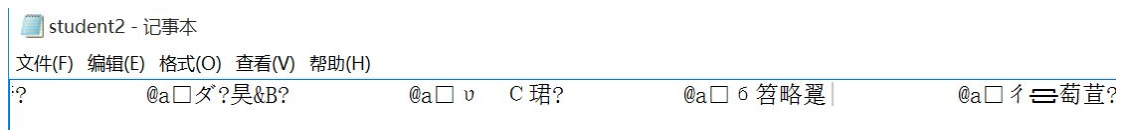
```
insert into student2 values(1080100001,'杨豪琛',99);  
insert into student2 values(1080100002,'楼河加',52.5);  
insert into student2 values(1080100003,'袁河辉',98.5);  
insert into student2 values(1080100004,'陈明分',91.5);  
insert into student2 values(1080100005,'劳 昊',72.5);  
insert into student2 values(1080100006,'袁帅超',89.5);  
insert into student2 values(1080100007,'殷月月',63);  
insert into student2 values(1080100008,'袁程亚',73.5);  
insert into student2 values(1080100009,'申辉幸',79.5);
```

命令行返回正确提示：



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:30:58)
insert into student2 values(1080100007,'殷月月',63);
insert into student2 values(1080100008,'袁程亚',73.5);
insert into student2 values(1080100009,'申辉幸',79.5);
insert into student2 succeeded.
insert into student2 succeeded.
insert into student2 succeeded.
insert into student2 succeeded.
insert into student2 succeeded.
insert into student2 succeeded.
insert into student2 succeeded.
insert into student2 succeeded.
insert into student2 succeeded.
```

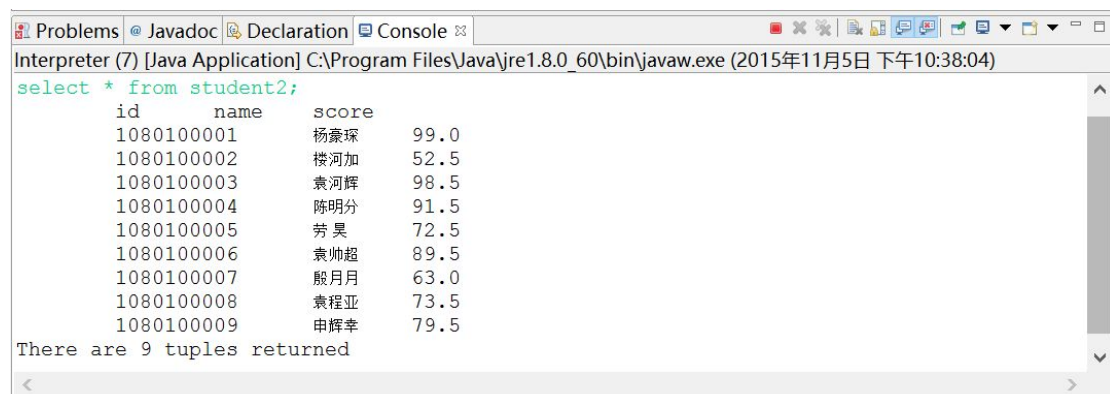
Record 及 index 文件中产生相应记录：（由于是以二进制插入文件，所以显示为乱码）



Select 功能

1. 基础功能

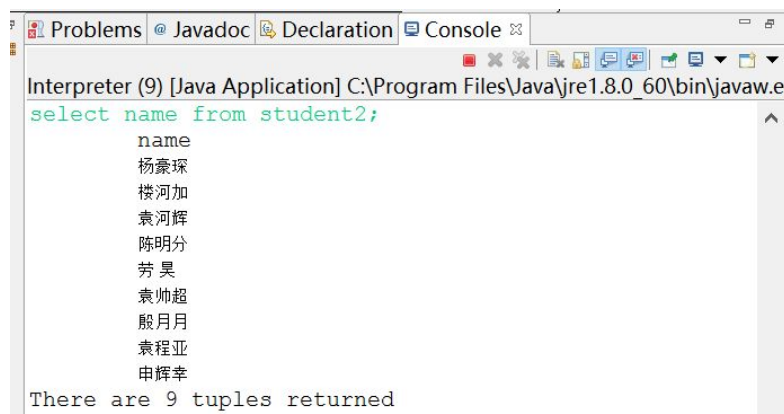
Select * 操作: `select * from student2 where score>90;`



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:38:04)
select * from student2;
  id      name      score
1080100001 杨豪琛    99.0
1080100002 楼河加    52.5
1080100003 袁河辉    98.5
1080100004 陈明分    91.5
1080100005 劳昊      72.5
1080100006 袁帅超    89.5
1080100007 殷月月    63.0
1080100008 袁程亚    73.5
1080100009 申辉幸    79.5
There are 9 tuples returned
```

