# MySQL 到 GBase 8s 迁移

## 技术指南



www.gbase.cn

# 目 录

# 1. 概述

将数据库从 MySQL 迁移到 GBase 8s 主要包括三个步骤： 数据库架构迁移(Database/DDL)、数据迁移(Data)和应用迁移(Application)。本文将以这三个步骤为主线，介绍 GBase 8s 不同于 MySQL 的技术特点，以及在迁移过程中的这两个数据库差异的转换方法与技巧。

与其他软件开发项目一样，数据库的迁移需要谨慎的规划以及良好的方法以确保其成功。这其中，数据库的设计至关重要，特别是关系型数据库的结构设计。可以通过新的数据库技术来替代之前使用的旧方式。在进行 MySQL 数据库迁移的时候，要按照 GBase 8s 的技术特点，进行一系列的数据库结构及应用程序的调整，有助于今后应用的平稳运行。在进行迁移项目之前，一定要对上述因素有一个完整的把握，这样可以到达事半功倍的效果，同时可以在一定程度上避免不必要的麻烦。

本文基于 GBase 8s v8.8 3.0.0_1 版本和 MySQL（基于 InnoDB 引擎）来介绍。

# 2. 概念及对比

## 2.1 实例

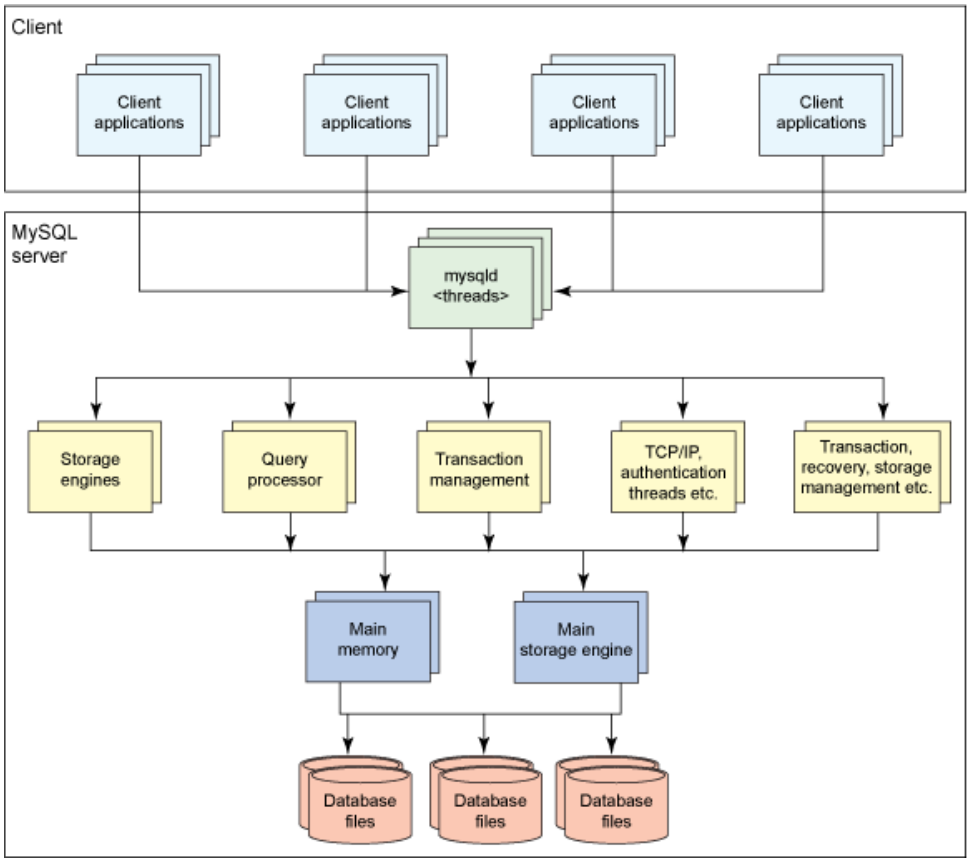

图 2.1.1 MySQL 架构和进程概览

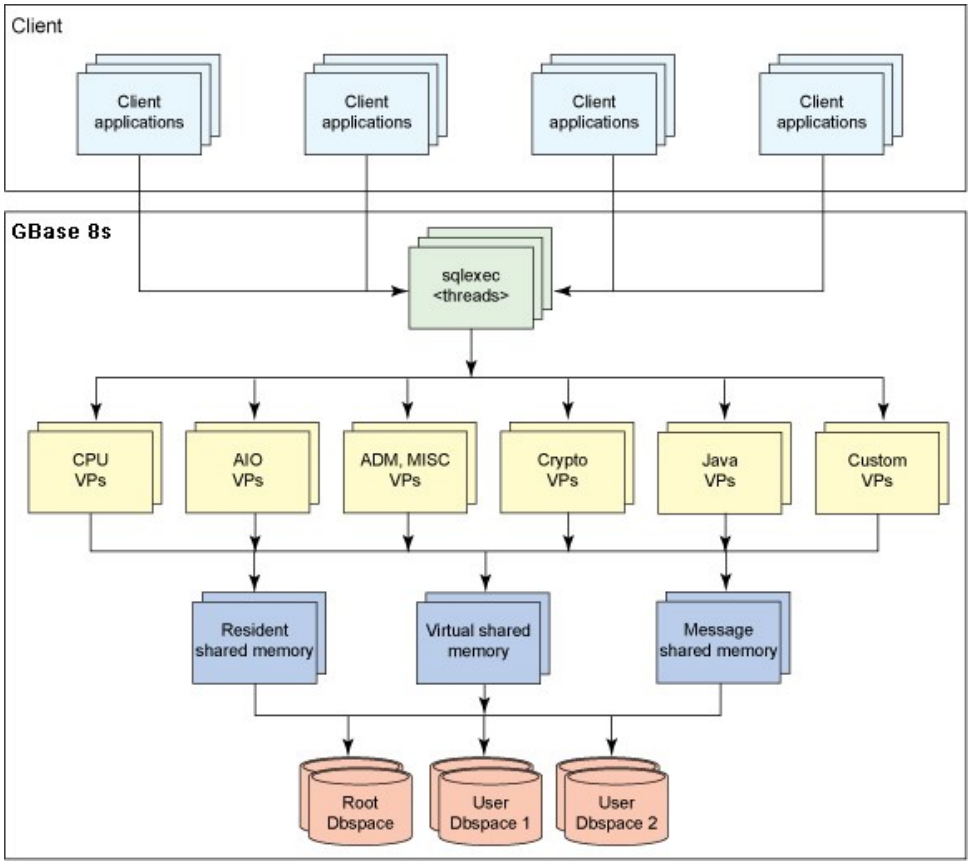


图 2.1.2 GBase 8s 架构和进程概览

