



# 移动TotemDB

武汉大学计算机学院 | 珞珈图腾数据库实验室

<https://github.com/whu-totemdb/tmdb>

# 目录

## CONTENTS

01

系统介绍

02

核心功能

03

实验题目

04

参考资料



# 移动端对象数据库的重要性

<https://realm.io/>

<https://github.com/realm/realm-java>

<https://objectbox.io/mobile-database/>

<https://github.com/objectbox/objectbox-java>

Kotlin Swift JavaScript C#

```
// Your object classes are your database schema definition
open class Dog(
    var name: String = "",
    var age: Int = 0
): RealmObject()

open class Person(
    @PrimaryKey var _id: String = "",
    var name: String = "",
    var age: Int = 0,
    // Create relationships by pointing an Object field to another Object
    var dogs: RealmList<Dog> = RealmList()
): RealmObject()
```

Java

Kotlin

Swift

Flutter Dart


```
1 @Entity public class Person {
2     @Id long id;
3     String firstName;
4     ...
5 }
6
7 // MyObjectBox is generated by ObjectBox
8 BoxStore boxStore = MyObjectBox.builder()
9     .androidContext(YourApp.this).build();
10
11 Box<Person> box = boxStore.boxFor(Person.class);
12
13 Person person = new Person("Joe", "Green");
14 long id = box.put(person); // Create
15 Person person = box.get(id); // Read
16 person.setLastName("Black");
17 box.put(person); // Update
18 box.remove(person); // Delete
```



# 01 移动 Totem D B



<https://github.com/whu-totemdb/tmdb>



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)

[whu-totemdb/tmdb](#) Private

[Unwatch 1](#) [Fork 0](#) [Star 0](#)














[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#) [Settings](#)

main

1 branch


0 tags

[Go to file](#) [Add file](#) [Code](#)

 <b>rp4ps</b> new readme	d61e1f6 24 days ago	2 commits
 <b>app</b>	first commit	24 days ago
 <b>gradle/wrapper</b>	first commit	24 days ago
 <b>CONTRIBUTING.md</b>	first commit	24 days ago
 <b>README.md</b>	new readme	24 days ago
 <b>build.gradle</b>	first commit	24 days ago
 <b>db.iml</b>	first commit	24 days ago
 <b>gradle.properties</b>	first commit	24 days ago
 <b>gradlew</b>	first commit	24 days ago
 <b>gradlew.bat</b>	first commit	24 days ago
 <b>local.properties</b>	first commit	24 days ago
 <b>readme.txt</b>	first commit	24 days ago
 <b>settings.gradle</b>	first commit	24 days ago


☰


README.md





### About

Totem Mobile Database

 [Readme](#)

 0 stars

 1 watching

 0 forks

### Releases

No releases published

[Create a new release](#)

### Packages

No packages published

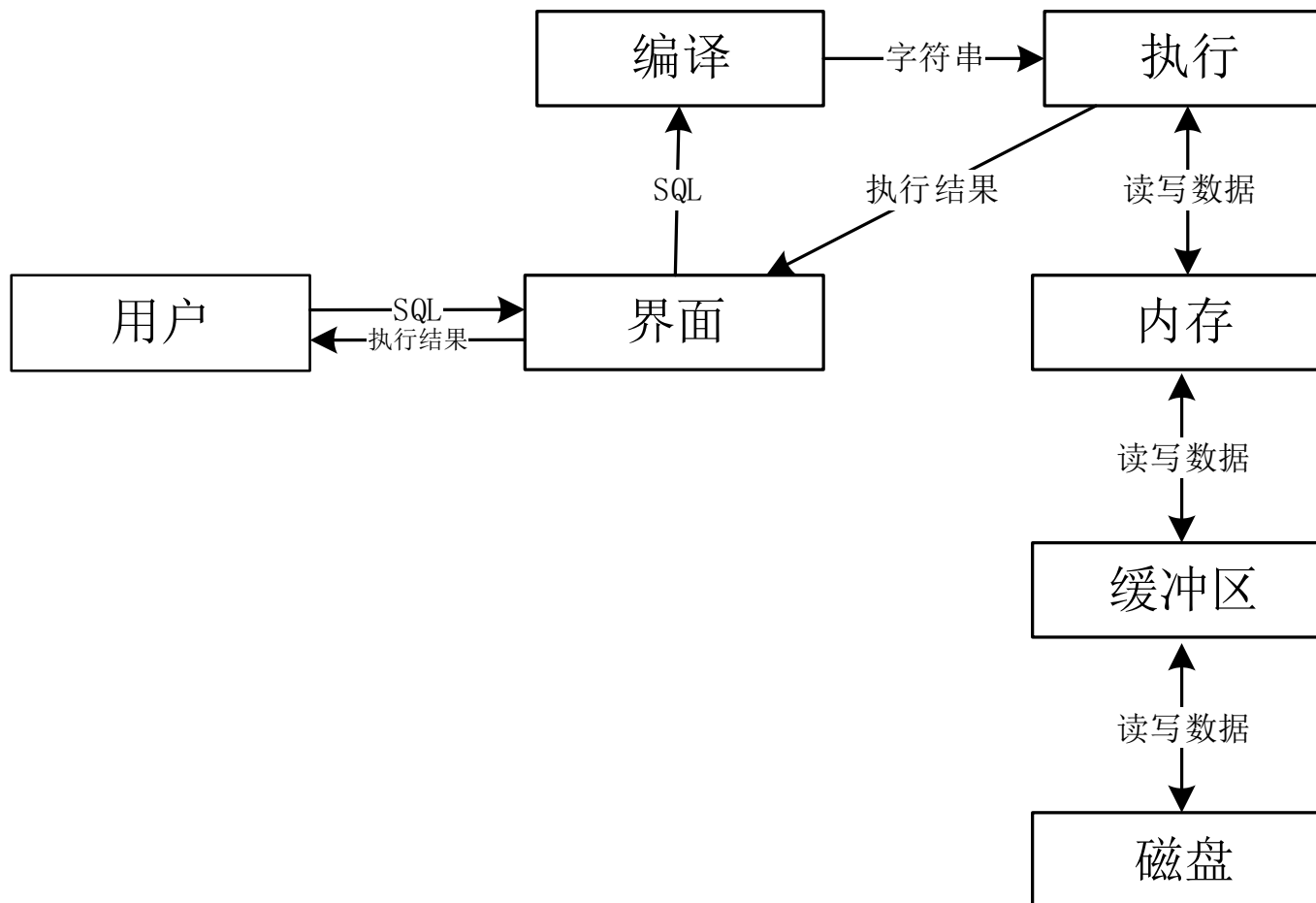
[Publish your first package](#)