2. 【BONUS 功能】Projection 功能

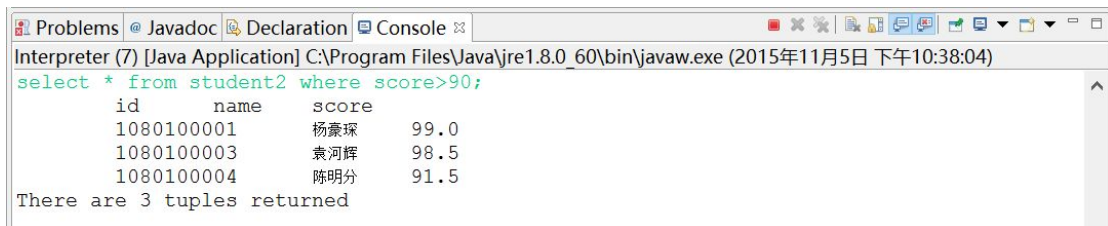
`select name from student2;`



```
Interpreter (9) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.e
select name from student2;
  name
 杨豪琛
 楼河加
 袁河辉
 陈明分
 劳昊
 袁帅超
 殷月月
 袁程亚
 申辉幸
There are 9 tuples returned
```

3. 条件判断功能

3.1 普通条件语句: `select * from student2 where score>90;`



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:38:04)

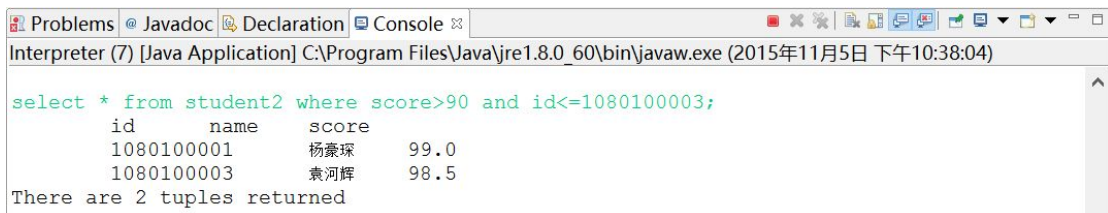
select * from student2 where score>90;

  id      name      score
1080100001  杨豪琛    99.0
1080100003  袁河辉    98.5
1080100004  陈明分    91.5

There are 3 tuples returned
```

带 and 的条件语句:

`select * from student2 where score>90 and id<=1080100003;`



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:38:04)

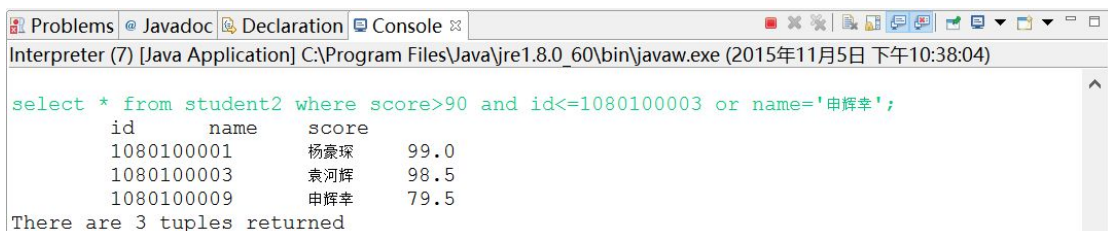
select * from student2 where score>90 and id<=1080100003;

  id      name      score
1080100001  杨豪琛    99.0
1080100003  袁河辉    98.5

There are 2 tuples returned
```