MySQL 和 GBase 8s 都使用基于线程的架构。一个 MySQL 实例（如 2.1.1 所示）能管理多个数据库。一个实例中的所有 MySQL 数据库共享一个名为 INFORMATION_SCHEMA 的公共系统目录。

GBase 8s 中的实例与 MySQL 中的实例类似，但它包含自己的系统目录、存储空间、缓冲区池、线程等等。

包含多个数据库的单个实例可以在 MySQL 和 GBase 8s 中可视化。当有几个 MySQL 实例运行在一台机器上且每个实例管理几个数据库时，可以将每个 MySQL 实例作为 GBase 8s 进行迁移。

## 2.2 数据库（DataBase）

GBase 8s 中数据库（database）的用最简明的一句话来解释，是一系列包含特定用途对象的集合。MySQL 中基本上是相同的概念。

## 2.3 存储（Storage）

GBase 8s 中存储相关的主要概念，从大到小分如下排列。 GBase 8s 与 MySQL（InnoDB）在存储方

面的概念类似，对应关系如下。

| 序号 | GBase 8s | MySQL（InnoDB） |
|---|---|---|
| 1 | 数据空间（dbspace） | 表空间（tablespace） |
| 2 | 块（chunk） | |
| 3 | 表空间（tablespace） | 段（segment） |
| 4 | 区段（extent） | 区段（extent） |
| 5 | 页（page） | 页（page） |

