MonetDB 快速入门

特性：

列式存储模型：在内存中数据都是以 BAT(Binary Association Table) (OID,value) pairs 形式存放，数据超过了内存和虚拟内存，使用磁盘存放文件，磁盘和内存通过内存映射实现。
基于 CPU 优化的查询架构，自动索引，实时查询优化。

1 架构：

三层软件架构：
SQL front-end：前端 SQL 解析，数据模型优化，降低数据中间结果的总量，最后将 SQL 语句解析为 MAL（MonetDB Assembly Language）。
Tactical-optimizers：一系列优化模块的集合，组成优化管道，这个模块提供功能从符号处理到实时数据分发和执行。
Columnar abstract-machine kernel:列式内核

2 MAL

MonetDB Assembly Language(MAL)
MAL 是 SQL 和 XQuery 前端查询的目标语言。
SQL 语句通过语法解析解析成 MAL，MAL 在经过优化器优化，重写成优化后的 MAL，提供给内核执行。

3 内核

执行内核是一个运行 MAL 语言的虚拟机。
复杂的查询被拆分成多个步骤，每个步骤操作一个列，叫 bulk processing。所有的 BAT 操作被映射成简单的数组操作，一个并行处理引擎。
内核运行过程中会根据输入属性和系统状态选择合适的优化算法和实现来执行 MAL 语言。
运行过程中操作优化:会根据输入属性和系统状态选择合适的优化算法和实现。

第一章、安装

第一节、编译安装

```
rpm -ivh pcre-devel-6.6-6.el5_6.1.x86_64.rpm

./configure --prefix=/opt/pub/MonetDB

make

make install

export PATH=$PATH:/opt/pub/MonetDB/bin
```

第二节、启动

在 Linux 上快速安装，启动 MonetDB 守护进程 Monetdbd。这个守护进程负责管理后台、本地、远程服务器，由 monetdb 控制。在这里，我们展示了一个简单的过程，涉及建立一个数据库，将数据加载和查询。

```
shell> monetdbd create /path/to/mydbfarm

shell> monetdbd start /path/to/mydbfarm

shell> monetdb create voc

shell> monetdb release voc

shell> mclient -u monetdb -d voc

password:<monetdb>
```

第三节、实用程序

mclient： 客户端连接命令

monetdb： 数据库操作命令

mondetdbd： 守护进程操作命令

mserver5： mserver5 是当前 MonetDB 服务器提供所有固定请求的，调整 mserver5 内核使用的参数。

mysqldump:dump 数据库

1、守护进程

一个机器上一个数据库实例，多个数据库实例可以同时在一个机器上，但是每个实例所使用的资源会受影响。

监控所有实例，作为客户端代理提供每个实例的访问，另外处理故障恢复和数据并发引擎。

Monetdb：守护进程，一台机器上只能有一个实例，一个实例有多个进程。

（1）创建守护进程目录

% monetdbd create ~/my-dbfarm

（2）查看守护进程设置

% monetdbd get all ~/my-dbfarm

（3）修改参数

% monetdbd set port=54321 ~/my-dbfarm

（4）启动守护进程

% monetdbd start ~/my-dbfarm

（5）创建一个数据库

% monetdb create my-first-db

（6）查看数据库状态，指定端口号，指定对应的数据库

% monetdb -p54321 status

（7）启动和解锁数据库

启动数据库

% monetdb start my-first-db

解锁数据库，否则数据库是锁定状态

% monetdb release my-first-db

2、连接

% mclient -dmy-first-db

-d 参数指定数据库名，默认用户名和密码是 monetdb/monetdb

连接不上，可以带上端口号

% mclient -p54321 -dmy-first-db

%mclient -u monetdb -d dbtest

3、停止

% monetdbd stop ~/my-dbfarm

4、SQL 导入三种方式

1:
shell> mclient -u voc -d voc voc_dump.sql
password:<voc>

2:
shell> mclient -u voc -d voc < voc_dump.sql
password:<voc>

3:
shell> mclient -u voc -d voc
password:<voc>
sql> \< voc_dump.sql

第二章、SQL

第一节、模式

sql>CREATE USER "tpc" WITH PASSWORD 'tpc' NAME 'TPC Explorer' SCHEMA "sys";

sql>CREATE SCHEMA "tpc" AUTHORIZATION "tpc";

sql>ALTER USER "tpc" SET SCHEMA "tpc";