### Languages

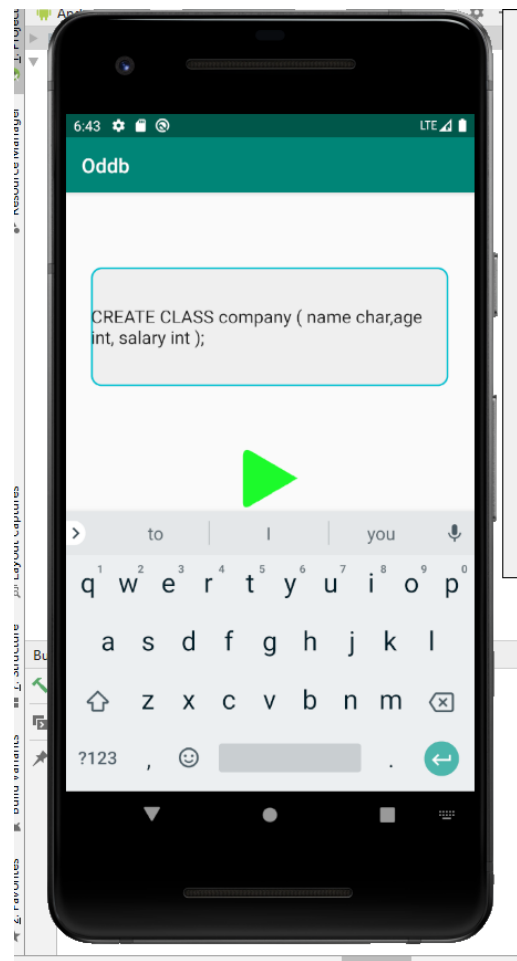
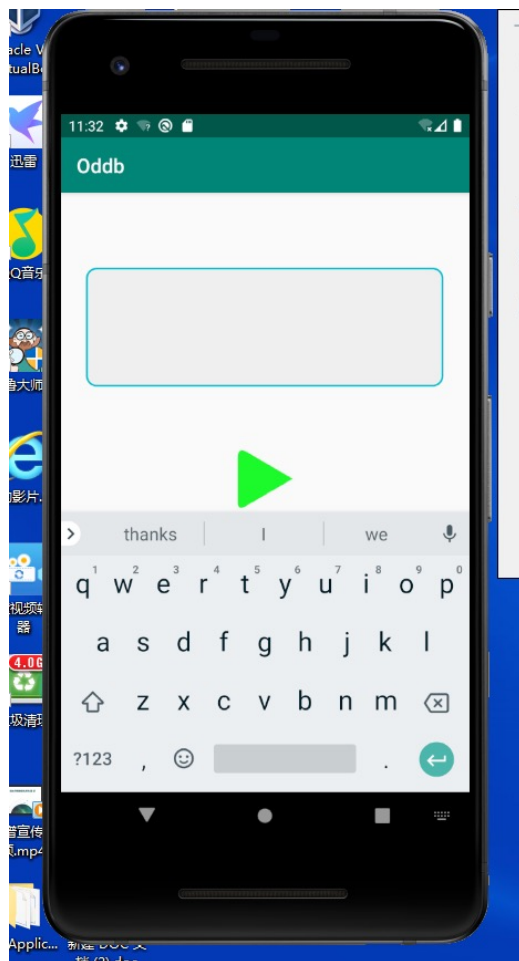
Java 100.0%

