GBase 数据库

介绍

GBase 8a能够实现大数据的全数据（结构化数据、半结构化数据和非结构化数据）存储管理和高效分析，为行业大数据应用提供完整的数据库解决方案。GBase 8a分析型数据库的主要市场是商业分析和商业智能市场。产品主要应用在政府、党委、安全敏感部门、国防、统计、审计、银监、证监等领域，以及电信、金融、电力等拥有海量业务数据的行业。

GBase 8a MPP Cluster， GBase 8a集群产品主要市场定位为满足在10TB到PB级海量数据级别中，可提供高速查询分析，同时可实现7*24高可用性，2000至10000及以上高并发，在线平滑扩展等市场需求。产品主要应用在政府、党委、安全敏感部门、国防、统计、审计、银监、证监等领域，以及电信、金融、电力等拥有海量业务数据的行业。

GBase 8t是一款与世界技术同级的国产事务型通用数据库系统。原型产品在世界各地金融、电信、政府、企业的核心业务系统中广泛应用，OLTP事务处理性能达到同代Oracle水平景中替代Oracle。

GBase 8m是将数据存储于内存介质并加以管理的新型数据库管理系统。在内存中存储和处理数据，可以获得极高的数据存取速度和极强的并发访问能力。内存数据库系统带来的优越性能，不仅仅在于对内存读写比对磁盘读写快上，更重要的是，从根本上抛弃了磁盘数据管理的许多传统方式，基于全部数据都在内存中管理进行了新的体系结构的设计，从而使数据处理速度对比数据库平均在10倍以上，某些情况甚至可以达到1000倍。

支持的数据类型

GBase 8a MPP Cluster的数据类型	类型名称	最小值	最大值	占用字节
数值型	TINYINT	-127	127	1
	SMALLINT	-32767	32767	2
	INT	-2147483647	2147483647	4
	BIGINT	-9223372036854775806	9223372036854775806	8
	FLOAT	-3.40E+38	3.40E+38	4
	DOUBLE	-1.7976931348623157E+308	1.7976931348623157E+308	8
	DECIMAL	-(1E+M -1)/(1E+D)	(1E+M -1)/(1E+D)	动态计算
	NUMERIC			
字符型	CHAR		255	8
	VARCHAR		10922(utf8)、16383(gbk)	
	TEXT		10922	
二进制类型	BLOB			32767
	LONGBLOB			67108864
日期和时间型	DATE			
	DATETIME			
	TIME			
	TIMESTAMP			

备注：显然数值型的最大值跟最小值跟其所占的字节数息息相关，一个字节占8位，一位表符号位，所以比如TINYINT,最大值为$127(2^7-1)$，

FLOAT 代表一个浮点型数值，占用 4 个字节，它所存储的数值不是一个 准确值。允许的值是-3.402823466E+38到-1.175494351E-38，0， 1.175494351E-38 到 3.402823466E+38。这些是理论限制，基于 IEEE 标准。 实际的范围根据硬件或操作系统的不同可能稍微小些。

GBase 8a 允许在关键字 FLOAT 后面的括号内选择用位指定精度，即 FLOAT(X)。0 到 23 的精度对应 FLOAT 列的 4 字节单精度，24 到 53 的精度对 应 DOUBLE 列的 8 字节双精度。当 24<=X<=53 时，FLOAT(X)与 DOUBLE(X)等价。

同时 GBase 8a 允许使用非标准语法 FLOAT(M,D) (M 是整数位数和小数位 数的总位数，D 是小数的个数) ，GBase 8a 保存值时进行四舍五入。

DECIMAL[(M[, D])]代表一个精确值，它所存储的数值范围是-(1E+M-1)/(1E+D)到(1E+M -1)/(1E+D)。 在DECIMAL[(M[, D])]数据类型中，M 是总位数，支持的最大长度为 65； D 是小数点后面的位数，支持的最大长度为30。 在不需要过高的数字精度的场景中，DECIMAL 中的 M 可以定义为 M≤18， 这样可以获得更好的查询性能。

NUMERIC数据类型与DECIMAL数据类型完全等价

CHAR 类型仅仅是为了兼容 SQL 标准，因此，不建议使用者在实际的项目 应用场景使用此数据类型，建议使用 VARCHAR 数据类型。

VARCHAR变长字符串，m 表示该列中串的长度，其范围是 1 到 10922 个字符。 当存储 VARCHAR 类型的数据时，不会用空格填充补足列定义长度，存储 的数据包含空格时，保留空格。

TEXT 类型仅仅是为了兼容其它数据库的类型，推荐使用 VARCHAR 类型。 TEXT 类型最大支持 10922 字符的存储长度，定义 TEXT 列时，不能为它 指定 DEFAULT 值。

BLOB 列支持 32KB 的存储容量。

创建表时，BLOB 列不可以有 DEFAULT 值

查询语句中，BLOB 列不支持过滤条件。

查询语句中，BLOB 列不支持 OLAP 函数。

日期格式

类型名称	最小值	最大值	格式
DATE	0001-01-01	9999-12-31	YYYY-MM-dd
DATETIME	0001-01-01 00:00:00.000000	9999-12-31 23:59:59.999999	YYYY-MM-dd HH-MI:SS.ffffff
TIME	-838:59:59	838:59:59	HHH:MI:SS
TIMESTAMP	1970_01-01 08:00:01	2038-01-01 00:59:59	YYYY-MM-DD HH:MI:SS

备注：

TIME 的范围是,-838:59:59'到,838:59:59'。TIME 类型不仅可以用

于表示一天的时间，而且可以用来表示所经过的时间或两个事件之间的时间

间隔（这可能比 24 小时大许多或是一个负值）。

对于以字符串指定的包含时间定界符的 TIME 值，小于 10 的时、分或秒，

可以不指定为两位数值。,8:3:2'与,08:03:02'是一致的。

安装流程

1. 获取8a单机安装包

   从 官网 上下载压缩包。

   
   GBase8a-NoLicense-Free-8.6.2_build43-R7-redhat7.3-x86_64.tar.bz2

2. 解压安装包，生成安装目录

```
1    tar xfj GBase8a-NoLicense-Free-8.6.2_build43-R7-redhat7.3-x86_64.tar.bz
```

3. 进入安装目录

```
cd GBaseInstall
ls
==>
clear.sh  config  docs  install_config  Install_lin.sh  install_templet  license.txt  log  readme.txt  server
```

4. 执行安装程序

```
./Install_lin.sh
```

5. 安装成功，设置全局变量

```
soure ~/.bashrc
```

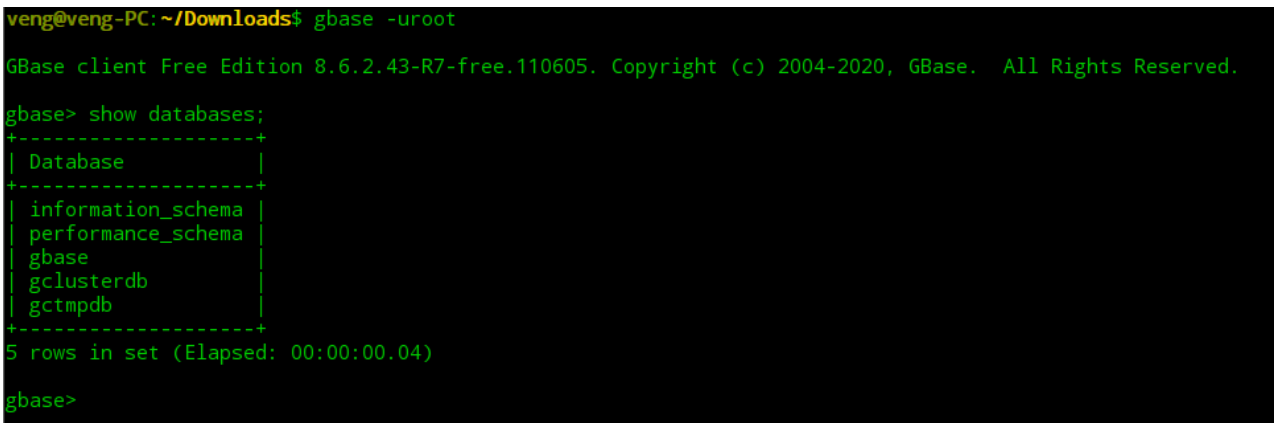
6. 启动数据库

```
gbase.server start
```

7. 进入数据库

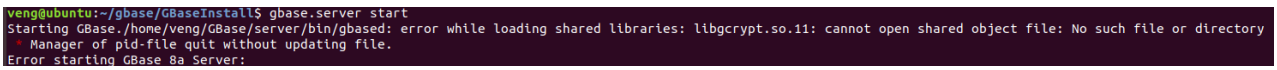
root用户进入，默认密码无。

```
gbase -uroot
```



安装过程中出现的错误

1. 缺少 `libgcrypt.so.11` 库



解决方案：安装相应的库。

先下载安装包

```
sudo wget https://launchpad.net/ubuntu/+archive/primary/+files/libgcrypt11_1.5.3-2ubuntu4_amd64.deb
```

然后安装：

```
sudo dpkg -i libgcrypt11_1.5.3-2ubuntu4_amd64.deb
```

2. 缺少 `libsasl2.so.3` 库

   在网上并未找到该版本的库

   折中方案：

   我在 `/lib/x86_64-linux-gnu` 下找到了 `libsasl2.so.2` 库，然后我把该版本的库复制一份并重新命名为 `libsasl2.so.3` ，解决。

用户的授权管理

帐号管理

新增用户

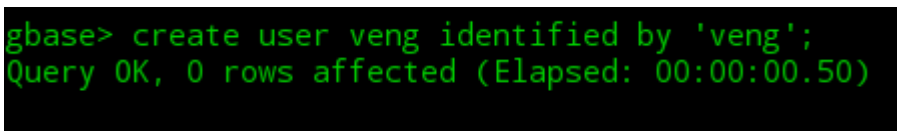
```
CREATE USER user [IDENTIFIED BY [PASSWORD] [password]]
```

user ：帐号名称，支持如下几种方式的书写：

- user@'%'//任何主机
- user@'localhost'//本机
- user@'192.168.0.0' //网段
- user@'192.168.10.6' //ip 地址
- user //与 user@'%'等价