3.2 【BONUS 功能】带有 or 的条件语句

`select * from student2 where score>90 and id<=1080100003 or name='申辉幸';`



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:38:04)

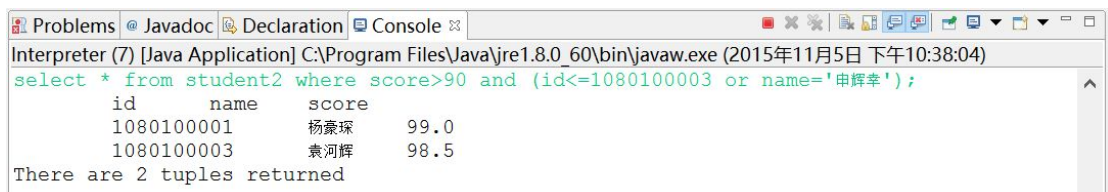
select * from student2 where score>90 and id<=1080100003 or name='申辉幸';

  id      name      score
1080100001  杨豪琛    99.0
1080100003  袁河辉    98.5
1080100009  申辉幸    79.5

There are 3 tuples returned
```

3.3 【BONUS 功能】带有括号的条件语句

`select * from student2 where score>90 and (id<=1080100003 or name='申辉幸');`



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:38:04)

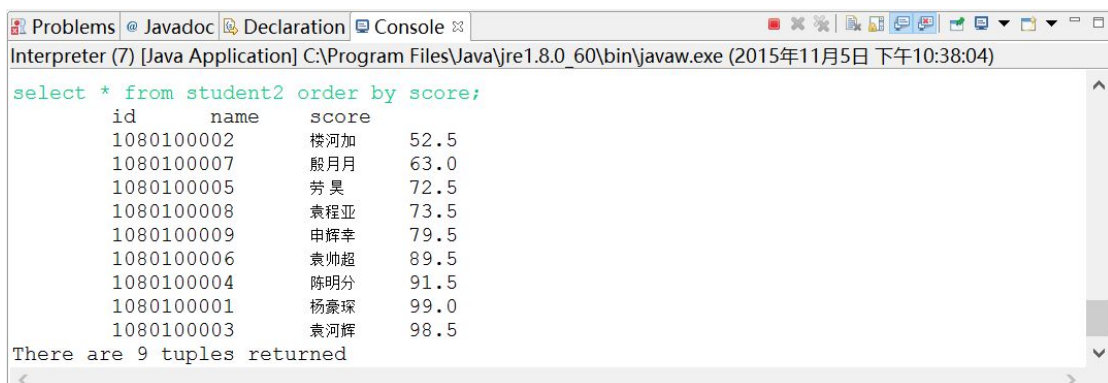
select * from student2 where score>90 and (id<=1080100003 or name='申辉幸');

  id      name      score
1080100001  杨豪琛    99.0
1080100003  袁河辉    98.5

There are 2 tuples returned
```

4. 【BONUS 功能】Order by 排序功能

`select * from student2 order by score;`



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:38:04)

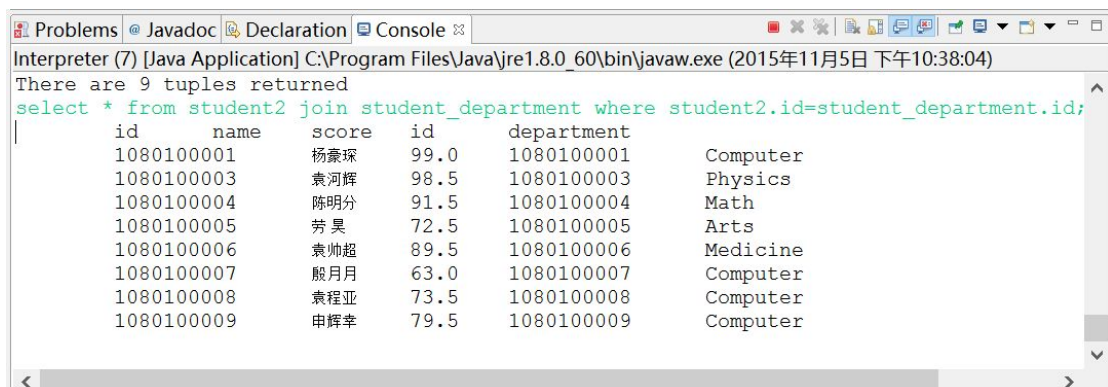
select * from student2 order by score;