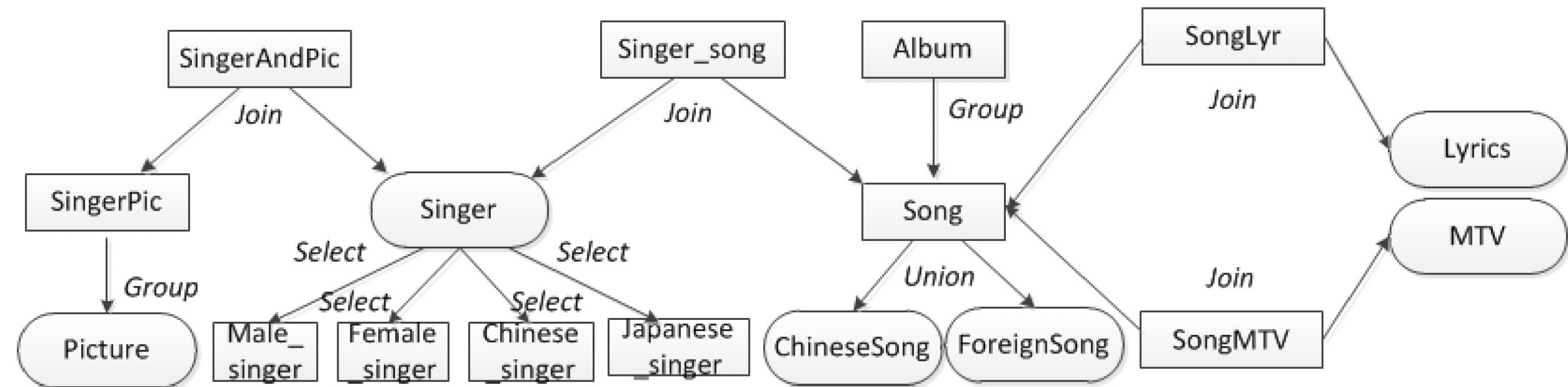


# 系统架构



# 界面









# 已有功能

- 存储
- SQL编译
- 选择代理
- 基本查询
- 更新迁移

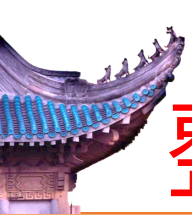


存

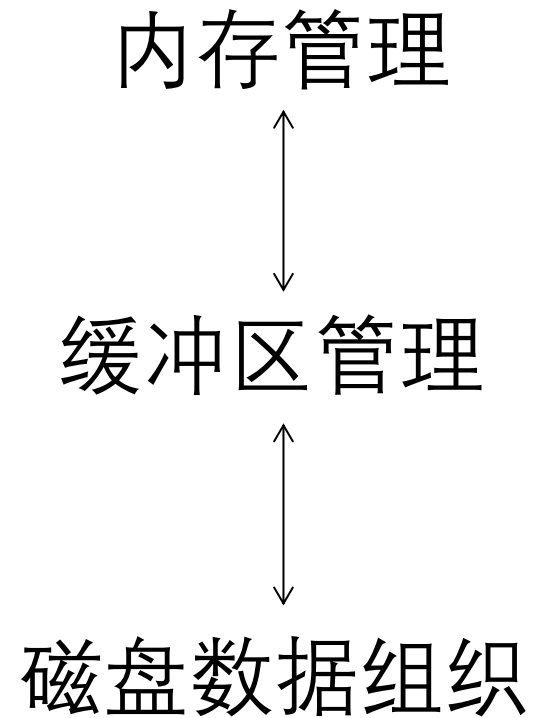
02

储





# 整体架构



系统表设计

数据结构

执行流程



# 系统表设计

## Class 系统表

classname	类名
classid	类id
attrnum	类属性个数
classtype	类类型

## Attribute表

classid	类id
attrid	属性id
attrname	属性名
attrtype	属性类型
lsdeputy	是否为虚属性



# 系统表设计

Deputy表

originid	类id
deputyid	代理类id
deputyruleid	代理规则id

DeputyRule表

deputyruleid	代理规则id
deputyrule	代理规则

Object表

classid	类id
tupleid	元组id
blockid	块id
offset	块内偏移



# 系统表设计

Switching表

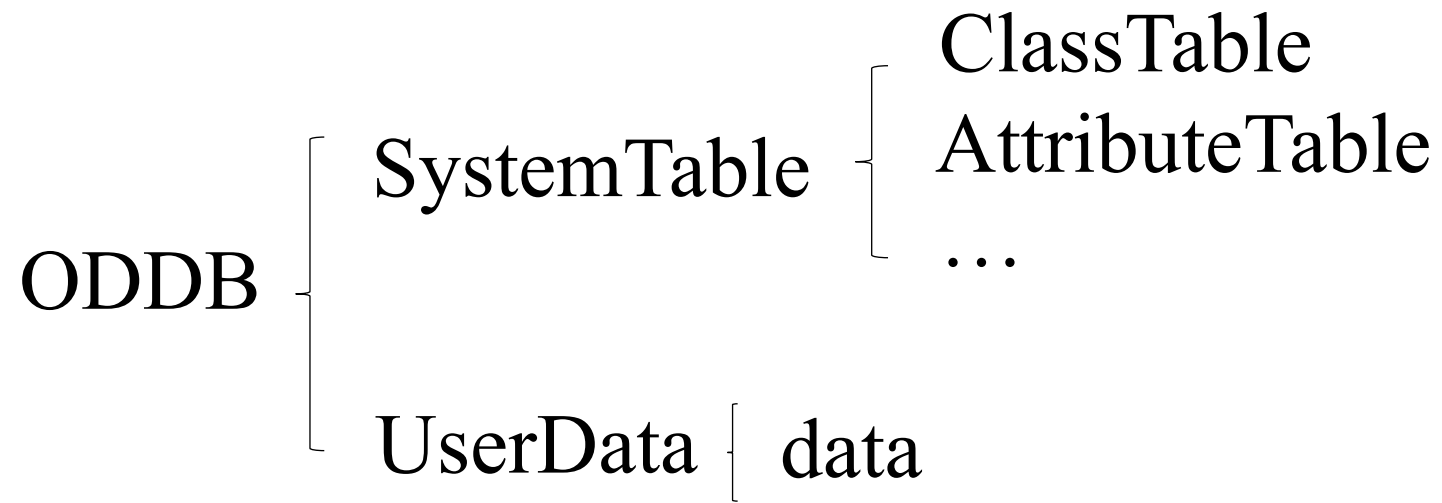
attrid	属性id
rule	Switch规则

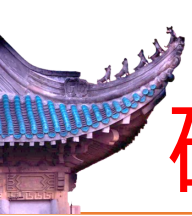
BiPointer表

classid	源类 id
objectid	源对象号
deputyclassid	代理类 id
deputyobjectid	代理对象号

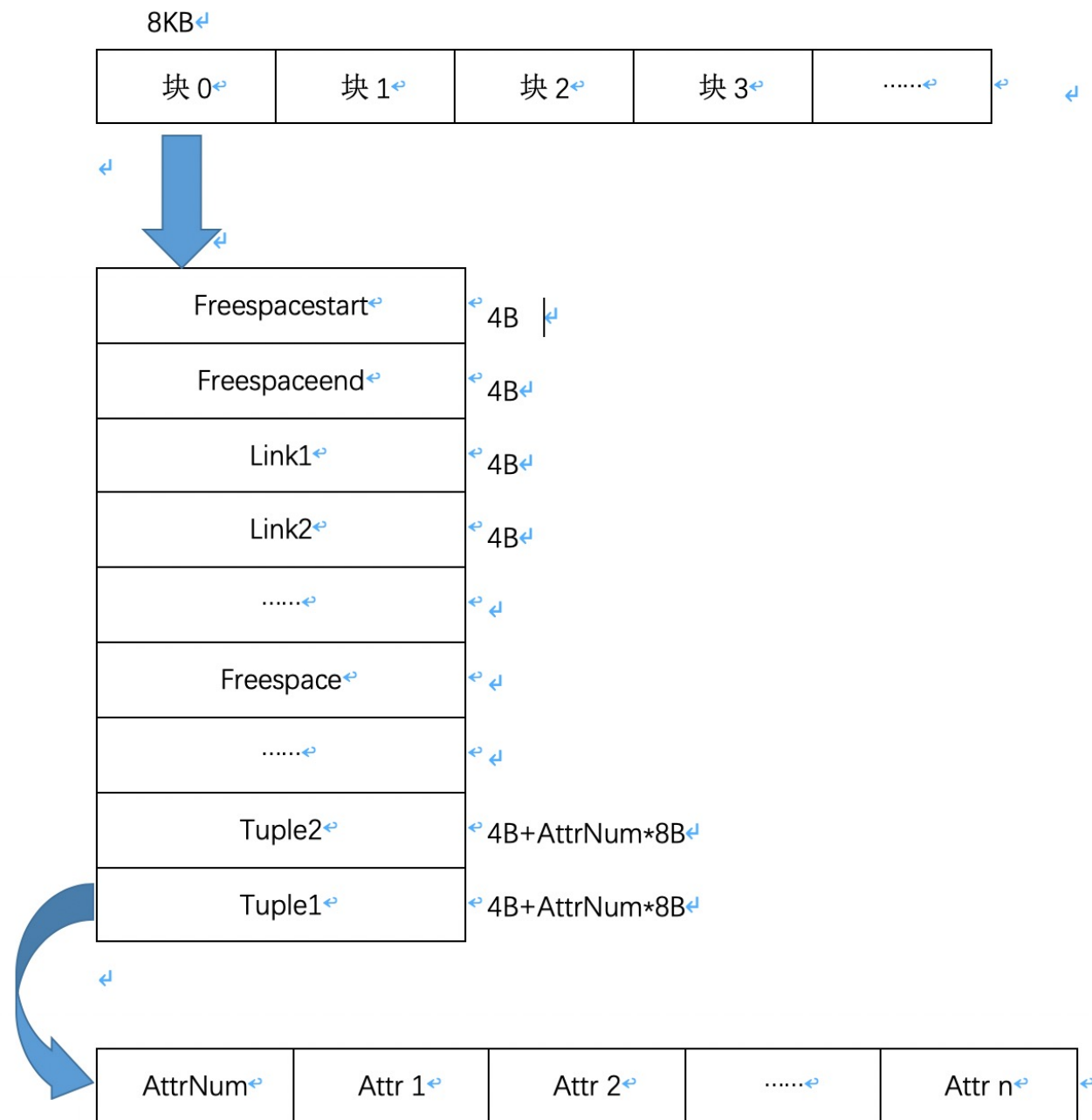


# 文件目录结构





# 磁盘数据







# 缓冲区管理

Java ByteBuffer 分配1000 \* 8KB缓冲区

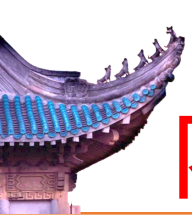
缓冲区0	缓冲区1	缓冲区2	.....	缓冲区999

```
private List<buffPointer> BuffPointerList = new  
ArrayList<>(); //构建缓冲区指针表
```

//缓冲区

```
class buffPointer {  
    int blockNum; //块号  
    Boolean flag; //标记改块是否为脏 (true为脏)  
    int buf_id; //缓冲区索引号  
}
```

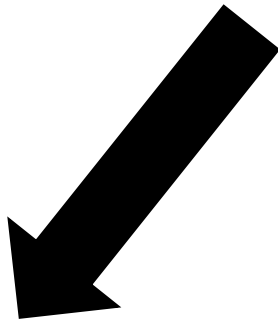
```