例如：

```
gbase> create user veng identified by 'veng';
```



修改用户账号名称

```
RENAME USER old_user TO new_user
```

```
gbase> rename user veng to veng2
    -> ;
Query OK, 0 rows affected (Elapsed: 00:00:00.22)

gbase> use gbase;
Query OK, 0 rows affected (Elapsed: 00:00:00.16)

gbase> select User from user;
+------------------+
| User             |
+------------------+
| gbase            |
| root             |
| veng2            |
+------------------+
3 rows in set (Elapsed: 00:00:00.01)
```

修改用户名称后，权限保留不变。

修改密码

```
SET PASSWORD [FOR user] = PASSWORD('newpassword')
```

FOR *user* ：指定要修改密码的帐户名称，如果省略，表示修改当前登录

集群帐户的的密码。

```
gbase> set password for veng2 = password('123456');
Query OK, 0 rows affected (Elapsed: 00:00:00.52)

gbase> exit
Bye
veng@veng-PC:/lib/x86_64-linux-gnu$ gbase -uveng2 -p
Enter password:

GBase client Free Edition 8.6.2.43-R7-free.110605. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase>
```

删除用户

```
DROP USER user;
```

DROP USER 语句删除一个 GBase 8a MPP Cluster 帐号。要使用该语句必

须有全局 DROP USER 权限。 DROP USER 不会自动关闭任何打开的用户会话。更确切地说，当存在使 用此用户打开的会话时，删除此用户并不会影响这些已打开会话的访问权限， 直到这些会话关闭.

授权管理

授权

```
GRANT
priv_type [(column_list)][,priv_type [(column_list)]] ...
ON [object_type] priv_level
TO user IDENTIFIED BY [[PASSWORD] [password]]
[WITH with_option ...]
```

`object_type`：

TABLE | FUNCTION | PROCEDURE

`priv_level`：

| . | database_name.* | database_name.table_name | table_name | database_name.routine_name

收回权限

```
REVOKE
priv_type [(column_list)]
[, priv_type [(column_list)]] ...
ON [object_type] priv_level
FROM user
```

权限级别

GRANT 和 REVOKE 语句允许系统管理员处理用户权限的赋予与收回。 对于 GRANT 和 REVOKE 中的权限级别，`priv_type` 指定为下列任一种。

权 限 意 义	---
ALL [PRIVILEGES]	设置除GRANT OPTION之外的所有简单权限
ALTER	允许使用ALTER TABLE
ALTER ROUTINE	更改或取消已存储的子程序
CREATE	允许使用CREATE TABLE
CREATE ROUTINE	创建已存储的子程序
CREATE TEMPORARY TABLES	允许使用CREATE TEMPORARY TABLE
CREATE USER	允许使用CREATE USER, DROP USER, RENAME USER和REVOKE ALL PRIVILEGES
CREATE VIEW	允许使用CREATE VIEW
DELETE	允许使用DELETE
DROP	允许使用DROP TABLE
EXECUTE	允许用户运行已存储的子程序
FILE	允许使用SELECT...FROM TABLE_NAME INTO OUTFILE等
GRANT OPTION	允许授予权限
INDEX	允许使用CREATE INDEX和DROP INDEX
INSERT	允许使用INSERT
PROCESS	允许使用SHOW FULL PROCESSLIST
RELOAD	允许使用FLUSH
SELECT	允许使用SELECT
SHOW DATABASES	SHOW DATABASES显示所有数据库
SHOW VIEW	允许使用SHOW CREATE VIEW
SHUTDOWN	允许使用gbaseadmin shutdown
SUPPER	允许使用KILL和SET GLOBAL语句
UPDATE	允许使用UPDATE
USAGE	仅仅用于连接登录数据库，主要用来设置with option部分EVENT 创建、修改、删除EVENT的权限

说明： 数据库权限分为数据库对象操作权限等以下 5 类： 数据库对象操作类权限；

数据操作类权限； 存储过程、自定义函数执行权限； 数据查看类权限； 数据库权限（包含用户管理）管理权限。

例子：

```
gbase> create database test;
Query OK, 1 row affected (Elapsed: 00:00:01.09)

gbase> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| performance_schema |
| gbase              |
| gclusterdb         |
| gctmpdb            |
| test               |
+--------------------+
6 rows in set (Elapsed: 00:00:00.04)

gbase> use test;
Query OK, 0 rows affected (Elapsed: 00:00:00.12)

gbase> create table user(age int,name varchar(20));
Query OK, 0 rows affected (Elapsed: 00:00:00.84)

gbase> insert into user values(23,'veng'),(11,'joy');
Query OK, 2 rows affected (Elapsed: 00:00:00.63)
Records: 2  Duplicates: 0  Warnings: 0

gbase> select *from user;
+------+------+
| age  | name |
+------+------+
|   23 | veng |
|   11 | joy  |
+------+------+
2 rows in set (Elapsed: 00:00:00.12)

gbase>
```

先用 `root` 用户创建一个测试数据库，以及新表，插入测试值。

然后我们用创建的用户查看，是查看不到的：

```
veng@veng-PC:/lib/x86_64-linux-gnu$ gbase -uveng2 -p
Enter password:

GBase client Free Edition 8.6.2.43-R7-free.110605. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| performance_schema |
+--------------------+
2 rows in set (Elapsed: 00:00:00.16)

gbase>
```

数据库都看不到。

赋予查的权限给该用户：

```
gbase> grant select(name) on test.user to veng2;
Query OK, 0 rows affected (Elapsed: 00:00:00.18)

gbase>
```

上面是给了查询user表name的权利给了该用户。

然后切到该用户再次查看：

```
gbase> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| performance_schema |
| test               |
+--------------------+
3 rows in set (Elapsed: 00:00:00.06)

gbase> select * from test;
ERROR 1046 (3D000): No database selected
gbase> select * from test.user;
ERROR 1143 (42000): SELECT command denied to user 'veng2'@'localhost' for column 'age' in table 'user'
gbase> select age  from test.user;
ERROR 1143 (42000): SELECT command denied to user 'veng2'@'localhost' for column 'age' in table 'user'
gbase> select name  from test.user;
+------+
| name |
+------+
| veng |
| joy  |
+------+
2 rows in set (Elapsed: 00:00:00.14)

gbase>
```

发现只能查看user表中name列，其他列不能查看。

用户组的权限管理

创建用户组

```
1   CREATE ROLE [IF NOT EXISTS] role [, role ] ......
```

```
gbase> create role selectAge;
Query OK, 0 rows affected (Elapsed: 00:00:00.13)
```

给用户组授于权限

```
1   GRANT SELECT ON TEST.T TO role_name
```

```
gbase> grant select(age) on test.user to selectAge;
Query OK, 0 rows affected (Elapsed: 00:00:00.16)
```

给用户授于用户组权限

```
1   GRANT role [, role] … TO user [, user] [WITH ADMIN OPTION]
```

注：使用 `WITH ADMIN OPTION` 标识用户是否拥有把用户组授于其它用户的权限。

```
gbase> grant selectAge to veng2;
Query OK, 0 rows affected (Elapsed: 00:00:00.09)
```

然后用该用户登录，再次查看表：

```
gbase> select * from test.user;
ERROR 1143 (42000): SELECT command denied to user 'veng2'@'localhost' for column 'age' in table 'user'
gbase> select age  from test.user;
+------+
| age  |
+------+
|   23 |
|   11 |
+------+
2 rows in set (Elapsed: 00:00:00.07)

gbase> select * from test.user;
ERROR 1143 (42000): SELECT command denied to user 'veng2'@'localhost' for column 'age' in table 'user'
gbase> select age,name  from test.user;
+------+------+
| age  | name |
+------+------+
|   23 | veng |
|   11 | joy  |
+------+------+
2 rows in set (Elapsed: 00:00:00.04)

gbase> select * from test.user;
ERROR 1143 (42000): SELECT command denied to user 'veng2'@'localhost' for column 'age' in table 'user'
```

会发现一个问题，当 `select *` 会报错，但是把 `select age,name` ，却不会报错。实际上该table只有这两个属性。

回收用户的用户组权限

```
1   REVOKE role [, role] … FROM user [, user]……
```

```
gbase> revoke selectAge from veng2;
Query OK, 0 rows affected (Elapsed: 00:00:00.06)

gbase>
```

改用户再次查看：

```
gbase> select age from test.user;
ERROR 1143 (42000): SELECT command denied to user 'veng2'@'localhost' for column 'age' in table 'user'
```

该用户已被收回该权限。

删除用户组

```
1   DROP ROLE [IF EXISTS] role [, role ] ...
```

```
gbase> drop role selectAge ;
Query OK, 0 rows affected (Elapsed: 00:00:00.40)
```

日志管理

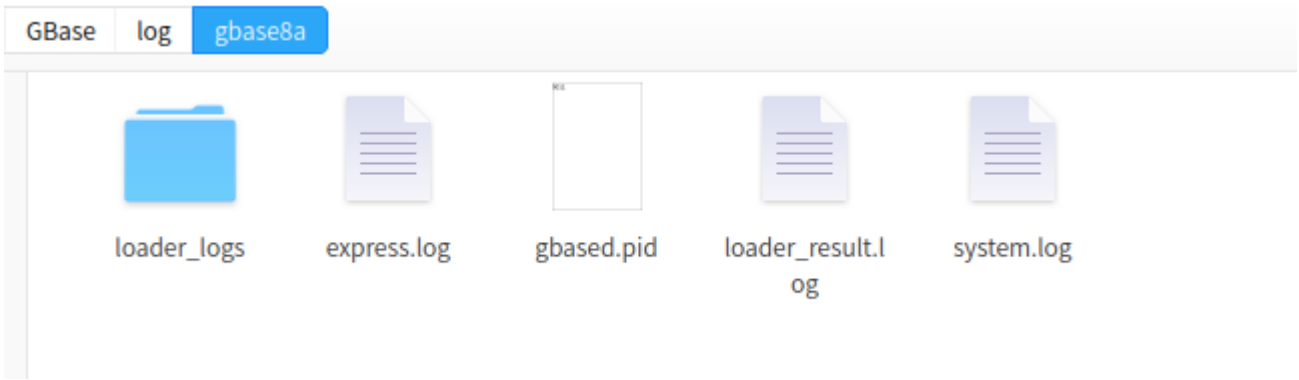
`GBase` 日志分为三类：

- trace日志
- express日志

- system日志

日志位于安装目录下： `/log/gbase8a`

如默认安装路径为：



trace 日志记录查询计划；

express.log 记录 SQL 执行过程以及执行过程中的警告和错误

system.log 主要记录数据库服务启停以及 crash 信息.