  id      name      score
1080100002  楼河加    52.5
1080100007  殷月月    63.0
1080100005  劳昊      72.5
1080100008  袁程亚    73.5
1080100009  申辉幸    79.5
1080100006  袁帅超    89.5
1080100004  陈明分    91.5
1080100001  杨豪琛    99.0
1080100003  袁河辉    98.5

There are 9 tuples returned
```


5. 【BONUS 功能】Join 功能

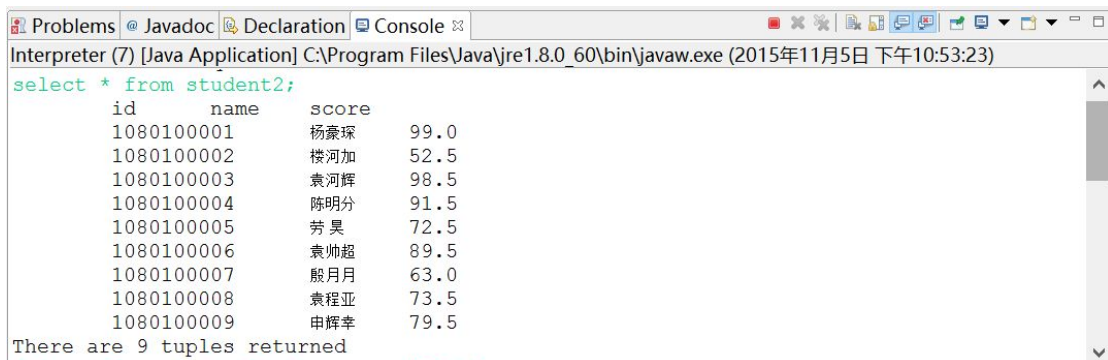
```
select * from student2 join student_department where student2.id=student_department.id;
```



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:38:04)
There are 9 tuples returned
select * from student2 join student_department where student2.id=student_department.id;
|      id      name      score      id      department
|-----|-----|-----|-----|-----|
| 1080100001  杨鑫琛    99.0    1080100001  Computer
| 1080100003  袁河辉    98.5    1080100003  Physics
| 1080100004  陈明分    91.5    1080100004  Math
| 1080100005  芳昊      72.5    1080100005  Arts
| 1080100006  袁帅超    89.5    1080100006  Medicine
| 1080100007  殷月月    63.0    1080100007  Computer
| 1080100008  袁程亚    73.5    1080100008  Computer
| 1080100009  申辉幸    79.5    1080100009  Computer
```

6. Delete 功能

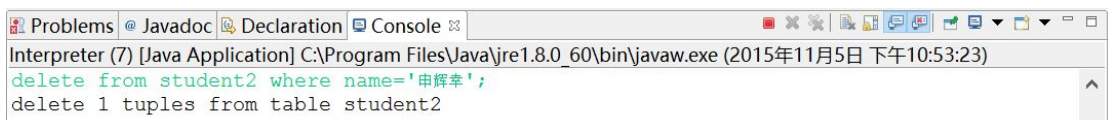
删除前:



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:53:23)
select * from student2;
|      id      name      score
|-----|-----|-----|
| 1080100001  杨鑫琛    99.0
| 1080100002  楼河加    52.5
| 1080100003  袁河辉    98.5
| 1080100004  陈明分    91.5
| 1080100005  芳昊      72.5
| 1080100006  袁帅超    89.5
| 1080100007  殷月月    63.0
| 1080100008  袁程亚    73.5
| 1080100009  申辉幸    79.5
There are 9 tuples returned
```

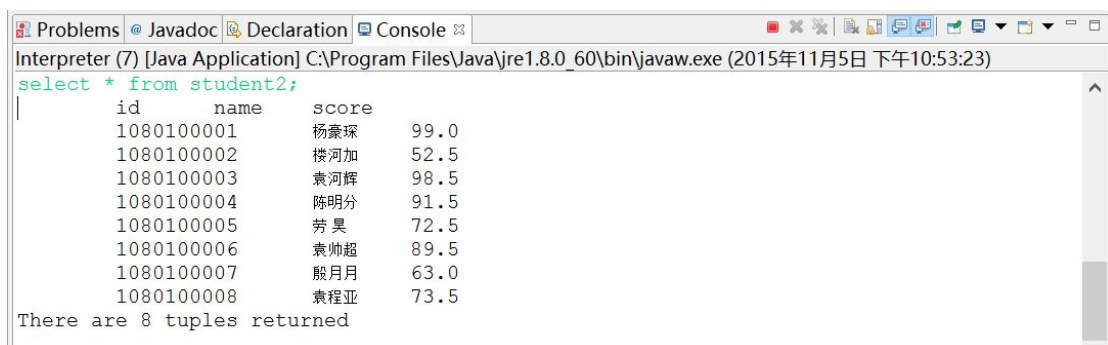
删除操作:

```
delete from student2 where name='申辉幸';
```



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:53:23)
delete from student2 where name='申辉幸';
delete 1 tuples from table student2
```

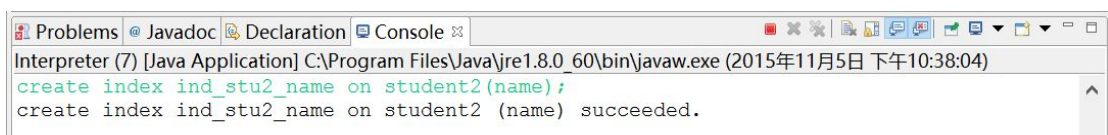
删除后:



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:53:23)
select * from student2;
|      id      name      score
|-----|-----|-----|
| 1080100001  杨鑫琛    99.0
| 1080100002  楼河加    52.5
| 1080100003  袁河辉    98.5
| 1080100004  陈明分    91.5
| 1080100005  芳昊      72.5
| 1080100006  袁帅超    89.5
| 1080100007  殷月月    63.0
| 1080100008  袁程亚    73.5
There are 8 tuples returned
```

7. Build index 功能

```
create index ind_stu2_name on student2 (name);
```



```
Interpreter (7) [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015年11月5日 下午10:38:04)
create index ind_stu2_name on student2 (name);
create index ind_stu2_name on student2 (name) succeeded.
```

index 文件中:



8. Drop table

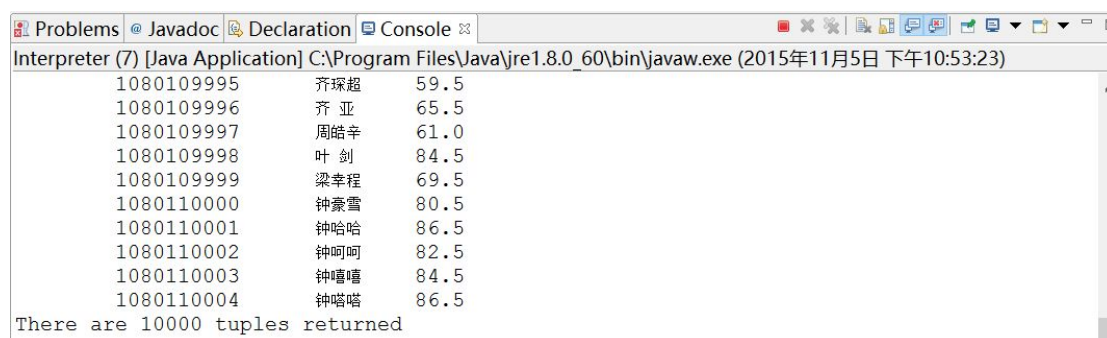
```
drop table student_department;
```



对应的 record 文件及 index 文件都被删除。

9. 大数据量测试

插入一万条数据，多次测试上述功能，无故障。



B 语法错误测试