private boolean[] buffuse=new boolean[bufflength];//缓  
冲区可用状态表, true为可用
```



# 内存管理系统表

BiPointer表

classid	objectid	deputyclassid	deputyobjectid
1	1	2	1
1	2	2	2
1	3	2	3



```
public class BiPointerTableItem implements Serializable {  
    public int classid = 0;  
    public int objectid = 0;  
    public int deputyid = 0;  
    public int deputyobjectid = 0;  
}
```

```
public BiPointerTableItem(int classid, int objectid, int  
deputyid, int deputyobjectid) {  
    this.classid = classid;  
    this.objectid = objectid;  
    this.deputyid = deputyid;  
    this.deputyobjectid = deputyobjectid;  
}
```

```
public List<BiPointerTableItem>  
biPointerTable=new ArrayList<>();
```



# 内存管理元组

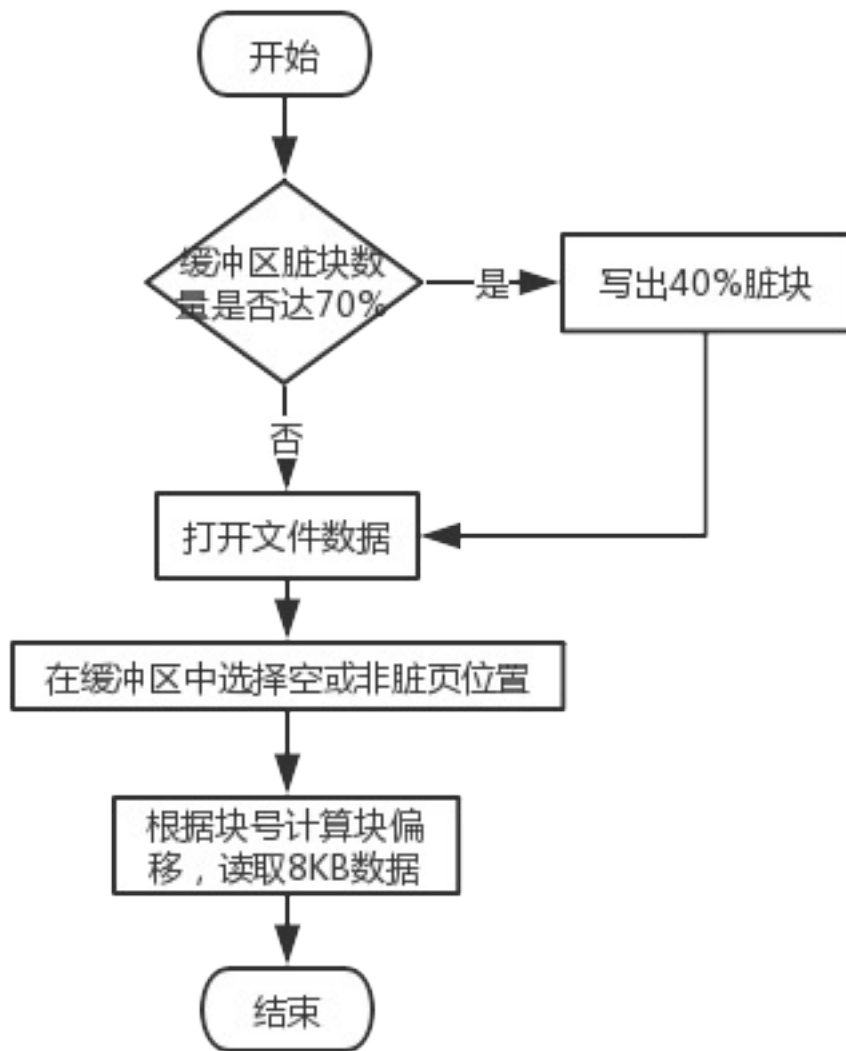
```
public class Tuple implements Serializable {  
    public int tupleHeader;  
    public Object[] tuple;  
  
    public Tuple(Object[] values) {  
        tuple = values.clone();  
        tupleHeader = values.length;  
    }  
  
    public Tuple(){}  
}
```

```
public class TupleList implements Serializable {  
    public List<Tuple> tuplelist = new ArrayList<Tuple>();  
    public int tuplenum = 0;  
  
    public void addTuple(Tuple tuple){  
        this.tuplelist.add(tuple);  
        tuplenum++;  
    }  
}
```