当需要分析 SQL 的查询计划时请查看 trace 日志，trace 日志详细记录了SQL 的查询和执行计划，通过分析这些计划来确定 SQL 的计划是否是最优的。

当需要分析 SQL 执行过程中的时间消耗以及执行过程中出现的警告和错误时请查看 express 日志，express 日志详细记录了 SQL 执行过程中各个步骤的时间以及警告和错误，但缺省只记录 SQL 执行过程中的警告和错误，此时可以通过分析这些警告和错误来定位问题的原因。

当 GBase 8a 正常启停和启动失败以及运行过程中出现 crash 时，信息都会记录到 system.log 中，通过分析这些日志来定位启动失败以及运行过程中出现crash 的原因。当运行过程中出现 crash 时，system.log 中记录了宕机的堆栈信息，core 文件中记录了宕机的详细的堆栈信息，如果用户希望看到详细的堆栈信息，则需要在 $HOME/GBase/config 路径下的 gbase_8a_gbase8a.cnf 配置文件中，将“core-file”参数前的注释符号“#”去掉。可以通过 gdp 方式查看详细的 crash 信息:

```
gdp /GBase/server/bin/gbased
thread apply all bt
```

在 GBase 配置文件中设置如下配置项后，查询计划可输出到.trc 文件中：

```
gbase_sql_trace =1
gbase_sql_trace_level = 3
```

```
gbase_sql_trace_level=3
gbase_sql_trace=1
#gbase_cache_type=0
```

gbase8a_root_1_20200304100823.trc

```
/home/veng/GBase/log/gbase8a/gbase8a_root_1_20200304100823.trc
Server Version: 8.6.2.43-R7-free.110605
Version Comment: 110605
Instance Name: gbase8a
Session ID: 1
User: root
Time: 20200304100823
GBASE_HOME=/home/veng/GBase/server/
CPUS: 1
MEM:  710 MB

2020-03-04 10:08:23.841 [M:    0B,    0B,D:    0B] [DC:      0,      0] SQL Statement:
select * from test.user
2020-03-04 10:08:23.842 [M:    0B,    0B,D:    0B] [DC:      0,      0] Start Query Execution
2020-03-04 10:08:23.843 [M:    0B,    0B,D:    0B] [DC:      0,      0]
2020-03-04 10:08:23.843 [M:    0B,    0B,D:    0B] [DC:      0,      0] BEGIN Materialization(2 rows, page size: 65536)
2020-03-04 10:08:23.843 [M:    0B,    0B,D:    0B] [DC:      0,      0] need not materialize here, materialize later
2020-03-04 10:08:23.858 [M: 128B,    0B,D:    0B] [DC:      0,      2] Send 2 rows already
2020-03-04 10:08:23.859 [M: 128B,    0B,D:    0B] [DC:      0,      2] ResultSender: send 0 rows.
2020-03-04 10:08:23.859 [M: 128B,    0B,D:    0B] [DC:      0,      2] output result done.

2020-03-04 10:08:23.859 [M: 128B,    0B,D:    0B] [DC:      0,      2] SUMMARY
2020-03-04 10:08:23.859 [M: 128B,    0B,D:    0B] [DC:      0,      2] elapsed time: 00:00:00.018
2020-03-04 10:08:23.859 [M: 128B,    0B,D:    0B] [DC:      0,      2] data loaded from storage: 119B,    0.012s, 2 DC.
2020-03-04 10:08:23.859 [M: 128B,    0B,D:    0B] [DC:      0,      2] data decompressed:
```

SQL与 mysql 的对比

创建数据库

Gbase：

```
CREATE DATABASE [IF NOT EXISTS] database_name
```

CREATE DATABASE 是以给定的名称创建一个数据库。用户需要获得创建 数据库的权限，才可以使用 CREATE DATABASE。

```
gbase> create database new_database_test;
Query OK, 1 row affected (Elapsed: 00:00:00.11)

gbase> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| performance_schema |
| gbase              |
| gclusterdb         |
| gctmpdb            |
| new_database_test  |
| test               |
+--------------------+
7 rows in set (Elapsed: 00:00:00.00)
```

mysql：

```
CREATE DATABASE 数据库名;
```

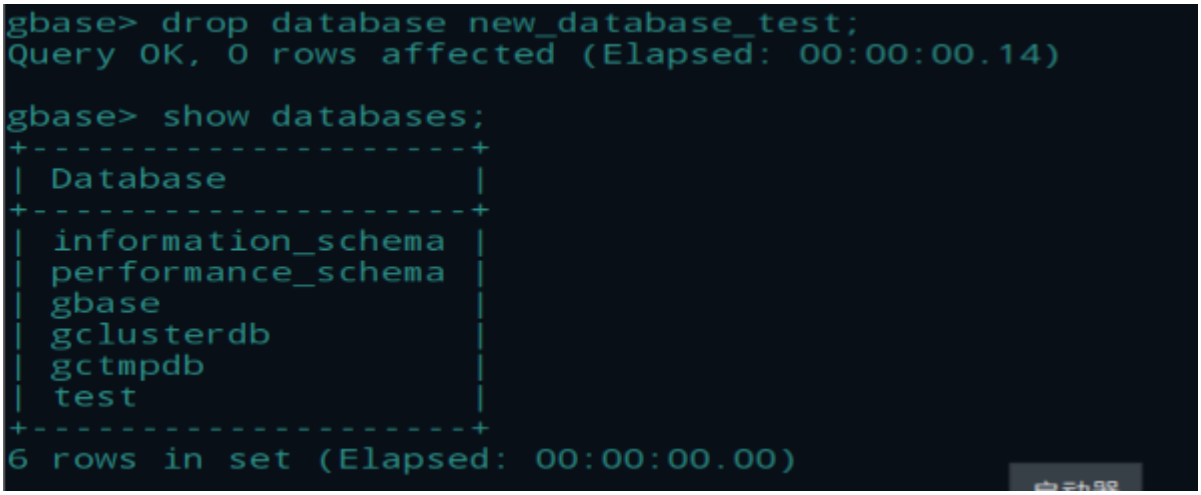
可以看出与 Gbase 数据库是一致的。

删除数据库

GBase：

```
DROP DATABASE [IF EXISTS] database_name
```

`DROP DATABASE` 删除指定的数据库以及它所包含的表。请小心使用此语 句！用户需要获得对数据库的 `DROP` 权限，才可以使用 `DROP DATABASE` 。 使用关键字 `IF EXISTS` ，以防止由于数据库不存在而报告错误。



`mysql` :

```
drop database <数据库名>;
```

与其一致。

创建表

`Gbase` :

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
[ database_name .] table_name
(column_definition [,column_definition], ... [, key_options])
[table_options]
[NOCOPIES];
```

```
column_definition:
column_name data_type [NOT NULL | NULL] [DEFAULT default_value ]
[COMMENT comment_value ]
```

```
key_options:
KEY key_name (' column_name ') [key_dc_size = dc_value USING HASH
GLOBAL]
```

```
table_options:
[REPLICATED | DISTRIBUTED BY (' column_name ') ]
[COMMENT 'comment_value']
```

`TEMPORARY` ：该参数为可选参数，创建临时表需要使用此关键字。

`comment_value` ：指定数据列的备注说明。例如：stu_no id COMMENT ' 学号'。

`key_options` :指定表中的 hash 列。同时可以使用 key_dc_size 参数指定 创建分段 Hash INDEX。

`dc_value` 值如下表所示：

最小值	最大值	备注
0	2147483646	默认为0（即在整列上创建HASH

`table_options` ：默认为随机分布表。

`REPLICATED` ：指定是否是复制表。

在创建一个表时，用户可以使用关键词 `REPLICATED` 来指定是否创建复制 表。如果指定了关键词 `REPLICATED` ，那么创建的复制表在 `GBase 8a MPP Cluster` 的各个节点上存放的是完整数据。

注意：复制表表名尾部不允许是_n{number}编号，例如，mytable_n1，mytable_n12 是不允许使用的。

`DISTRIBUTED BY column_name` ：指定创建表中的物理列 column_name 是 哈希列，这样创建的表，称为哈希分布表。哈希列的值不能为空，且必须是 INT 或者 VARCHAR 类型（执行 OGG Kafka 数据同步时，哈系列的值同样不能 为空）。

`key_dc_size = dc_value USING HASH GLOBAL` ：配合 DISTRIBUTED BY 使用，指定分段 HASH 列，同时创建多个分段 HASH 列。

`COMMENT` ：指定表的备注说明。可以用 SHOW CREATE TABLE table_name 和 SHOW FULL COLUMNS FROM table_name 语句来显示备注信息。

示例：创建一张随机分布表。

```
1   CREATE TABLE t1(a int, b int);
```

```
gbase> create table t1(a int ,b int);
Query OK, 0 rows affected (Elapsed: 00:00:00.12)

gbase> show tables;
+-----------------+
| Tables_in_test |
+-----------------+
| t1              |
| user            |
+-----------------+
2 rows in set (Elapsed: 00:00:00.01)

gbase> show columns from t1;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| a     | int(11) | YES  |     | NULL    |       |
| b     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (Elapsed: 00:00:00.14)
```

示例 2：创建一张复制表。

```
1   CREATE TABLE t2(a int, b int) REPLICATED;
```

```
gbase> create table t2(a int ,b int) replicated;
Query OK, 0 rows affected (Elapsed: 00:00:00.04)

gbase> show columns from t2;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| a     | int(11) | YES  |     | NULL    |       |
| b     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (Elapsed: 00:00:00.01)
```

示例 3：创建带有列注释信息的表。