第二节、数据类型和表

1、原始数据类型

CHAR[ACTER] '(' *length* ')'	character string with length upperbound
VARCHAR '(' *length* ')' | CHARACTER VARYING '(' *length* ')'	string with atmost length upperbound
CLOB | CHARACTER LARGE OBJECT	
BLOB | BINARY LARGE OBJECT	
DECIMAL '(' P ',' S ')' | NUMERIC '(' P ',' S ')'	with precision P and scale S
TINYINT	8 bit integer
SMALLINT	16 bit integer
INT	32 bit integer
BIGINT	64 bit integer
REAL	32 bit floating point
FLOAT	64 bit floating point
DOUBLE [PRECISION]	64 bit floating point
BOOLEAN	
DATE	
TIME ['(' *posint* ')'] [WITH TIME ZONE]	time of day with precision and time zone
TIMESTAMP ['(' *posint* ')'] [WITH TIME ZONE]	date concatenated with unique time, precision and time zone
INTERVAL *interval_qualifier*	a temporal interval

2、建表示例

```
sql>CREATE TABLE test (
more>    id int,
more>    data varchar(30)
more> );

sql>\d test
CREATE TABLE "voc"."test" (
    "id" int,
    "data" varchar(30)
);
```

第三节、函数

基本与 Postgresql 兼容，只是类型转换不一样。

1、类型转换函数

第四节、加载数据

1、普通装载

方式一：直接使用 Insert into，可以通过 START TRANSACTION 和 COMMIT 减少事物提交。这种方式因为每次查询都是独立的，所以每次只能使用到一个 CPU 核。

方式二：COPY INTO

COPY INTO TABLE FROM ‘FILE’;

大量数据插入式，server 不知道需要分配多少内存，因此只会分配很少，也就是在插入过程中，需要不停的分配内存，这个开销会非常大。因此，最好能给定一个值多少条记录会被插入。

COPY n RECORDS INTO table FROM ‘file’

N 必须比实际插入的数字要大，如果文件实际的值大于 N，只会有 N 条记录会被插入。在同一表同时有多个 COPY INTO 查询，给一个更大的值会非常有效。

offset 值指定数据加载开始位置，第一条记录 offset 为 1

完整性约束最好在文件被加载完了之后再添加，因为 ALTER 命令是批检查和处理，性能会更好。

2、导出

COPY INTO 命令把表 dump 成一个 ASCII 文件。

导入导出可以指定 gz 和 bz2 的压缩算法。

3、二进制批加载

COPY 命令，性能主要消耗在将 ASCII 值转化为二进制，MONETDB 针对多核进行了高度优化，多个线程会并行处理。

用户直接根据 BAT 模型，生成二进制文件。

```
create table Tmp( i integer, f real, s string);
copy binary into Tmp from ('path_to_file_i', 'path_to_file_f', 'path_to_file_s');
```

文件名是列明的绝对路径，这个路径需要和 farm 同样的文件系统。他们会直接替换 TMP 的内容。文件被拷贝完了之后，原来的空间就可以被回收利用。

每个文件直接用二进制表示，是一个 C 语言数组的 DUMP。

Char(1byte) tinyint(8-bits) smallint(16bits) int(32bits) bigint(64bit)

Real 和 double 映射 C 语言的 float 和 double 类型。

可变字符串，文件中存放的对应的 C 语言的字符串，每行通过分割符分割，并且没有转义字符。所有文件需要对其，有多个值在文件中，表中就有多少条记录。

其他的类型（包括 UTF-8 和转义字符）必须要用 COYP INTO 加载。

第五节、JDBC

1、驱动

JDBC 下载：http://dev.monetdb.org/downloads/Java/Latest/

2、加载驱动

```
// make sure the ClassLoader has the MonetDB JDBC driver loaded
Class.forName("nl.cwi.monetdb.jdbc.MonetDriver");
// request a Connection to a MonetDB server running on 'localhost'
Connection con = DriverManager.getConnection("jdbc:monetdb://localhost/database", "monetdb", "monetdb");
```