GBase 8s 存储相关限制

| GBase 8s 存储相关限制 | 最大值 |
|---|---|
| 一页包含的数据行数 | 255 |
| 一个表或者分片最大的行数 | 4,277,659,295 |
| 一个表或者分片最大的数据页数 | 16,775,134 |
| 一个表或者分片最大占用的空间大小<br>（除 BLOB,CLOB,BYTE,TEXT 以外） | 2K page size = 33,818,670,144<br>4K page size = 68,174,144,576<br>8K page size = 136,885,093,440<br>12K page size = 205,596,042,304<br>16K page size = 274,306,991,168 |
| 一行记录的最大 rowsize | 2.0 版本：32,767<br>3.0 版本：40M |
| 表最大支持的字段数 | 32,767 |
| 一个复合索引支持最多的列数 | 16 |
| 一个函数索引的字段数 | 102 (for C UDRs)<br>341 (for SPL or Java™ UDRs) |
| 一个索引的字节数（对于给定的页大小） | 2K page size = 387<br>4K page size = 796<br>8K page size = 1615<br>12K page size = 2435<br>16K page size = 3254 |
| 单个 SQL 语句的最大长度 | 受限于可用内存 |
| 一个 Instance 最大支持的 database 数 | 21,000,000 |
| 一个 Instance 最大支持的 tables 数 | 477,102,080 |
| 一个 Instance 最大支持的活动用户数 active users | 32,767 |
| 一个 session 最大同时访问的 database 数 | Windows: 8<br>UNIX: 32 |
| 最大的页清理进程 page-cleaner | 128 |
| 一个 dbspace 上最大支持的 partition 数 | 4K page size 1,048,445<br>2K page size 1,048,314 |
| 一个用户同时可以使用 Lock Table 锁住的表的个数 | 32 |
| 一个 dbspace 的最大空间 | 128 PB |
| 一个 chunk 的最大空间 | 4 TB |
| 一个 Instance 最多支持的 chunks 数 | 32,766 |
| 一个 chunk 最大存储的数据页数 pages | 2,000,000,000 |

| | |
|---|---|
| 一个 Instance 最大支持的 dbspace 数 | 2047 |
| 一个 Instance 最大支持的存储空间 | 128 PB |

## 2.4 锁（Locks）

建表语句中的 LOCK MODE 子句指定了 GBase 8s 的锁模式：行级锁或页级锁。GBase 8s 中默认是页级锁(可通过配置参数 DEF_TABLE_LOCKMODE 或环境变量修改这个默认值)，而 MySQL（InnoDB）中默认是行级锁。

## 2.5 隔离级别（Isolation）

GBase 8s 为应用提供的隔离级别及对应关系请参考下表。

| GBase 8s | MySQL | ANSI |
|---|---|---|
| DIRTY READ | READ UNCOMMITTED | READ UNCOMMITTED |
| **COMMITTED READ 默认** | **READ COMMITTED 默认** | READ COMMITTED |
| REPEATABLE READ | REPEATABLE READ | REPEATABLE READ |
| REPEATABLE READ | SERIALIZABLE | SERIALIZABLE |
| CURSOR STABILITY | | |
| COMMITTED READ LAST COMMITTED | | |