```
CREATE TABLE t1 (f_username varchar(10) comment 'name' );
```

```
gbase> create table t3(name varchar(10) comment '姓名');
Query OK, 0 rows affected (Elapsed: 00:00:00.34)

gbase> show create table t3;
+-------+-----------------------------------------------------------------------------------------
| Table | Create Table
+-------+-----------------------------------------------------------------------------------------
| t3    | CREATE TABLE "t3" (
  "name" varchar(10) DEFAULT NULL COMMENT '姓名'
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace'    |
+-------+-----------------------------------------------------------------------------------------
1 row in set (Elapsed: 00:00:00.01)
```

`mysql`：

该数据库并非分布式，故创建表的 `sql`：

```
CREATE TABLE <表名> ([表定义选项])[表选项][分区选项];
```

CREATE TABLE 命令语法比较多，其主要是由表创建定义（create-definition）、表选项（table-options）和分区选项（partition-options）所组成的。

与 `GBase` 相比，可以看出创建普通的表，创建带注释的表也是相同，也是大同小异，基本类似,还有就是 `Gbase` 没有主键，外键等概念。

表的投影

```
CREATE TABLE table_name _[( column_definition , ... )] [REPLICATED]
[DISTRIBUTED BY] [AS] SELECT ...
```

根据列定义以及投影列创建表结构，并且将 SELECT 中查询的数据复制到 所创建的表中。

参数说明如下： `REPLICATED` ：指定创建复制表选项。 `DISTRIBUTED BY` ：指定创建表中的物理列 column_name 是哈希列。 `AS` ：指定 SELECT 语句，可选关键字。

示例 1：复制全表表结构及数据来创建随机分布表

```
CREATE TABLE t7(a INT, b DECIMAL, c FLOAT, d DATETIME);
INSERT INTO t7 VALUES (1,2,3.345,'2011-11-11 11:11:11'),(3,5,5.678,'2011-11-11 22:22:22');
CREATE TABLE t8 SELECT * FROM t7;
SELECT * FROM t8;
```

```
gbase> create table t7(a int,b decimal,c float,d datetime);
Query OK, 0 rows affected (Elapsed: 00:00:00.12)

gbase> insert into t7 values(1,2,3.345,'2011-11-11 11:11:11'),(3,5,5.678,'2011-11-11 22:22:22');
Query OK, 2 rows affected (Elapsed: 00:00:00.19)
Records: 2  Duplicates: 0  Warnings: 0

gbase> create table t8 select * from t7;
Query OK, 2 rows affected (Elapsed: 00:00:00.25)
Records: 2  Duplicates: 0  Warnings: 0

gbase> select * from t8;
+------+------+-------+---------------------+
| a    | b    | c     | d                   |
+------+------+-------+---------------------+
|    1 |    2 | 3.345 | 2011-11-11 11:11:11 |
|    3 |    5 | 5.678 | 2011-11-11 22:22:22 |
+------+------+-------+---------------------+
2 rows in set (Elapsed: 00:00:00.01)
```

复制表结构

```
CREATE TABLE table_name1 LIKE table_name2;
```

复制 table_name2 的表结构来创建表 table_name1。

注：不支持 源为 dblink 表，例如： create table test1 like test1@oracle_link1; -- 不支持

示例 1：随机分布表

```
CREATE TABLE t5(a int,b datetime);
INSERT INTO t5 VALUES(1,NOW());
CREATE TABLE t6 LIKE t5;
DESC t6;
SELECT * FROM t6;
```

```
gbase> CREATE TABLE t5(a int,b datetime);
Query OK, 0 rows affected (Elapsed: 00:00:00.07)

gbase> INSERT INTO t5 VALUES(1,NOW());
Query OK, 1 row affected (Elapsed: 00:00:00.14)

gbase> CREATE TABLE t6 LIKE t5;
Query OK, 0 rows affected (Elapsed: 00:00:00.05)

gbase> DESC t6;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| a     | int(11)  | YES  |     | NULL    |       |
| b     | datetime | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
2 rows in set (Elapsed: 00:00:00.04)

gbase> SELECT * FROM t6;
Empty set (Elapsed: 00:00:00.02)
```

创建临时表

```
CREATE TEMPORARY TABLE ...
```

在创建一个表时，用户可以使用关键词 TEMPORARY。临时表被限制在当前连接中，当连接关闭时，临时表会自动地删除。这就意味着，两个不同的连接可以使用同一个临时表名而不会发生冲突，也不会与同名现有的表冲突 （现有表将被隐藏，直到临时表被删除）。使用此种方法，一旦客户端与 `GBase 8a MPP Cluster` 断开连接，临时表将自动删除。

注意事项： 临时表支持除 ALTER 之外的所有 DDL 及 DML 操作。 临时表不能被备份。 临时表不支持使用 `gcdump` 工具导出表结构。 临时表支持在当前连接中使用查询结果导出语句导出表中数据。 在进行集群数据重分布、备份恢复前需要清除当前连接中的临时表。

示例 1：创建临时表。

```
CREATE TEMPORARY TABLE tem_table (a int);
INSERT INTO tem_table VALUES(1),(2),(7),(9);
SELECT * FROM tem_table;
EXIT
gbase -uroot;
SELECT * FROM tem_table;
```

```
gbase> CREATE TEMPORARY TABLE tem_table (a int);
Query OK, 0 rows affected (Elapsed: 00:00:00.05)

gbase> INSERT INTO tem_table VALUES(1),(2),(7),(9);
Query OK, 4 rows affected (Elapsed: 00:00:00.06)
Records: 4  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM tem_table;
+------+
| a    |
+------+
|    1 |
|    2 |
|    7 |
|    9 |
+------+
4 rows in set (Elapsed: 00:00:00.01)

gbase> exit
Bye
veng@veng-PC:~$ gbase -uroot

GBase client Free Edition 8.6.2.43-R7-free.110605. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase> select * from test.tem_table;
ERROR 1146 (42S02): Table 'test.tem_table' doesn't exist
gbase>
```

创建不带副本的表

```
CREATE TABLE [IF NOT EXISTS] table_name( col_name type, ... ) NOCOPIES;
```

`NOCOPIES` ：在创建表时，使用关键词 `NOCOPIES` 可以创建一张不带副本的表。

`nocopies` 表和分布表（随机分布表和哈希分布表）一样，数据分布存储， 支持表的 DDL 语法和 DML 语法；与分布表的区别是，不产生副本，所以无法 提供数据的高可用性。 `nocopies` 表支持使用 `gcdump` 工具导出表结构。

`nocopies` 表支持使用 `SELECT INTO OUTFILE` 和 `rmt: SELECT INTO OUTFILE` 语句导出数据

示例 1：创建 `ocopies` 表。

```
CREATE TABLE t_1 (a int) NOCOPIES;
SHOW CREATE TABLE t_1;
```

```
gbase> create table t_1(a int) nocopies;
Query OK, 0 rows affected (Elapsed: 00:00:00.04)

gbase> show create table t_1;
+-------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Table | Create Table                                                                                                                                                          |
+-------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| t_1   | CREATE TABLE "t_1" (
  "a" int(11) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace' |
+-------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (Elapsed: 00:00:00.01)
```

因为是单机版，故而没有 `nocopies` .

编辑列

```
ALTER TABLE [database_name.]table_name
 alter_specification [, alter_specification] ...
```

```
alter_specification:
 ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
 | ADD [COLUMN] (column_definition,... )
 | ADD INDEX [index_name] [index_type] (index_col_name,... )
 | CHANGE [COLUMN] old_col_name new_col_name column_definition
| CHANGE [COLUMN] old_col_name new_col_name VARCHAR(length)
 | MODIFY [COLUMN] col_name column_definition
 FIRST | AFTER col_name
| MODIFY [COLUMN] col_name VARCHAR(length)
FIRST | AFTER col_name
 | RENAME [TO] new_table_name
 | DROP [COLUMN] col_name
 | DROP NOCOPIES
| SHRINK SPACE
```

参数说明如下：

| ADD [COLUMN] (column_definition,...)：用于增加新的数据列，如果使用 FIRST，则新增加的列位于所有数据列的前面；如果使用 AFTER，则新增加的列，位于指定数据列的后面。默认不使用 FIRST、AFTER，则将增加的新列追加到末尾处。

CHANGE [COLUMN] *old_col_name new_col_name* column_definition：修改列名称。不支持修改列定义。CHANGE [COLUMN] *old_col_name new_col_name* VARCHAR(*length*)：当字段为 VARCHAR 类型时，使用 CHANGE 除修改列名称外还可以增加字段的长度。

MODIFY [COLUMN] col_name column_definition COMMENT comment_value FIRST | AFTER *col_name* ：修改表中存在列的位置和列注释。除 varchar 类型以外，其他类型的列定义不支持修改。

MODIFY *col_name* VARCHAR(*length*) FIRST | AFTER *col_name* ：当字段为 varchar 类型时，使用 MODIFY 除修改列位置外还可以增加字段的长度。

RENAME [TO] new_table_name：修改表名称为 new_table_name。

DROP [COLUMN] *col_name* ：删除表中存在的列。

SHRINK SPACE：释放被删除或未提交的数据文件所占的磁盘空间。

补充说明：

修改 nocopies 表的语法和普通表一样，不必使用关键字 NOCOPIES。

修改列顺序时如果不填写原有的列注释信息，相当于将列注释修改为空。

示例 1：增加列，不写 AFTER 和 FIRST，默认追加到末尾。

```
gbase> desc tt;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| a     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
1 row in set (Elapsed: 00:00:00.00)

gbase> alter table tt add b varchar(20);
Query OK, 0 rows affected (Elapsed: 00:00:00.08)
Records: 0  Duplicates: 0  Warnings: 0
```

```
gbase> desc tt;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| b     | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (Elapsed: 00:00:00.00)
```

示例 2：增加列，并指定增加列的位置.