3、示例

```
import java.sql.*;

public class MJDBCTest {
  public static void main(String[] args) throws Exception {
    // make sure the driver is loaded
    Class.forName("nl.cwi.monetdb.jdbc.MonetDriver");
      Connection con = DriverManager.getConnection("jdbc:monetdb://localhost/database", "monetdb", "monetdb");
    Statement st = con.createStatement();
    ResultSet rs;

     rs = st.executeQuery("SELECT a.var1, COUNT(b.id) as total FROM a, b WHERE a.var1 = b.id AND a.var1 = 'andb' GROUP BY a.var1 ORDER BY a.var1, total;");
    // get meta data and print columns with their type
    ResultSetMetaData md = rs.getMetaData();
    for (int i = 1; i <= md.getColumnCount(); i++) {
      System.out.print(md.getColumnName(i) + ":" +
        md.getColumnTypeName(i) + "\t");
    }
    System.out.println("");

    for (int i = 0; rs.next() && i < 5; i++) {
      for (int j = 1; j <= md.getColumnCount(); j++) {
        System.out.print(rs.getString(j) + "\t");
```

```
        }
        System.out.println("");
    }

    // tell the driver to only return 5 rows, it can optimize on this
    // value, and will not fetch any more than 5 rows.
    st.setMaxRows(5);
    // we ask the database for 22 rows, while we set the JDBC driver to
    // 5 rows, this shouldn't be a problem at all...
    rs = st.executeQuery("select * from a limit 22");
    // read till the driver says there are no rows left
    for (int i = 0; rs.next(); i++) {
        System.out.print("[" + rs.getString("var1") + "]");
        System.out.print("[" + rs.getString("var2") + "]");
        System.out.print("[" + rs.getInt("var3") + "]");
        System.out.println("[" + rs.getString("var4") + "]");
    }
    // unset the row limit; 0 means as much as the database sends us
    st.setMaxRows(0);
    // we only ask 10 rows
    rs = st.executeQuery("select * from b limit 10;");
    // and simply print them
    while (rs.next()) {
        System.out.print(rs.getInt("rowid") + ", ");
        System.out.print(rs.getString("id") + ", ");
        System.out.print(rs.getInt("var1") + ", ");
        System.out.print(rs.getInt("var2") + ", ");
        System.out.print(rs.getString("var3") + ", ");
        System.out.println(rs.getString("var4"));
    }
    st.executeUpdate("delete from a where var1 = 'zzzz'");
    con.close();
  }
}
```

第六节、语法树

详细内容参见：

“http://www.monetdb.org/Documentation/Manuals/SQLreference/DataDefinition”