GBase 8s 提供五种主要隔离级别，它们为处理数据并发性问题提供一些机制：

脏读隔离（读未提交）

提交读隔离（读提交）

游标稳定性隔离

可重复读隔离（可重复读和可序列化）

最后提交读隔离（读提交，最后一次提交）

**DIRTY READ 脏读隔离级别（读未提交）**

脏读隔离不需要在读取的行上使用锁，因此不检查所需的行上是否有锁。使用该隔离级别的会话可能会得到脏数据，即该数据被另一个会话更新但尚未提交。

对于非日志数据库，这是唯一可以使用的隔离级别。

以上列出的所有 3 种现象都可以在这个隔离级别上出现。

**COMMITTED READ 提交读隔离级别（读提交）**

提交读隔离仅允许会话读取已提交的行。它不在正在读取的行上放置锁，但检查该行是否放置了锁。

因此这个隔离级别不会发生脏读。

**CURSOR STABILITY 游标稳定性隔离级别**

游标稳定性隔离在读取的行上放置一个共享锁，并且在锁定下一个行时释放该锁。

这个隔离级别能够阻止脏读和非重复读现象。

**REPEATABLE READ 可重复读隔离级别**

在 GBase 8s 中实现的可重复读隔离级别相当于 ANSI SQL 92 中的可重复读和可序列化。

这个隔离级别除了能够阻止脏读和非重复读现象之外，它还能阻止伪读现象。

如果数据库服务器需要为使用可重复读隔离的会话对表执行序列读，它将在该表上放置一个共享锁。不过，如果使用了索引扫描的话，共享锁仅能锁定与受影响的行相关的索引键。

语句 SET ISOLATION COMMITTED READ 加上 LAST COMMITTED 关键字选项会减少锁冲突的风险。此语法告诉服务器返回最近提交的行版本，即使另一个并发会话持有一个独占锁。

与此相关的数据库参数 USELASTCOMMITTED 指定一种能使 COMMITTED READ 隔离级别的 LAST COMMITTED 功能隐性生效的隔离级别。

## 2.6 标识符（Identifiers）

数据库对象的名字由标识符来表示，如表名、字段名称、索引名、视图名、存储过程名称等等。MySQL（InnoDB）最大表名长度 64 个字节，可以包括字符、数字、“_”、“$”、“#”等任意可见字符（反引号“`“除外），若遇上特殊字符或者系统保留关键字时，可使用反引号包围标识符。GBase 8s 最大表名长度 128 个字节，可以包括字符、数字、“_”、“$”等，但不支持“#”和”`”等特殊字符，表名的第一个字符应当为普通字符，不符合 GBase 8s 命名规则的标识符，需要在移植时用正确的名称替换。GBase 8s 在 UTF-8 环境下支持中文的表名和字段名。

## 2.7 关键字（Keywords）

GBase 8s 的关键字（保留字）有与 MySQL 有着大量的不同，具体包括的保留字请参考附录 A。

## 2.8 大小写（Case sensitivity）

GBase 8s 默认大小写不敏感，但在引号中的内容则是大小写敏感的； MySQL（InnoDB）在 Linux

环境下默认是大小写敏感的，在 Windows 环境下默认是大小写不敏感。

例如： GBase 8s 中 gbase == GBASE == GBase=="gbase" != "GBASE"，而在 MySQL 中则是完全不同。

所以在使用 mysqldump 或者其它工具导出建表语句中，带双引号的表名和字段名，都要加以处理才可以被 GBase 8s 使用。

GBase 8s 的建表语句在主体结构上是与 MySQL 相同的，但在一些特性上有不同的地方。

## 2.9 用户（User）

MySQL 需要创建数据库用户来进行数据库权限的管理。而 GBase 8s 则可以使用操作系统用户来进行同等的权限管理，可以使用 grant、revoke 命令对用户进行权限的授予与回收。不同数据库主机间的数据库访问，如同义词、跨实例视图等，需要在两台服务器上配置相应用户的信任关系，具体配置方法与使用的平台相关。

gbasedbt 用户是 GBase 8s 的超级管理员。

GBase 8s 同时允许用户通过内部授权服务（比如 Kerberos 或者 Microsoft Active Directory）来连接数据库而无需系统用户。这种方法使系统可以通过映射系统中已存在的除 gbasedbt 和 root 外的其他用户，来访问数据库。这种机制将会减少为了使用数据库而创建的大量系统用户。减少 DBA 的工作量，增加系统的安全系数。