```
gbase> alter table tt add c int after a;
Query OK, 0 rows affected (Elapsed: 00:00:00.08)
Records: 0  Duplicates: 0  Warnings: 0

gbase> desc tt;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| c     | int(11)     | YES  |     | NULL    |       |
| b     | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (Elapsed: 00:00:00.00)
```

示例 3：一次性增加多列。

```
gbase> alter table tt add (d int,e int);
Query OK, 0 rows affected (Elapsed: 00:00:00.11)
Records: 0  Duplicates: 0  Warnings: 0

gbase> desc tt;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| c     | int(11)     | YES  |     | NULL    |       |
| b     | varchar(20) | YES  |     | NULL    |       |
| d     | int(11)     | YES  |     | NULL    |       |
| e     | int(11)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
5 rows in set (Elapsed: 00:00:00.01)
```

示例 4：删除多列。

```
gbase> alter table tt drop d,drop e;
Query OK, 0 rows affected (Elapsed: 00:00:00.08)
Records: 0  Duplicates: 0  Warnings: 0

gbase> desc tt;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| c     | int(11)     | YES  |     | NULL    |       |
| b     | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (Elapsed: 00:00:00.00)

gbase>
```

示例 5：变更表名称。

```
gbase> alter table tt rename tt_rename;
Query OK, 0 rows affected (Elapsed: 00:00:00.01)

gbase> show tables;
+----------------+
| Tables_in_test |
+----------------+
| course         |
| stu_course     |
| student        |
| student_course |
| t0             |
| t00            |
| t000           |
| t0000          |
| t1             |
| t2             |
| t3             |
| t5             |
| t6             |
| t7             |
| t8             |
| t_1            |
| t_2            |
| tt_rename      |
| user           |
+----------------+
19 rows in set (Elapsed: 00:00:00.37)
```

示例 6：变更列名 b 为新列名 d。

```
gbase> desc tt_rename;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| c     | int(11)     | YES  |     | NULL    |       |
| b     | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (Elapsed: 00:00:00.14)

gbase> alter table tt_rename change b d varchar(11);
ERROR 1235 (42000): This version of GBase doesn't yet support 'ALTER column definition except increasing length of varchar'
gbase> alter table tt_rename change b d varchar(21);
Query OK, 0 rows affected (Elapsed: 00:00:00.02)
Records: 0  Duplicates: 0  Warnings: 0

gbase> desc tt_rename;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| c     | int(11)     | YES  |     | NULL    |       |
| d     | varchar(21) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (Elapsed: 00:00:00.00)
```

发现虽然可以修改varchar得长度，但是只能增长不能减短，按理说应该如此，如果存了数据，剪短会导致数据缺失。

示例 7：变更列的位置至最前。

```
gbase> alter table tt_rename modify d first;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corr
gbase> alter table tt_rename modify d varchar(21) first;
Query OK, 0 rows affected (Elapsed: 00:00:00.12)
Records: 0  Duplicates: 0  Warnings: 0

gbase> desc tt_rename;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| d     | varchar(21) | YES  |     | NULL    |       |
| a     | int(11)     | YES  |     | NULL    |       |
| c     | int(11)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (Elapsed: 00:00:00.06)
```

必须加上列类型。

示例 8：变更某列的位置到指定列的后面。

```
gbase> alter table tt_rename modify d varchar(21) after a;
Query OK, 0 rows affected (Elapsed: 00:00:00.11)
Records: 0  Duplicates: 0  Warnings: 0

gbase> desc tt_rename;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| d     | varchar(21) | YES  |     | NULL    |       |
| c     | int(11)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (Elapsed: 00:00:00.00)
```

插入数据

```
INSERT [INTO] [database_name.]table_name [(col_name,...)]
VALUES ({expr | DEFAULT},...),(...),...
or
INSERT [INTO] [database_name.]table_name [(col_name,...)]
SELECT ... FROM [database_name.]table_name ...
```

其中INSERT...VALUES 形式的语句基于明确的值插入记录行，这与Mysql语法是一致得。

INSERT ... SELECT 形式的语句从另一个或多个表中选取出值，并将其插入。

参数说明如下：

table_name：是要被插入数据的表。

col_name：指出语句指定的值赋给哪个列。

备注：

如果在 INSERT...VALUES 或 INSERT...SELECT 中没有指定 column 列，那么所有列的值必须在 VALUES()列表中或由 SELECT 提供。如果用户不知道表的列的次序，可以使用 DESC table_name 来查看。

使用关键词 DEFAULT，明确地把列设置为默认值。这样，编写向所有列赋值的 INSERT 语句时可以更容易，因为使用 DEFAULT 可以避免编写出不完整的、未包含全部列值的 VALUES 清单。如果不使用 DEFAULT，用户必须注明每一个列的名称，与 VALUES 中的每个值对应。

可以使用 DEFAULT (col_name)作为设置列默认值的一个更通用的形式。

如果 column 列表和 VALUES 列表都为空，INSERT 将创建一个行，它的每 一列都设置为它的默认值。

示例 1：INSERT INTO...

```
CREATE TABLE t0(id int) REPLICATED;
INSERT INTO t0 VALUES(1),(2),(3),(4);
```

```
gbase> create table t0(a int) replicated;
Query OK, 0 rows affected (Elapsed: 00:00:00.15)

gbase> insert into t0 values(1),(2),(3),(4);
Query OK, 4 rows affected (Elapsed: 00:00:00.05)
Records: 4  Duplicates: 0  Warnings: 0

gbase> select * from t0;
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
+------+
4 rows in set (Elapsed: 00:00:00.10)
```

示例2：INSERT INTO...SELECT...

```
insert into t0(a) select age from user limit 10;
select * from t0;
```

从之前 `user` 表中选出其 `age` 列，插入到刚刚创建的 `t0` .

```
gbase> insert into t0(a) select age from user limit 10;
Query OK, 2 rows affected (Elapsed: 00:00:00.02)
Records: 2  Duplicates: 0  Warnings: 0

gbase> select * from t0;
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   23 |
|   11 |
+------+
6 rows in set (Elapsed: 00:00:00.00)
```

示例 3：INSERT INTO ... VALUES(DEFAULT) 有默认值的！！

```
CREATE TABLE t00(id int DEFAULT 1) REPLICATED;
INSERT INTO t00 (id) VALUES(DEFAULT);
```

```
gbase> create table t00(id int default 1) replicated;
Query OK, 0 rows affected (Elapsed: 00:00:00.03)

gbase> INSERT INTO t00 (id) VALUES(DEFAULT);
Query OK, 1 row affected (Elapsed: 00:00:00.04)

gbase> select * from t00;
+------+
| id   |
+------+
|    1 |
+------+
1 row in set (Elapsed: 00:00:00.00)
```

示例 4：INSERT 时，自动更新 `TIMESTAMP` 列

```
CREATE TABLE t000(a timestamp , b int) DISTRIBUTED BY ('b');
INSERT INTO t000(b) VALUES(1);
INSERT INTO t000(b) VALUES(2);
```

```
gbase> CREATE TABLE t000(a timestamp , b int) DISTRIBUTED BY ('b');
Query OK, 0 rows affected (Elapsed: 00:00:00.04)

gbase> INSERT INTO t000(b) VALUES(1);
Query OK, 1 row affected (Elapsed: 00:00:00.06)

gbase> INSERT INTO t000(b) VALUES(2);
Query OK, 1 row affected (Elapsed: 00:00:00.05)

gbase> select * from t000;
+---------------------+------+
| a                   | b    |
+---------------------+------+
| 2020-03-05 19:44:23 |    1 |
| 2020-03-05 19:44:33 |    2 |
+---------------------+------+
2 rows in set (Elapsed: 00:00:00.02)
```

时间戳会自动记录。

修改数据

```
UPDATE [database_name.]table_name
SET col_name1 =expr1 [, col_name2 =expr2 ...]
[WHERE where_definition]
```

当更新列的值是一个合法的表达式时，也可以进行正确的更新赋值操作。语法跟 `Mysql` 的语法一致。

需要说明的是，UPDATE 操作不支持更新 DISTRIBUTED BY 列的值。

示例 1：简单的修改数据

```
gbase> select * from user;
+------+------+
| age  | name |
+------+------+
|   23 | veng |
|   11 | joy  |
+------+------+
2 rows in set (Elapsed: 00:00:00.00)

gbase> update user set age=12 where name='joy';
Query OK, 1 row affected (Elapsed: 00:00:00.03)
Rows matched: 1  Changed: 1  Warnings: 0

gbase> select * from user;
+------+------+
| age  | name |
+------+------+
|   23 | veng |
|   12 | joy  |
+------+------+
2 rows in set (Elapsed: 00:00:00.00)
```

示例2：UPDATE 时，更新数据行中的 `TIMESTAMP` 列的数值自动更新。

```
gbase> select * from t000;
+---------------------+------+
| a                   | b    |
+---------------------+------+
| 2020-03-05 19:44:23 |    1 |
| 2020-03-05 19:44:33 |    2 |
+---------------------+------+
2 rows in set (Elapsed: 00:00:00.01)

gbase> update t000 set b=3 where b>1;
Query OK, 1 row affected (Elapsed: 00:00:00.03)
Rows matched: 1  Changed: 1  Warnings: 0

gbase> select * from t000;
+---------------------+------+
| a                   | b    |
+---------------------+------+
| 2020-03-05 19:44:23 |    1 |
| 2020-03-05 19:44:33 |    3 |
+---------------------+------+
2 rows in set (Elapsed: 00:00:00.00)
```

问题：可以看出时间戳并没有改变，与参考手册上说的不一致。

这是该表的信息：

```
gbase> show create table t000;
+-------+--------------------------------------------------------------------------
-------------------------------+
| Table | Create Table
                               |
+-------+--------------------------------------------------------------------------
-------------------------------+
| t000  | CREATE TABLE "t000" (
  "a" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  "b" int(11) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace' |
+-------+--------------------------------------------------------------------------
-------------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

快速 UPDATE模式

快速 UPDATE 模式，即先删除符合更新条件的数据，然后再向表的末尾插 入需要更新的新数据。 相对于传统的行存储数据库来说，列存储的数据中 UPDATE 更新少量行时，操作效率相对来说是耗时的，因此， `GBase 8a MPP Cluster` 针对此特点，专门设计了快速 UPDATE 模式，用以提高数据更新操作。

快速 UPDATE 模式目前只支持针对表对象的操作。 要使用快速UPDATE模式，必须在客户端使用 SET gbase_fast_update =1;的命令打开快速 UPDATE 模式。更新大批量数据的时候建议使用默认 UPDATE模式，更新少量数据的时候建议使用快速 UPDATE 模式.