# 缓冲区<->磁盘管理

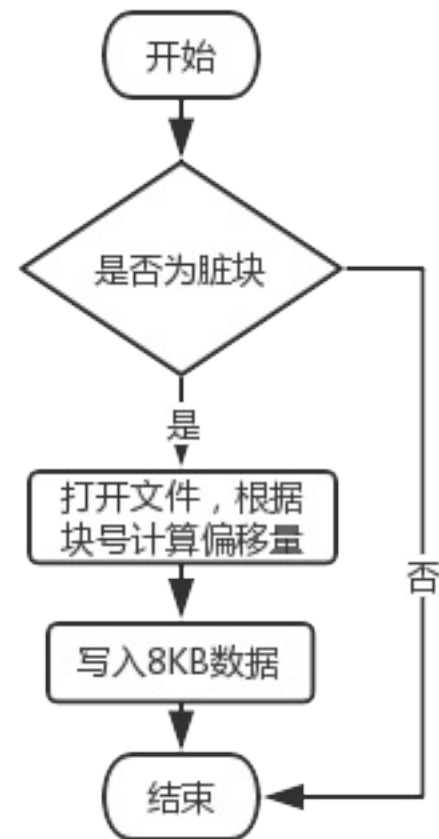
//负责将块内容从磁盘读入缓冲区





# 缓冲区 $\leftrightarrow$ 磁盘管理

//负责将块内容从缓冲区写入磁盘





# 缓冲区替换策略

List< sbufesc> freeListC;                      //记录未修改的缓冲区

缓冲区0	缓冲区1	缓冲区2	.....	缓冲区999

List< sbufesc> freeListD;                      //记录修改过缓冲区

替换策略

LRU                       $\longrightarrow$                       CFLRU<sub>(clean first Least Recently Used)</sub>