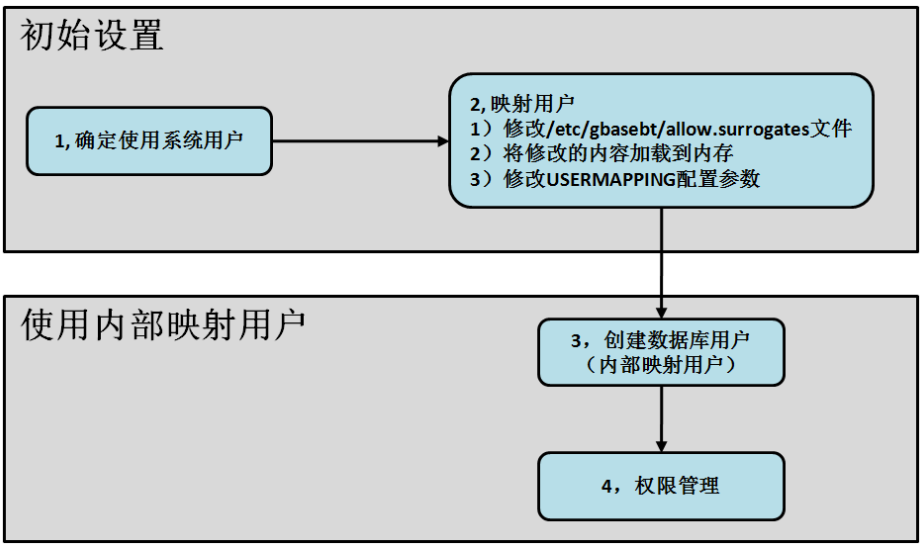


图 2.9.1 非系统用户访问数据库设置方法示意图

非系统用户（内部映射用户）访问数据库设置示例：

```
# 编辑 /etc/gbasedbt/allowed.surrogates 文件:
USER:daemon
# 加载到内存，修改配置文件
onmode -cache surrogates
onmode -wf USERMAPPING='ADMIN'
# 设置默认用户，指定默认目录（默认目录的权限是 777），用户密码需满足不少于 8 位含大小写字母及数字
dbaccess sysuser -
CREATE DEFAULT USER WITH PROPERTIES USER daemon HOME "/home/gbase/users" ;
CREATE USER sue WITH PASSWORD 'GBase123';
CREATE USER bale WITH PASSWORD 'GBase123';
# 授权管理
dbaccess testdb -
GRANT CONNECT TO sue;
GRANT CONNECT TO bale;
```

# 3. 数据库对象迁移

## 3.1 数据类型

### 3.1.1 数据类型对照表

**数值类型**

| GBase 8s | | MySQL |
|---|---|---|
| **SMALLINT** | | TINYINT / TINYINT UNSIGNED / SMALLINT |
| 2 字节 | -32,767 至 32,767 | |
| **INT / INTEGER** | | SMALLINT UNSIGNED / MEDIUMINT / MEDIUMINT UNSIGNED / INT / INTEGER |
| 4 字节 | -2,147,483,647 至 2,147,483,647 | |
| **BIGINT** | | INT UNSIGNED / BIGINT |
| 8 字节 | -9,223,372,036,854,775,807<br>至 9,223,372,036,854,775,807 | |
| **DECIMAL(p)** | | BIGINT UNSIGNED / DOUBLE UNSIGNED |
| 可变 | 最多 32 位有效位的浮点精度 | |
| **SMALLFLOAT / REAL** | | FLOAT |
| 4 字节 | 6 至 7 位有效数字，能保证 6 位 | |
| **FLOAT / DOUBLE PRECISION** | | FLOAT UNSIGNED / DOUBLE |
| 8 字节 | 14 至 15 位有效数字，能保证 14 位 | |
| **DECIMAL(p,s) / NUMERIC(p,s)** | | DECIMAL(p,s) / NUMERIC(p,s)<br>其中 p<=32, s<=20 |
| 可变 | 在小数部分中最多 20 位有效数字，<br>或在小数点的左边最多 32 位有效数字 | |