SET gbase_fast_update =0;表示关闭快速 UPDATE 模式。 SET gbase_fast_update =1;表示开启快速 UPDATE 模式。

删除数据

```
DELETE [FROM] [ database_name .] table_name [tbl_alias] [WHERE
where_definition]
```

其中，关键字[FROM]和表别名[tbl_alias]是可选关键字。跟 `mysql` 也是一致的. 当 DELETE 语句中包含别名时，可以省略 FROM 关键字。 `mysql` 并不能省去From。

示例：根据条件删除数据

```
gbase> select * from t0;
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   23 |
|   11 |
+------+
6 rows in set (Elapsed: 00:00:00.00)

gbase> delete from t0 where a>10;
Query OK, 2 rows affected (Elapsed: 00:00:00.01)

gbase> select * from t0;
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
+------+
4 rows in set (Elapsed: 00:00:00.00)
```

示例:删除全表数据。

```
DELETE FROM t0;
```

```
gbase> delete from t0;
Query OK, 4 rows affected (Elapsed: 00:00:00.01)

gbase> select * from t0;
Empty set (Elapsed: 00:00:00.00)
```

查找数据

```
SELECT
[ALL | DISTINCT | DISTINCTROW ]
select_expr, ...
[FROM table_references]
[WHERE where_definition]
[GROUP BY { col_name | expr | position}
, ...]
[HAVING where_definition]
[ORDER BY { col_name | expr | position}
[ASC | DESC] , ...]
[LIMIT {[offset,] row_count | row_count OFFSET offset }]
[INTO OUTFILE ' file_name ' export_options]
```

在 SELECT 关键字之后可以给出大量的选项，它们会影响到语句的操作。

ALL，DISTINCT 和 DISTINCTROW 选项指定了是否返回重复的行，默认为ALL（所有匹配的行都返回）。DISTINCT 和 DISTINCTROW 是同义的，用于删除结果集中重复的行。

select_expr：指查询显示的列，可以使用 AS 来为 SELECT 显示的列命名别名，别名不要和 SELECT 显示的列名重复。

table_references：指定从其中找出行的一个或多个表。它的语法在JOIN 语法中有描述。

where_definition：含有 WHERE 关键字，其后是查询所要满足的一个或多个条件的表达式，但是 where 中不能使用聚合函数。

分组 group by

```
GROUP BY { col_name | expr | position}, ... [HAVING
where_definition]
```

参数说明如下：

col_name ：指定分组的数据列，多列之间用,,'分隔。 col_name 可以是 SELECT 中使用 AS 定义的别名。

expr：指定分组的表达式，多列之间用,,'分隔。

注意：上面的 col_name 和 expr 中定义的数据列或表达式，可以不是,SELECT col_name_1,..., col_name_n FROM'之间的数据列，这一点是 GBase 8a MPP Cluster 中比较特殊的语法。

position：在,SELECT col_name_1,..., col_name_n FROM'之间的,col_name_1,..., col_name_n'的序号，position 是整数型数值，从 1 开始。例如：,SELECT stu_no,stu_name FROM stundent GROUP BY 1 ;'语句 中，,1'就是指代数据列 stu_no。

HAVING where_definition：使用GROUP BY子句对数据进行分组，对 GROUP BY 子句形成的分组，使用聚集函数计算各个分组的值，最后用 HAVING 子句过滤掉不符合条件的分组。

注意：

HAVING 子句中的每一个元素不必出现在 SELECT 列表中。 HAVING 子句限制的是分组，而不是行，因此可以使用聚合函数。

示例 1：按照性别分组，计算不同性别的人数。

```
gbase> select * from student;
+--------+---------+------------+
| stu_no | stu_sex | stu_name   |
+--------+---------+------------+
|      1 |       1 | 小明       |
|      2 |       0 | 小红       |
|      3 |       0 | 蔡徐坤     |
|      4 |       1 | 小刚       |
|      5 |       1 | 小V        |
+--------+---------+------------+
5 rows in set (Elapsed: 00:00:00.00)

gbase> select stu_sex,count(*) from student group by stu_sex;
+---------+----------+
| stu_sex | count(*) |
+---------+----------+
|       1 |        3 |
|       0 |        2 |
+---------+----------+
2 rows in set (Elapsed: 00:00:00.00)
```

排序 order by

```
ORDER BY { col_name | expr | position} [ASC | DESC] , ...
```

参数说明如下： ORDER BY 用于对结果集进行排序，数据列列名称或者表达式。

col_name ：指定排序的数据列，多列之间用,,'分隔。 col_name 可以是 SELECT 中使用 AS 定义的别名。

expr：指定排序的表达式，多列之间用,,'分隔。

position：在,SELECT col_name_1,..., col_name_n FROM '之间的,col_name_1,..., col_name_n'的序号，position 是整数型数值，从,1'开始。

ASC | DESC：如果希望对记录进行排序，可以使用 ASC 或 DESC 关键字来指定排序规则，ASC 代表升序规则，DESC 代表降序规则。默认按照升序对记录进行排序。

示例：从大到小排列数学成绩。

```
gbase> select * from student_course;
+--------+-------+-------------+
| stu_no | sorce | course_name |
+--------+-------+-------------+
|      1 |    60 | math        |
|      1 |    70 | english     |
|      2 |    70 | math        |
|      2 |    80 | english     |
|      3 |    75 | math        |
|      3 |    90 | english     |
|      4 |    95 | math        |
|      4 |    97 | english     |
|      5 |    97 | math        |
|      5 |    96 | english     |
+--------+-------+-------------+
10 rows in set (Elapsed: 00:00:00.00)

gbase> select * from student_course where course_name='math' order by sorce desc;
+--------+-------+-------------+
| stu_no | sorce | course_name |
+--------+-------+-------------+
|      5 |    97 | math        |
|      4 |    95 | math        |
|      3 |    75 | math        |
|      2 |    70 | math        |
|      1 |    60 | math        |
+--------+-------+-------------+
5 rows in set (Elapsed: 00:00:00.00)
```

限制数量 limit

```
LIMIT {[offset,] row_count | row_count OFFSET offset}
```

LIMIT row_count：row_count 是一个整数型数值，表示从记录集开始返回 row_count行结果集。如果row_count指定的数值大于SELECT后的结果集，那么 row_count 将不起作用。

LIMIT row_count 等价于 LIMIT 0, row_count 或者等价于 LIMIT row_count OFFSET 0

LIMIT row_count OFFSET offset：row_count 指定返回结果集的行数，offset 指定结果集的偏移量，初偏移量的起始值是 0（而不是 1），即偏移量 0 对应 SELECT 返回的第一行结果集。

示例:

```
gbase> select * from student limit 3;
+--------+---------+------------+
| stu_no | stu_sex | stu_name   |
+--------+---------+------------+
|      1 |       1 | 小明       |
|      2 |       0 | 小红       |
|      3 |       0 | 蔡徐坤     |
+--------+---------+------------+
3 rows in set (Elapsed: 00:00:00.00)
```

关联表 JOIN

跟mysql的语法一致，有全连接，左连接，右连接，内连接.

示例：内连接

```
gbase> select s.stu_no,stu_name,c.c_id,c.name from student s ,stu_course sc, course c where s.stu_no=sc.stu_no and sc.c_id = c.c_id;
+--------+------------+------+---------+
| stu_no | stu_name   | c_id | name    |
+--------+------------+------+---------+
|      1 | 小明       |    1 | math    |
|      1 | 小明       |    2 | english |
|      3 | 蔡徐坤     |    3 | chinese |
|      3 | 蔡徐坤     |    2 | english |
|      4 | 小刚       |    4 | physics |
|      4 | 小刚       |    2 | english |
|      5 | 小V        |    4 | physics |
|      5 | 小V        |    1 | math    |
+--------+------------+------+---------+
8 rows in set (Elapsed: 00:00:00.00)
```

示例：左连接

```
gbase> select s.stu_no,stu_name,c.c_id,c.name from student s left join stu_course sc on s.stu_no=sc.stu_no
    -> left join course c on sc.c_id = c.c_id;
+--------+------------+------+---------+
| stu_no | stu_name   | c_id | name    |
+--------+------------+------+---------+
|      1 | 小明       |    1 | math    |
|      1 | 小明       |    2 | english |
|      3 | 蔡徐坤     |    3 | chinese |
|      3 | 蔡徐坤     |    2 | english |
|      4 | 小刚       |    4 | physics |
|      4 | 小刚       |    2 | english |
|      5 | 小V        |    4 | physics |
|      5 | 小V        |    1 | math    |
|      2 | 小红       | NULL | NULL    |
+--------+------------+------+---------+
9 rows in set (Elapsed: 00:00:00.00)
```

没有选课的人也会 出现。

示例：右连接

```
gbase> select sc.stu_no,s.stu_name ,sc.c_id, c.name from stu_course sc right join course c on c.c_id = sc.c_id left join student s on  sc.stu_no=s.stu_no ;
+--------+------------+------+-----------+
| stu_no | stu_name   | c_id | name      |
+--------+------------+------+-----------+
|      1 | 小明       |    1 | math      |
|      1 | 小明       |    2 | english   |
|      3 | 蔡徐坤     |    3 | chinese   |
|      3 | 蔡徐坤     |    2 | english   |
|      4 | 小刚       |    4 | physics   |
|      4 | 小刚       |    2 | english   |
|      5 | 小V        |    4 | physics   |
|      5 | 小V        |    1 | math      |
|   NULL | NULL       | NULL | geography |
+--------+------------+------+-----------+
9 rows in set (Elapsed: 00:00:00.00)
```

示例：全连接

```
gbase> select s.stu_no,s.stu_name,sc.c_id,c.name from stu_course sc full join course c on c.c_id = sc.c_id full join student s on s.stu_no = sc.stu_no;
+--------+------------+------+-----------+
| stu_no | stu_name   | c_id | name      |
+--------+------------+------+-----------+
|      1 | 小明       |    1 | math      |
|      1 | 小明       |    2 | english   |
|      3 | 蔡徐坤     |    3 | chinese   |
|      3 | 蔡徐坤     |    2 | english   |
|      4 | 小刚       |    4 | physics   |
|      4 | 小刚       |    2 | english   |
|      5 | 小V        |    4 | physics   |
|      5 | 小V        |    1 | math      |
|      2 | 小红       | NULL | NULL      |
|   NULL | NULL       | NULL | geography |
+--------+------------+------+-----------+
10 rows in set (Elapsed: 00:00:00.00)
```