```
create table t2
(
num1 int ,
num2 int
);
Synthetic error: no primary key defined

insert into aaa values
('33333333','abababababababab',18,m,4.5);
Synthetic error near: m

select * from table aaa;
Synthetic error near: table

delete id from aaa;
Synthetic error near: id
```

C 语义错误测试

对于以下现有的表及索引信息：

Welcome to MiniSql. Please enter the command

```
show tables;
```

There are 1 tables in the database:

Table 1

Table name: aaa

Number of Columns: 5

Primary key: sno

Number of tuples: 0

Index keys: 1

	Index name	Table name	Attribute name:
	aaa_prikey	aaa	sno

Attributes: 5

	Attribute name	Type	length	isUnique
	sno	char	8	true
	sname	char	16	true
	sage	int	4	false
	sgender	char	1	false
	smoney	float	4	false

```
show indexes;
```

There are 2 indexes in the database:

	Index name	Table name	Attribute name:
1	aaa_prikey	aaa	sno
2	ab	aaa	sname

创建表

```
create table aaa (  
    sno char(8),  
    sname char(16) unique,  
    sage int,  
    sgender char (1),  
    smoney float,  
    primary key ( sno )  
);  
The table aaa already exists, create table aaa failed
```

```
create table aa (  
    sno char(8),  
    sname char(16) unique,  
    sage int,  
    primary key ( sn )  
);  
The attribute sn doesn't exist, create table aa failed
```



```
create table aa (
    sno char(8),
    sname char(16) unique,
    sgender char (1),
    sgender char (1),
    sgender float,
    primary key ( sno )
);
```

Duplicated attribute names sgender, create table aa failed

删除表

```
drop table ac;
```

The table ac doesn't exist, drop table ac failed

创建索引

```
create index ac on aaa(sgender);
```

The attribute sgender on aaa is not unique, create index failed

```
create index ac on aab(sname);
```

The table aab doesn't exist, create index failed

```
create index ab on aaa(sname);
```

The index ab already exist, create index failed

```
create index ac on aaa(sname);
```

The attribute sname on aaa is already an index, create index failed

删除索引

```
drop index acc;
```

The index acc doesn't exist, drop index acc failed

插入记录

```
insert into aaa values
```

```
('33333333','abababababababab',18,6,4.5);
```

The type of value +6 should be char(1), not be int, insert failed

```
insert into aaa values
```

```
('33333333','abababababababab',18,'m',4.5,'m');
```

The number of values is larger than that of attributes, insert failed

```
select * from aaa;
```

sno	sname	sage	sgender	smoney
33333334	abababababababab	18	m	4.5

There are 1 tuples returned

```
insert into aaa values
```

```
('33333333','abababababababab',18,m,4.5);
```

The value abababababababab already exists in the unique attribute sname, insert failed

查询语句

对于表

```
Table 2
Table name: student_department
Number of Columns: 2
Primary key: id
Number of tuples: 33
Index keys: 1
      Index name      Table name      Attribute name:
      student_department_prikey      student_department      id
Attributes: 2
      Attribute name  Type      length  isUnique
      id              int       4        true
      department      char     20       false

select * from student_department where id='1080100017';
The type of value +1080100017 should be int(4), not char(10), select tuples failed
select * from student_department where id='108010017';
The attribute id doesn't exist, select tuples failed

select * from student_department where id>108010017 order by id;
The attribute id doesn't exist, select tuples failed

select * from student_department where id>department;
The two attributes are in different types and cannot be compared, select tuples failed
```

4. 成员分工

小组的成员分工以及工作量统计表如下所示：

姓名	负责模块	代码行数	总代码量	报告分工
丘颖悦	API	255	2018	负责代码模块描述
	Interpreter	1213		语法测试
	CatalogManager	550		
叶俊利	RecordManager	716	1155	负责代码模块描述
	BufferManager	384		功能测试
	FileManager	55		
郑濡樟	IndexManager	1349	1349	负责代码模块描述
				系统架构
				整合