第三章、基础管理

第一节、执行计划

可以用 Explain 语句查看 SQL 编译器产生的中间代码。它给出了详细处理过程的动作描述。下面是展示的是一个例子。输出依赖于优化器的设置。

sql>select count(*) from tables;

```
sql>explain select count(*) from tables;
+-----------------------------------------------------------------------------------------+
| mal                                                                                     |
+=========================================================================================
=====+
| function user.s3_2{autoCommit=true}():void;                                             |
|    _2 := sql.mvc();                                                                     |
| barrier _143 := language.dataflow();                                                    |
|    _23:bat[:oid,:sht]  := sql.bind(_2,"sys","_tables","type",1);                        |
|    _24 := algebra.thetauselect(_23,2:sht,"<");                                          |
|    _25:bat[:oid,:oid]  := sql.bind_dbat(_2,"sys","_tables",1);                          |
|    _27 := bat.reverse(_25);                                                             |
|    _97 := algebra.kdifference(_24,_27);                                                 |
|    _110 := algebra.markT(_97,5,4);                                                      |
|    _117 := bat.reverse(_110);                                                           |
|    _30:bat[:oid,:int]  := sql.bind(_2,"sys","_tables","id",1);                          |
|    _142 := algebra.leftjoin(_117,_30);                                                  |
|    _64:bat[:oid,:sht]  := sql.bind(_2,"sys","_tables","type",0,27@0,nil:oid);           |
|    _72 := algebra.thetauselect(_64,2:sht,"<");                                          |
|    _20:bat[:oid,:sht]  := sql.bind(_2,"sys","_tables","type",2);                        |
|    _76 := algebra.kdifference(_72,_20);                                                 |
|    _22 := algebra.thetauselect(_20,2:sht,"<");                                          |
|    _80 := algebra.semijoin(_22,_64);                                                    |
|    _85 := algebra.kunion(_76,_80);                                                      |
...
|    _13 := algebra.kdifference(_8,_12);                                                  |
|    _14 := algebra.markT(_13,0@0);                                                       |
|    _15 := bat.reverse(_14);                                                             |
|    _16:bat[:oid,:int]  := sql.bind(_2,"tmp","_tables","id",0);                          |
|    _18 := algebra.leftjoin(_15,_16);                                                    |
| exit _143;                                                                              |
|    _33:bat[:oid,:int]  := bat.new(nil:oid,nil:int);                                     |
| barrier _146 := language.dataflow();                                                    |
|    _32 := mat.pack(_134,_136,_138,_140,_142);                                           |
```

```
|    _36 := bat.append(_33,_32,true);                                                    |
|    _38 := bat.append(_36,_18,true);                                                    |
|    _39 := aggr.count(_38);                                                  |
| exit _146;                                                    |
|    sql.exportValue(1,".tables","L6","wrd",64,0,6,_39,"");                          |
| end s3_2;                                                     |
+-------------------------------------------------------------------------------------+
86 tuples
sql>
```

第二节、数据字典

参见：http://www.monetdb.org/Documentation/SQLcatalog/TablesColumns

第三节、监控数据库活动

1、正在执行的任务

select * from sys.queue();

第四节、监控存储使用

磁盘空间可以通过 du 命令查看 dbfarm 目录或者数据库中执行 select * from storage();

第五节、备份还原

1、SQL 转储

SQL 转储是对数据库镜像或者迁移的常用的方法。如在两个版本之间。SQLdump 是 ASCII 编码的 SQL 的集合。执行这些脚本，可以重新创建一样的数据库。MonetDB 不提供全局锁，并发情况下这个转储可能无效。

shell> mclient -lsql --database=voc --dump >/tmp/voc.sql

可以检察/tmp/voc.sql，确认它是可读的。如果空间是问题，可以用管道把输出给一个压缩工具，或者使用 linux

工具直接发送到另外一台机器，移动数据文件到新机器。MonetDB 可以用于创建数据库。

```
shell> mclient -lsql --database=voc /tmp/voc.sql
```

2、快速备份

```
shell> monetdb stop demo
shell> monetdb lock demo
shell> monetdb release demo
```

第四章、技巧

第一节、单步调试执行

SQL 语句被翻译成为 Mal 程序，优化存储为用户模块。代码可以用 MAL_debugger 跟踪，以下是一个调试的例子。

```
>debug select count(*) from tables;
#   mdb.start()
mdb>next
#   user.s1_0()
mdb>next
#   _2:bat[:oid,:int]  := sql.bind(_3="sys", _4="ptables", _5="id", _6=0)
mdb>next
#   _8:bat[:oid,:int]  := sql.bind(_3="sys", _4="ptables", _5="id", _9=1)
mdb> ...
.
```

第二节、集群

http://blog.csdn.net/wzgyahoo/article/details/12972111

第三节、索引

支持标准 SQL 的索引创建，但是不起作用，MonetDB 会自动创建和维护索引。
外键约束会内部自动创建索引。尽量创建外键约束，外联结时有 1 倍以上性能提高。

第四节、事务

显示事物：

START TRANSACTION

COMMIT

ROOLBACK。

也可以将 session 参数设置为 auto_commit 为 true，这样单 SQL 会是一个独立的事物。

一行被删除，只是标记一下，不会降低表的大小，需要通过回收算法回收。

事物使用的是乐观并发控制：Optimistic concurrency control，提交前，每个事物检测没有其他事物修改数据，如果检查到了冲突修改，提交的事物就被回滚。