并集 union

```
SELECT ...UNION [ALL | DISTINCT]SELECT ...
[UNION [ALL | DISTINCT]
SELECT ...]
```

UNION 用来将多个 SELECT 语句的结果合并到一个结果集中。

不同的数据类型，不能使用 UNION 和 UNION ALL，例如：数值型，字符型，日期和时间型之间不能使用 UNION 和 UNION ALL，但是 DATETIME 和TIMESTAMP类型可以使用 UNION和 UNION ALL，其它的日期和时间类型则不行,总而言之，同一种类型格式一样的，才能合并啊。

如果 UNION 和 UNION ALL 两边的数据类型为 CHAR 类型，进行 UNION 和UNION ALL 操作时，结果返回 VARCHAR 类型。

例如：SELECT CHAR(10) UNION SELECT CHAR(255)的结果集为 VARCHAR(255)。

UNION 和 UNION ALL 结果由小的数据类型向大的数据类型转换，如：INT → BIGINT → DECIMAL → DOUBLE。

例如：SELECT INT UNION SELECT DOUBLE 的结果集为 DOUBLE 型。

NULL 可以和任何类型做 UNION 和 UNION ALL。

例如：SELECT NULL UNION SELECT 1;

如果在多个SELECT的UNION查询中，同时存在UNION [DISTINCT]和UNION ALL，那么 UNION ALL 会被忽略，最终返回UNION [DISTINCT]后的结果集（过滤掉重复的记录行）。

如果希望使用 ORDER BY 或 LIMIT 子句来分类或限制整个 UNION 结果，可以给单独的 SELECT 语句加上括号或者把 ORDER BY 或 LIMIT 置于最后。

```
gbase> select *from student;
+--------+---------+-----------+
| stu_no | stu_sex | stu_name  |
+--------+---------+-----------+
|      1 |       1 | 小明      |
|      2 |       0 | 小红      |
|      3 |       0 | 蔡徐坤    |
|      4 |       1 | 小刚      |
|      5 |       1 | 小V       |
+--------+---------+-----------+
5 rows in set (Elapsed: 00:00:00.10)

gbase> select stu_name from student where stu_name like '%小%'
    -> union
    -> select stu_name from student where stu_name like '%坤%'
    -> order by stu_name
    -> limit 5;
+-----------+
| stu_name  |
+-----------+
| 小V       |
| 小刚      |
| 小明      |
| 小红      |
| 蔡徐坤    |
+-----------+
5 rows in set (Elapsed: 00:00:00.10)
```

这种 ORDER BY 不能使用包括表名的列引用（如，table_name.col_name格式的名字）。相对的，可以在第一个 SELECT 语句中提供一个列的别名并在ORDER BY 中引用这个别名。

交集 intersect

```
SELECT ...
INTERSECT
SELECT ...;
```

类比于并集是一样的道理。

```
gbase> select * from student;
+--------+---------+-----------+
| stu_no | stu_sex | stu_name  |
+--------+---------+-----------+
|      1 |       1 | 小明      |
|      2 |       0 | 小红      |
|      3 |       0 | 蔡徐坤    |
|      4 |       1 | 小刚      |
|      5 |       1 | 小V       |
+--------+---------+-----------+
5 rows in set (Elapsed: 00:00:00.00)

gbase> select stu_no,stu_name from student where stu_no>2  intersect  select stu_no,stu_name from student where stu_no<5;
+--------+-----------+
| stu_no | stu_name  |
+--------+-----------+
|      3 | 蔡徐坤    |
|      4 | 小刚      |
+--------+-----------+
2 rows in set (Elapsed: 00:00:00.01)

gbase>
```

差集 minus

```
SELECT ... MINUS SELECT ... ;
```

类比于并集是一样的道理。

```
gbase> select * from student;
+--------+---------+-----------+
| stu_no | stu_sex | stu_name  |
+--------+---------+-----------+
|      1 |       1 | 小明      |
|      2 |       0 | 小红      |
|      3 |       0 | 蔡徐坤    |
|      4 |       1 | 小刚      |
|      5 |       1 | 小V       |
+--------+---------+-----------+
5 rows in set (Elapsed: 00:00:00.00)

gbase> select stu_no,stu_name from student where stu_no>1  minus  select stu_no,stu_name from student where stu_no>3;
+--------+-----------+
| stu_no | stu_name  |
+--------+-----------+
|      2 | 小红      |
|      3 | 蔡徐坤    |
+--------+-----------+
2 rows in set (Elapsed: 00:00:00.01)
```

show语句管理 - 常用

show columns

```
SHOW [FULL] COLUMNS FROM table_name [FROM database_name ] [LIKE
' pattern ']
```

SHOW COLUMNS 显示一个给定表中列的信息。该语句在视图中同样适用。

可以将 table_name FROM database_name 语法修改为 database_name.table_name，下面的两个语句是等价的：SHOW COLUMNS FROM t FROM test; SHOW COLUMNS FROM test.t; SHOW FIELDS 和 SHOW COLUMNS 的作用相同。

```
gbase> show columns from student from test;
+-----------+-------------+------+-----+---------+--------+
| Field     | Type        | Null | Key | Default | Extra  |
+-----------+-------------+------+-----+---------+--------+
| stu_no    | int(11)     | YES  |     | NULL    |        |
| stu_sex   | tinyint(1)  | YES  |     | NULL    |        |
| stu_name  | varchar(10) | YES  |     | NULL    |        |
+-----------+-------------+------+-----+---------+--------+
3 rows in set (Elapsed: 00:00:00.00)
```

SHOW CREATE DATABASE

```
SHOW CREATE {DATABASE | SCHEMA} database_name
```

显示创建了给定数据库的 CREATE DATABASE 语句。可以用 SHOW CREATE SCHEMA 语句实现相同的功能。

```
gbase> show create database test;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| test     | CREATE DATABASE "test" /*!40100 DEFAULT CHARACTER SET utf8 */ |
+----------+-----------------------------------------------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

SHOW DATABASES

```
SHOW {DATABASES | SCHEMAS} [LIKE ' pattern ']
```

SHOW DATABASES 列出在 GBase 8a MPP Cluster 服务器主机上的数据库。 除非拥有全局的SHOW DATABASES权限，用户只能看到自己拥有权限的数据库。

SHOW CREATE TABLE

```
SHOW CREATE TABLE [database_name.]table_name
```

显示创建了给定表的 CREATE TABLE 语句。该语句在视图中也使用。

```
gbase> show create table student;
+---------+-----------------------------------------------------------------------------------------------------------
-----------------------------------+
| Table   | Create Table
                                    |
+---------+-----------------------------------------------------------------------------------------------------------
-----------------------------------+
| student | CREATE TABLE "student" (
  "stu_no" int(11) DEFAULT NULL,
  "stu_sex" tinyint(1) DEFAULT NULL,
  "stu_name" varchar(10) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace' |
+---------+-----------------------------------------------------------------------------------------------------------
-----------------------------------+
1 row in set (Elapsed: 00:00:00.00)

gbase>
```

nodejs连接数据库

使用 `mysql` 库进行连接。

安装：

```
npm install mysql
```

配置如下：

```
var mysql      = require('mysql');
var connection = mysql.createConnection({
  host     : '192.168.75.129',
  user     : 'root',
  password : '',
  database : 'test',
  port: 5258,
});
```

测试：

```
connection.connect();

connection.query('SELECT * from student', function (error, results, fields) {
  if (error) throw error;
  console.table(results);
});
connection.end();
```

结果：

(index)	stu_no	stu_sex	stu_name
0	1	1	'小明'
1	2	0	'小红'
2	3	0	'蔡徐坤'
3	4	1	'小刚'
4	5	1	'小V'

完全适用， `Gbase` 的语法跟 `mysql` 的语法基本相似，故而可以如此使用。

`eggjs-sequelize` 对 `Gbase` 的支持

测试其基本的增删改查是否支持。

配置

```
module.exports = {
  dialect: 'mysql',
  host: '192.168.75.129',
  port: 5258,
  database: 'test',
  user: 'root',
  password: '',
};
```

model:

```
module.exports = app => {
  const { STRING, INTEGER } = app.Sequelize;
  const student = app.model.define('student',
    {
      stu_no: { type: INTEGER, primaryKey: true },
      stu_sex: { type: INTEGER(1) },
      stu_name: { type: STRING(10) },
    },
    {
      freezeTableName: true,
      timestamps: false,
    });
  return student;
};
```

这里实际上并没有主键，但是如果不舍主键的话，则会默认有一个 `id` 的主键，故而还是设 `stu_no` 为主键。

查询数据

service:

```
    async findAll() {
      const { ctx } = this;
      return await ctx.model.Student.findAll();
    }
```

test:

```
const { app } = require('egg-mock/bootstrap');

describe('test/app/service/student.test.js', () => {
  it('student findAll()', async () => {
    const ctx = app.mockContext();
    const student = await ctx.service.student.findAll();
    console.log(JSON.stringify(student));
  });
});
```

以上是测试查询代码。

结果：

```
  test/app/service/student.test.js
[{"stu_no":1,"stu_sex":1,"stu_name":"小明"},{"stu_no":2,"stu_sex":0,"stu_name":"小红"},{"stu_no":3,"stu_sex":0,"stu_name":"蔡徐坤"}
,{"stu_no":4,"stu_sex":1,"stu_name":"小刚"},{"stu_no":5,"stu_sex":1,"stu_name":"小V"}]
    √ student findAll()
```

是没有问题的。

增加数据

service:

```
    async addStudent({ stu_no, stu_sex, stu_name }) {
      const { ctx } = this;
      await ctx.model.Student.create({ stu_no, stu_sex, stu_name });
    }
```

test:

```
    it('student insertOne', async () => {
      const ctx = app.mockContext();
      const newStudent = {
        stu_no: 8,
        stu_sex: 1,
        stu_name: '小新',
      };
      await ctx.service.student.addStudent(newStudent);
    });
```

结果：