注：DECIMAL(p,s)超过精度的将无法迁移到 GBase 8s 上。

**字符类型**

| GBase 8s | | MySQL |
|---|---|---|
| **CHAR(n)** | | CHAR(n) |
| n 字节 | n<=32,767，不指定 n 时为 1 | |
| **VARCHAR(n) / LVARCHAR(n)** | | VARCHAR(n) 其中 n<=65,535 / TINYTEXT / TEXT 长度小于 32765 |
| n 字节 | n<=32,765，不指定 n 时为 1 | |
| **TEXT** | | TEXT / LONGTEXT / MEDIUMTEXT / LONGTEXT 长度小于 2GB / VARCHAR(n) |
| 可变 | 最大 2GB | |
| **CLOB** | | LONGTEXT |
| 可变 | 最大 4TB | |
| **BYTE** | | TINYBLOB / BLOB / MEDIUMBLOB / LONGBLOB 长度小于 2GB |
| 可变 | 最大 2GB | |
| **BLOB** | | LONGBLOB |
| 可变 | 最大 4TB | |

| | | |
|---|---|---|
| | | |

需要注意的是 text 或者 clob 数据类型有操作上有较大的限制，比如：不能将 text/clob 数据类型与其它字符类型进行比较操作；不能对 text/clob 进行 like 操作；无法直接通过 SQL 语句 insert text/clob 字段，但可通过扩展函数对 clob 数据类型进行操作，如：to_clob，clob_to_char 等。

**时间日期类型**

| GBase 8s | | MySQL |
|---|---|---|
| DATE | | DATE |
| 4 字节 | 0001-01-01 至 9999-12-31 | |
| **DATETIME HOUR TO SECOND** | | TIME |
| 4 字节 | 00:00:00 至 23:59:59 | |
| **DATETIME YEAR TO YEAR / CHAR(4)** | | YEAR |
| | 0001 至 9999 | |
| **DATETIME YEAR TO SECOND** | | DATETIME<br>TIMESTAMP |
| 8 字节 | 0001-01-01 00:00:00 至<br>9999-12-31 23:59:59 | |

## 3.1.2 数据类型说明

**数值类型：**

SMALLINT 类型包含 MySQL 的 TINYINT、TINYINT UNSIGNED 或 SMALLINT 数据类型值范围，SMALLINT 最小值-32,768 除外。

INTEGER 或 INT 类型包含 MySQL 的 SMALLINT UNSIGNED、MEDIUMINT、MEDIUMINT UNSIGNED、INT 或 INTEGER 数据类型值范围，INTEGER 最小值-2,147,483,648 除外。

BIGINT 类型包含 MySQL 的 INT UNSIGNED 或 BIGINT 数据类型值范围，BIGINT 最小值 -9,223,372,036,854,775,808 除外。

DECIMAL(p)类型包含 MySQL 的 BIGINT UNSIGNED 或 DOUBLE UNSIGNED 数据类型值范围，但需要注意的是 DECIMAL(p)最多只能保留 32 位有效精度。

SMALLFLOAT 或 REAL 类型包含 MySQL 的 FLOAT 数据类型值范围。

FLOAT 或 DOUBLE PRECISION 类型包含 MySQL 的 FLOAT UNSIGNED 或 DOUBLE 数据类型值范围，但需要注意边界数据值。

DECIMAL(p,s)或 NUMERIC(p,s)类型映射 MySQL 的 DECIMAL(p,s)数据类型，但需要注意的是，只能接受在小数部分中最多 20 位有效数字或在小数点的左边最多 32 位有效数字的值。

**字符类型：**

CHAR(n)类型包含 MySQL 的 CHAR(n)数据类型值范围。

VARCHAR(n)包含 MySQL 的 TINYTEXT、字符数对应字节小于 32765 的 VARCHAR(n) 或 长度小于 32765 的 TEXT 数据类型值范围。

TEXT 类型包含 MySQL 的 TEXT、LONGTEXT、MEDIUMTEXT、长度小于 2GB 的 LONGTEXT 或 VARCHAR(n) 的数据类型值范围。