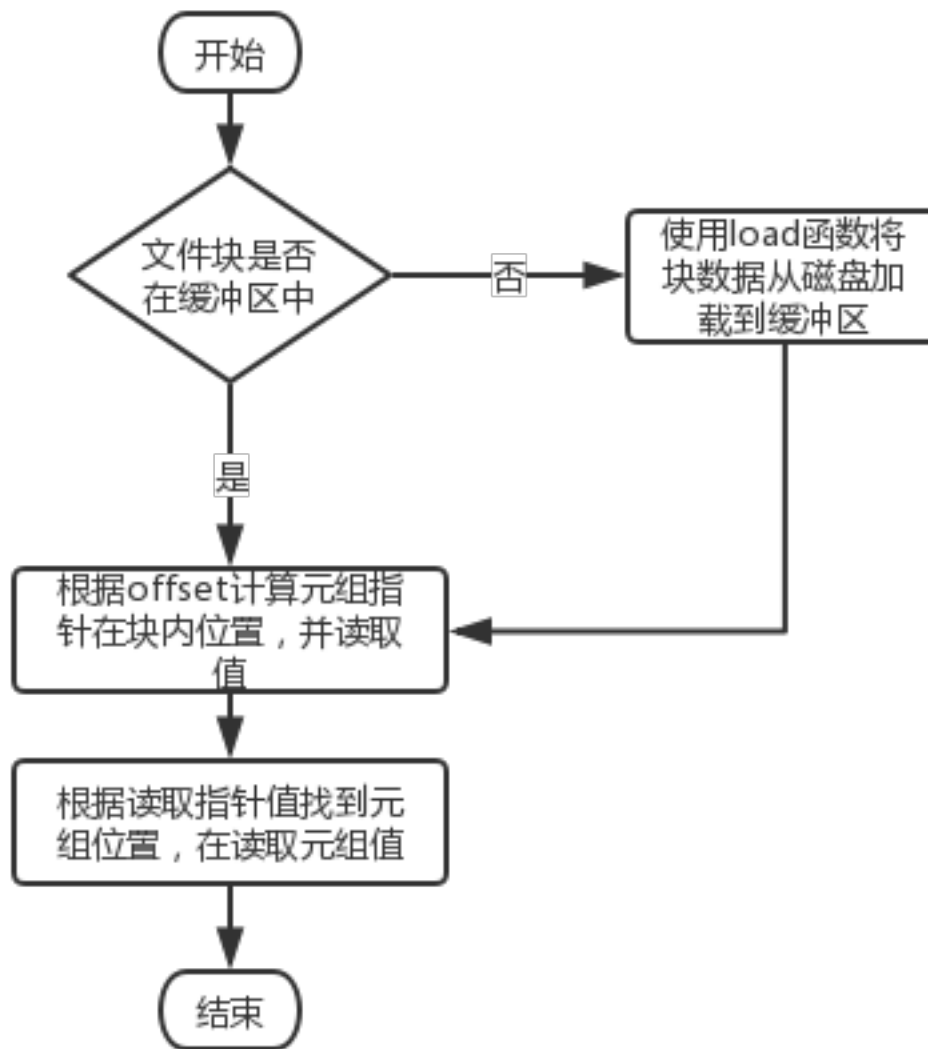
替换策略改变原因：移动端为固态硬盘，特点为写代价远大于读代价。

Seon-Yeong Park, Dawoon Jung, Jeong-Uk Kang, Jinsoo Kim, Joonwon Lee: CFLRU: a replacement algorithm for flash memory. CASES 2006: 234-241



# 缓冲区->内存

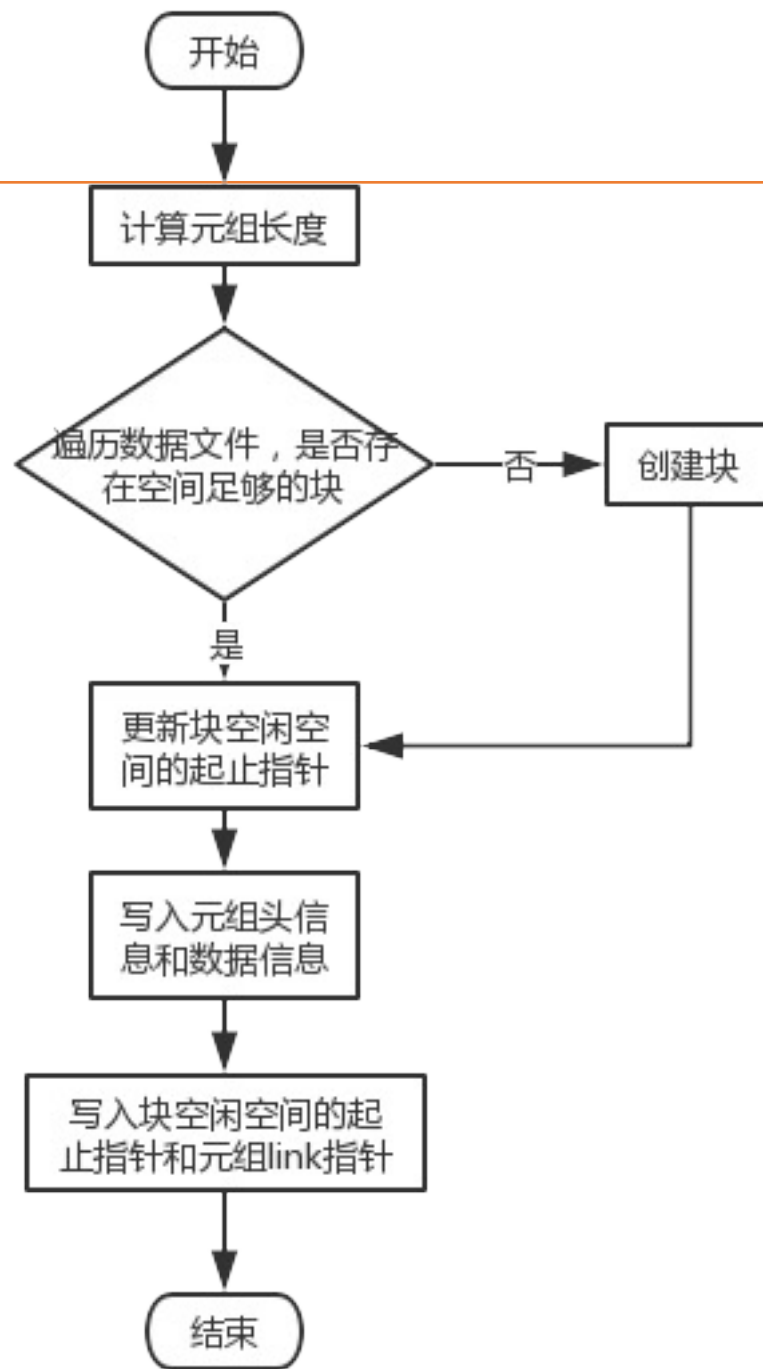
//从缓冲区中块读指定元组。





# 缓冲区<-内存

//将元组写入缓冲区中块，返回元组写入的块号，偏移量。







# 从缓冲区读元组到内存代码示例

```
public Tuple readTuple(int blocknum,int offset){
    Tuple ret=new Tuple();
    buffPointer s=null;
    if((s=findBlock(blocknum))!=null){
        //当块不在缓冲区中时, 从磁盘加载块到缓冲区
        s=load(blocknum);
    }
    //根据偏移获取元组在块内的指针link
    byte[] sta=new byte[4];
    for(int i=0;i<4;i++){
        sta[i]=MemBuff.get(s.buf_id*blocklength+offset+i);
    }
    int start=bytes2Int(sta,0,4);
    //开始读元组, 先读头文件即元组属性个数
    byte[] header=new byte[4];
    for(int i=0;i<4;i++){
        header[i]=MemBuff.get(s.buf_id*blocklength+start+i);
    }
    ret.tupleHeader=bytes2Int(header,0,4);
    ret.tuple=new java.lang.Object[ret.tupleHeader];
    byte[] temp=new byte[ret.tupleHeader*attrstringlen];
    for(int i=0;i<ret.tupleHeader*attrstringlen;i++){
        temp[i]=MemBuff.get(s.buf_id*blocklength+start+4+i);
    }
    for(int i=0;i<ret.tupleHeader;i++){
        String str=byte2str(temp,i*attrstringlen,attrstringlen);
        ret.tuple[i]=str;
    }
    return ret;
}
```