```
gbase> select * from student;
+--------+---------+-----------+
| stu_no | stu_sex | stu_name  |
+--------+---------+-----------+
|      1 |       1 | 小明      |
|      2 |       0 | 小红      |
|      3 |       0 | 蔡徐坤    |
|      4 |       1 | 小刚      |
|      5 |       1 | 小V       |
|      8 |       1 | 小新      |
+--------+---------+-----------+
6 rows in set (Elapsed: 00:00:00.34)
```

修改数据

service:

```
    /**
     * 根据学号修改学生姓名
     * @param {int} stu_no 学号
     * @param {string} stu_name  修改后的姓名
     */
    async updateNameById(stu_no, stu_name) {
      const { ctx } = this;
      await ctx.model.Student.update({
        stu_name,
      }, { where: { stu_no } });
    }
```

test:

```
    it('student update', async () => {
      const ctx = app.mockContext();
      await ctx.service.student.updateNameById(8, '大新');
    });
```

结果：

```
gbase> select * from student;
+--------+---------+-----------+
| stu_no | stu_sex | stu_name  |
+--------+---------+-----------+
|      1 |       1 | 小明      |
|      2 |       0 | 小红      |
|      3 |       0 | 蔡徐坤    |
|      4 |       1 | 小刚      |
|      5 |       1 | 小V       |
|      8 |       1 | 大新      |
+--------+---------+-----------+
6 rows in set (Elapsed: 00:00:00.09)
```

删除数据

service:

```
  /**
   * 根据学号修改学生姓名
   * @param {int} stu_no 学号
   * @param {string} stu_name  修改后的姓名
   */
  async updateNameById(stu_no, stu_name) {
    const { ctx } = this;
    await ctx.model.Student.update({
      stu_name,
    }, { where: { stu_no } });
  }
```

test:

```
  it('student delete', async () => {
    const ctx = app.mockContext();
    await ctx.service.student.deleteById(8);
  });
```

result:

```
gbase> select * from student;
+--------+---------+-----------+
| stu_no | stu_sex | stu_name  |
+--------+---------+-----------+
|      1 |       1 | 小明      |
|      2 |       0 | 小红      |
|      3 |       0 | 蔡徐坤    |
|      4 |       1 | 小刚      |
|      5 |       1 | 小V       |
+--------+---------+-----------+
5 rows in set (Elapsed: 00:00:00.00)
```

由此可发现，基本的增删改查，都是支持的。

数据的备份与恢复

数据的备份用的 `Gbase` 的 `rcman` 工具。

对数据库进行一次全备份

1. 先创建备份目录

```
mkdir ~/gbase-back
```

2. 打开数据库服务

```
gbase.server start
```

3. 进入到 `rcman` 工具所在目录，默认在 `Gbase/server/bin` 下.

```
cd ~/Gbase/server/bin
```

4. 进入 `rcman` 工具，并配置备份目录

```
rcman -d ~/gbase-back
```

```
RCMAN>^Cveng@veng-PC:~/GBase/server/bin$ rcman -d ~/gbase-back
Welcome to the GBase backup and recovery tool.

RCMAN>
```

5. 进行一次全备份

```
backup level 0
```

```
RCMAN>backup level 0
------------------------------
Estimate backup require disk space begin
Estimate backup require disk space end
------------------------------
------------------------------
Backup begin
Backup end
------------------------------
------------------------------
Check Backup Point begin
Check Backup Point end
------------------------------
------------------------------
Record Backup Point begin
Record Backup Point end
------------------------------
Backup success!
```

备份结束后会有提示，`Backup success`.

在数据库中增加数据库，并插入数据，然后进行一次增量备份

1. 我新建一个数据库，然后新建一个表，`student`并插入了一些数据，不再详细说明.

```
gbase> create database veng;
Query OK, 1 row affected (Elapsed: 00:00:00.03)

gbase> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| performance_schema |
| gbase              |
| gclusterdb         |
| gctmpdb            |
| test               |
| veng               |
+--------------------+
7 rows in set (Elapsed: 00:00:00.00)

gbase> use veng;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)

gbase> create table student(id int,name varchar(40),sex char(1) not null);
Query OK, 0 rows affected (Elapsed: 00:00:00.24)

gbase> insert into student values(1,'一号','m'),(2,'二号','f'),(3,'test','m');
Query OK, 3 rows affected (Elapsed: 00:00:00.22)
Records: 3  Duplicates: 0  Warnings: 0

gbase> select * from stduent;
ERROR 1146 (42S02): Table 'veng.stduent' doesn't exist
gbase> select * from student;
+------+--------+-----+
| id   | name   | sex |
+------+--------+-----+
|    1 | 一号   | m   |
|    2 | 二号   | f   |
|    3 | test   | m   |
+------+--------+-----+
3 rows in set (Elapsed: 00:00:00.06)
```

2. 开始进行增量备份，跟刚才一样，进入`rcman`工具。

```
rcman -d ~/gbase-back
```

3. 进行增量备份，刚才`level`为`0`时，是全备份，`1`则是增量备份.

```
backup level 1
```

4. 查看一下备份的文件

```
1    show backup
```

```
RCMAN>show backup
cycle    point    backupdir
0        0        20200316132318.lv0
0        1        20200316134742.lv1
```

我们这里备份了两次，一次全量备份，一次增量备份。

将数据库恢复到第一次全备份的状态下

1. 恢复备份的时候，需要把 `Gbase` 服务停掉.

```
1    gbase.server stop
```

2. 进入 `rcman` 工具.

```
1    rcman -d ~/gbase-back
```

3. 根据之前备份文件中的， `cycle` 以及 `point` 数来恢复指定备份数据。

```
RCMAN>show backup
cycle    point    backupdir
0        0        20200316132318.lv0
0        1        20200316134742.lv1
```

```
1    recover 0 0
```

```
RCMAN>recover 0 0
--------------------------------
Check Backup Point begin!
Check Backup Point success!
--------------------------------
--------------------------------
Check Disk Space begin!
Check Disk Space end!
--------------------------------
--------------------------------
recovery start
recovery end
--------------------------------
Recover success!
You must restart GBASE!!
```

此时我们先恢复到原来第一次的备份，`0 0` .

4. 然后我们启动 `Gbase` 服务，再次查看数据库是否已经恢复.

```
veng@veng-PC:~$ gbase.server start
[ ok ing GBase..
veng@veng-PC:~$ gbase -uroot

GBase client Free Edition 8.6.2.43-R7-free.110605. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| performance_schema |
| gbase              |
| gclusterdb         |
| gctmpdb            |
| test               |
+--------------------+
6 rows in set (Elapsed: 00:00:00.00)
```

发现之前创建的数据库 `veng` 已经没有了，验证了备份恢复成功。

将数据恢复到增量备份的状态下

跟之前全量备份的操作一样，只是 `recover` 的 `cycle point` 不同。

```
recover 0 1
```

```
RCMAN>recover 0 1
--------------------------------
Check Backup Point begin!
Check Backup Point success!
--------------------------------
--------------------------------
Check Disk Space begin!
Check Disk Space end!
--------------------------------
--------------------------------
recovery start
recovery end
--------------------------------
Recover success!
You must restart GBASE!!
```

然后我们再次查看数据库，进行验证.

```
veng@veng-PC:~$ gbase.server start
[ ok ing GBase..
veng@veng-PC:~$ gbase -uroot

GBase client Free Edition 8.6.2.43-R7-free.110605. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase>
gbase> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| performance_schema |
| gbase              |
| gclusterdb         |
| gctmpdb            |
| test               |
| veng               |
+--------------------+
7 rows in set (Elapsed: 00:00:00.00)

gbase> select * from veng.student;
+------+--------+-----+
| id   | name   | sex |
+------+--------+-----+
|    1 | 一号   | m   |
|    2 | 二号   | f   |
|    3 | test   | m   |
+------+--------+-----+
3 rows in set (Elapsed: 00:00:00.00)
```

我们发现数据已经恢复到了增量备份的状态.

删除备份文件

同样进入到 `rcman` 工具,然后:

```
delete last
```

删除最近的一次备份文件.

```
RCMAN>show backup
cycle    point    backupdir
0        0        20200316132318.lv0
0        1        20200316134742.lv1

RCMAN>delete last
delete backup cycle 0 point 1 start
delete backup cycle 0 point 1 end
RCMAN>show backup
cycle    point    backupdir
0        0        20200316132318.lv0

RCMAN>delete last
delete backup cycle 0 point 0 start
delete backup cycle 0 point 0 end
RCMAN>show backup
There is no backup point!
```

`rcman` 的其他命令，可以输入 `help`

```
1    RCMAN>help
```

```
RCMAN>help

   You can do backup and recovery for whole instance or single table
-------------------------------------------------------------------------------------
SHOW BACKUP                                          -- lookup all backup points of instance
BACKUP LEVEL <0|1>                                   -- 0 means whole backup,1 means increment backup
BACKUP TABLE <DBNAME1.TBNAME1,DBNAME2.TBNAME2,....> LEVEL <0|1>
                                                     -- backup multi tables
BACKUP DATABASE <DBNAME> LEVEL <0|1>                 -- backup database
RECOVER [<CYCLE_NO> [POINT_NO]]                      -- recover to last backup point or cycle or point_id
RECOVER TABLE <DBNAME1.TBNAME1,DBNAME2.TBNAME2,....> <CYCLE> <POINT_NO>
                                                     -- recover multi tables to point_id
RECOVER DATABASE <DBNAME> <CYCLE_NO> <POINT_NO>      -- recover database to point_id
DELETE <CYCLE_NO|LAST>                               -- delete backup cycle or last point
CLEANUP                                              -- delete invalid backup data
QUIT                                                 -- quit from rcman
EXIT                                                 -- exit from rcman
HELP                                                 -- show the help info
-------------------------------------------------------------------------------------
```

首先我们看到删除指令是后面追加 `cycle` 或者 `last`.

我们这里不能用 `delete 0`，因为 `cycle = 0` 时，这是最开始的备份文件，故不能用 `delete 0` 进行删除.

从当中，可以看到对某个 `database` 或者 `table` 的进行备份和恢复，与之前的大同小异.