查

询

02

&

编

译





- JavaCC(Java Compiler Compiler)是一个用JAVA开发的受欢迎的语法分析生成器。这个分析生成器工具可以读取上下文无关且有着特殊意义的语法并把它转换成可以识别且匹配该语法的JAVA程序。
- JavaCC的输入文档是一个词法和语法的规范文件，其中也包括一些动作的描述，它的后缀是jj或者jjt。通过JavaCC处理生成7个java类来完成语法分析的工作。由此，用户可输入语句，通过在jj 文件中定义的语法分析类，获取最终的分析结果。



- 使用方式

- 本示例使用javaCC-5.0
- 打开javaCC-5.0/bin
- 在此路径下打开命令行
- 输入 javacc /存储路径/example.jj
- 在 /存储路径/ 下生产对应的7个.java文件，将生产的java文件拷贝到工程下，即可使用该类进行语法词法分析

```
D:\javacc-5.0\bin>javacc Example1.jj
Java Compiler Compiler Version 5.0 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file Example1.jj . . .
Warning: Choice conflict in [...] construct at line 60, column 7.
        Expansion nested within construct and expansion following construct
        have common prefixes, one of which is: "c"
        Consider using a lookahead of 2 or more for nested expansion.
File "TokenMgrError.java" does not exist. Will create one.
File "ParseException.java" does not exist. Will create one.
File "Token.java" does not exist. Will create one.
File "SimpleCharStream.java" does not exist. Will create one.
Parser generated with 0 errors and 1 warnings.
```

- Example.java
- ExampleConstants.java
- ExampleTokenManager.java
- ParseException.java
- SimpleCharStream.java
- Token.java
- TokenMgrError.java
- Example1.jj

执行完成，生  
产7个java类



- 语法文件撰写

一个jj文档包括以下部分：

- Options{}部分：这个部分对产生的语法分析器的特性进行说明，例如向前看的token的个数（用来解除冲突）。这部分可以省略的，因为每一个选项都有默认值，当我们没有对某个选项进行说明时，它就采用默认值。示例如下：

```
options {  
  
    STATIC = false ; //生成非静态类  
  
    LOOKAHEAD = 2; //向前看2个字母,消除歧义用的  
  
    DEBUG_PARSER = true; //以debug形式生成,便于调试  
  
}
```



- 分析器类的声明：这个部分指定了分析器类的名字，以及其他类中成员的声明。
- 这个部分是必需有的，调用分析器的接口定义由该部分实现。即后续语法分析调用SqlParser类实现。
- 示例如右：

```
PARSER_BEGIN(Sqlparser)                                }

public class Sqlparser{                                  try

    private static Sqlparser p = null;                  {p.start();

    public static Executor executor = null;              }

    public static void sqlParser(String                  catch(Exception e)
    sqlInput){                                           {

        InputStream s=new                               System.out.println("oops");
        ByteArrayInputStream(sqlInput.getBytes()
        s())

        if(p!=null){                                     System.out.println(e.getMessage());

                                                    e.printStackTrace();

            p.Relnit(s);

        }

    }else{

    }

    p=new Sqlparser(s);

}PARSER_END(Sqlparser)
```



- 词法分析器：主要定义了SKIP和TOKEN，SKIP指在进行语法分析时可略过的输入，TOKEN指由用户定义的终结符。示例如下：

```
SKIP : { " " }
```

```
SKIP : { "\n" | "\r" | "\r\n" } //被忽略的空格换行等
```

```
TOKEN: { //终结符
```

```
<INTEGER: ["1"-"9"](<DIGIT>)*> //整数
```

```
|<CREATE: "CREATE" | "create"> // "CREATE"
```

```
| ...
```

```
}
```



- 语法分析器：语法中的每一个非终结符都对应一个函数，利用函数调用来表示非终结符之间的结构关系。
- 在进行语法规约时，可在语法中任意插入“{}”，括号体中写入当语法规约到该处时应调用的函数或方法。示例如右：

```
String className():{  
    String clsName;  
    Token clsn;  
    }  
    {  
        clsn = <IDLIST >// 类名为终结符IDLIST，返回类型为Token  
        {//语法规约到此处调用该括号内方法，最终返回String类  
            clsName = clsn.image;  
            return clsName;  
        }  
    }
```





```
//insert into source class only

void insert():{
    String className;
    String valueList;
}
{
    <INSERT><INTO>className=classname()<VALUES><LEFT>valueList=valuelist()<RIGHT>
    {
        executor = new Insert();
        ...//生成语法分析结果
    }
}
```



大写为终结符，小写为非终结符

## 1. 创建源类

CREATE CLASS className

LEFT\_BRACKET attrName attrType [(COMMA attrName attrType)+] RIGHT\_BRACKET ;

• 示例：

CREATE CLASS singer

(singerId int, singerName char(10)···.);



## 2. 创建代理类

```
CREATE SELECTDEPUTY className
```

```
[Attribute LEFT_BRACKET attrName attrType  
[(COMMA attrName attrType)+]  
RIGHT_BRACKET]
```

```
SELECT sAttr AS dAttr [(COMMA sAttr AS  
dAttr)+]
```

```
FROM sClassName
```

```
WHERE whereCluase ;
```

- 示例：

```
CREATE SELECTDEPUTY popSinger
```

```
SELECT singerId AS popSingerId,
```

```
singerName AS popSingerName
```

```
FROM singer
```

```
WHERE singerType = 'pop' ;
```

仅考虑SELECT 代理类，且暂时忽略代理类的实属性。



## 3.删除类

DROP CLASS classname ;

- 示例：

DROP CLASS singer ;



## 4. 向源类插入对象：

INSERT INTO className VALUES

LEFT\_BRACKET valueList RIGHT\_BRACKET ; WHERE whereClause ;

- 示例：

INSERT INTO singer VALUES

(1, 'Liu Dehua', ...);

## 5. 删除一个源类对象

DELETE FROM className

- 示例：

DELETE FROM singer

WHERE singerId = 1 ;

查询条件不涉及关联查询，仅涉及单一类，  
但查询条件可有嵌套，形如：

“WHERE a = 1 AND (b=2 OR c = 3)”



## 6. 普通查询语句：

SELECT attrList FROM className WHERE whereCluase ;

- 示例：

SELECT singerName

FROM singer

WHERE gender = 'famale'

AND singerType = 'jazz' ;

查询条件不涉及关联查询，仅涉及单一类，但查询条件可有嵌套，形如：

“WHERE a = 1 AND (b=2 OR c = 3)”



## 7.跨类查询

```
SELECT className [(CROSS className)+]DOTattrName  
FROM className  
WHERE whereCluase ;
```

- 示例

```
SELECT popSinger -> singer.nation  
FROM popSinger  
WHERE singerName = 'Jay Zhou' ;
```

FROM后接起始类名，WHERE筛选条件以起始类的属性作为筛选属性



# 语法分析结果



结果均以字符串数组存储

## 1. 创建源类

- 示例：

```
CREATE CLASS company ( name char,age int, salary int );
```

- 结果：

1,3,company,name,char,age,int,salary,int

对应位置含义（创建源类， 属性个数， 类名， 属性名， 属性类型， …）





# 语法分析结果

## 2. 创建选择代理类

- 示例：

```
CREATE SELECTDEPUTY nandb SELECT name AS n1,{(age+5*2)*2+salary} AS  
birth,salary AS s1 FROM company WHERE age=20;
```

- 结果：

2,3,nandb,name,n1,{(age+5\*2)\*2+salary},birth,salary,s1,company,age,=,20

对应位置含义（创建选择代理类， 属性个数， 类名， 源属性名或切换表达式， 代理属性名， 源属性名或切换表达式， 代理属性名， 源属性名或切换表达式， 代理属性名， 源类名， 代理规则的属性名， 代理规则的判断符号， 代理规则的条件值）



# 语法分析结果

## 3. 删除类

- 示例：

DROP CLASS company;

- 结果：

3,company

对应位置含义（删除类，类名）



# 语法分析结果

## 4.向源类插入对象：

- 示例：

```
INSERT INTO company VALUES  
( "aa",20,1000 );
```

- 结果：

4,3,company,"aa",20,1000

对应位置的含义（插入对象，属性个数，类名，属性1待插入的值，属性2待插入的值，...）

## 5.删除一个源类对象

- 示例：

```
DELETE FROM company WHERE  
name="aa";
```

- 结果：

5,company,name,=,"aa "

对应位置含义（删除对象，类名，删除条件属性名，删除条件的判断符号，删除条件的比较值）



# 语法分析结果

## 6. 普通查询语句：

- 示例：

```
SELECT n1 AS names,birth AS births,s1 AS salarys FROM nandb WHERE  
n1="gg";
```

- 结果：

```
6,3,n1,names,birth,births,s1,salarys,nandb,n1,=,"gg"
```

对应位置含义（查询对象， 属性个数， 类属性值， 查询结果显示属性名， 类属性值， 查询结果显示属性名， 类属性值， 查询结果显示属性名， 类名， 查询条件的属性名， 查询条件的判断符号， 查询条件的比较值）



# 语法分析结果

## 7.跨类查询

- 示例：

SELECT nands -> company -> nandb.n1 FROM nands WHERE s=3010;

- 结果：

7,3,nands,company,nandb,n1,nands,s,=,3010

对应位置含义（跨类查询，跨类的个数，跨类查询的第一个类名，跨类查询的第二个类名，跨类查询的第三个类名，类名，查询条件的属性名，查询条件的判断符号，查询条件的比较值）



作

业

03

要

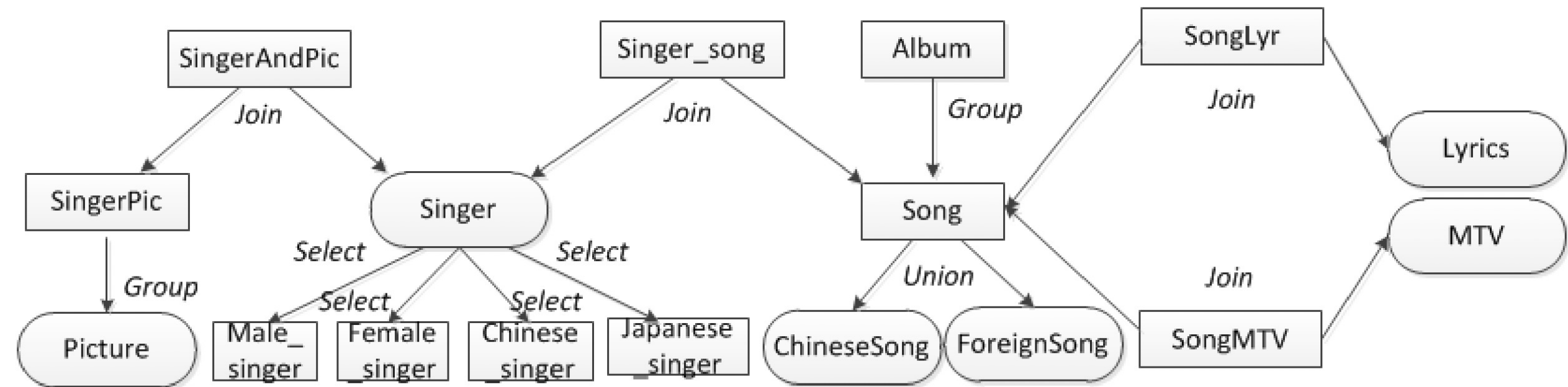
求





# 基本要求

- 增加union代理类， 调整更新迁移算法， 轨迹应用
- <https://github.com/whu-totemdb/tmdb>
- 各种app的轨迹进行Union
- JavaCC sql支持
- 分组可采用第一次大作业模式不变







# 其他说明

- 若发现系统有bug，请提交issue到GitHub
- 第九次课：轨迹的select, union, 调整更新迁移算法，实验指导
- 第十次课：轨迹查询应用及实验指导



# 对象数据库TotemDB参考资料



- <http://totemdb.whu.edu.cn/upload/202102/02/202102022020113648.pdf>
- <http://totemdb.whu.edu.cn/upload/202102/02/202102022020276488.pdf>



# 谢谢大家

武汉大学计算机学院 | 珞珈图腾数据库实验室