CLOB 类型包含 MySQL 的 LONGTEXT 数据类型值范围。

BYTE 类型包含 MySQL 的 TINYBLOB、BLOB、MEDIUMBLOB 或 长度小于 2GB 的 LONGBLOB 的数据类型值范围。

BLOB 类型包含 MySQL 的 LONGBLOB 数据类型值范围。

**时间日期类型：**

DATE 类型包含 MySQL 的 DATE 数据类型值范围，0000 年除外。

DATETIME HOUR TO SECOND 类型映射 MySQL 的 TIME 数据类型值范围，但仅限 24 小时内的值。

DATETIME YEAR TO YEAR 或 CHAR(4)类型包含 MySQL 的 YEAR 数据类型值范围。

DATETIME YEAR TO SECOND 类型包含 MySQL 的 DATETIME 或 TIMESTAMP 数据类型值范围，不允许实际不存在的日期时间值（如 0000-00-00 00:00:00）。

## 3.2 表

### 3.2.1 建表语句

GBase 8s 中建表语句与 MySQL 是相似的，只有一些细节需要修改。

**语法**

```
CREATE [TEMP] TABLE table-name
        (
        column-name
            {
            datatype
            | {BYTE|TEXT} [IN {TABLE | BLOBspace-name]}]
            }
            [DEFAULT default_opts]
            [table-constraint-definition][,...]
            [column-constraint-definition][,...]
            [NOT NULL] [UNIQUE [CONSTRAINT constr-name]][,...]
            [UNIQUE (unique-column-list) [CONSTRAINT constr-name]][,...]
            [[COLUMN] SECURED WITH label-name]
        )
        [SECURITY POLICY policy-name]
        [WITH NO LOG] [IN DBspace-name] [EXTENT SIZE extent-size]
        [NEXT SIZE next-size]   [LOCK MODE {PAGE | ROW }];
```

在语句中指定 DEFAULT 值时，MySQL 的字符型字段可以不加引号，但 GBase 8s 必须用引号标注。

| GBase 8s | MySQL |
|---|---|
| CHAR(1) default **'0'** | CHAR(1) default 0 |

在语句中指定 COMMENT 时，MySQL 的注释可以直接写在字段或者表后面，但 GBase 8s 需要独立指定注释。

示例：

| GBase 8s | MySQL |
|---|---|
| create table tabc (<br>  id int,<br>  name varchar(255)<br>);<br>**comment on table tabc is '表 C';**<br>**comment on column tabc.id is '自增 ID';** | create table tabc (<br>  id int comment '自增 ID',<br>  name varchar(255)<br>) comment='表 C'; |

**引擎、字符集选项**

MySQL 中的引擎、字符集选项是 GBase 8s 没有的，需要去除；同样的标识符也要去除。

示例：

| GBase 8s | MySQL |
|---|---|
| create table tabc (<br>  id **serial** not null,<br>  name varchar(255) default null,<br>  primary key(id) | CREATE TABLE `tabc` (<br>  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '自增 ID',<br>  `name` varchar(255) DEFAULT NULL, |

| | |
|---|---|
| );<br>**comment on table tabc is '表 C';**<br>**comment on column tabc.id is '自增 ID';** | PRIMARY KEY (`id`)<br>) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |

使用 SERIAL 或 BIGSERIAL 数据类型替换 AUTO_INCREMENT 子句，移除存储元素子句（例如：TYPE 和 ENGINE，DEFAULT CHARSET）。

**临时表创建的区别：**

示例：

| GBase 8s | MySQL |
|---|---|
| CREATE TEMP TABLE temp_sales(<br>col1 int, col2 int<br>) ;<br>INSERT INTO temp_sales SELECT * FROM sales; | CREATE TEMPORARY TABLE temp_sales AS<br>SELECT * FROM sales; |

GBase 8s 的临时表不支持 create temp table TEMPTABLE as SELECT … 语法，需要逐步创建。

### 3.2.2 修改表语句

ALTER TABLE 语句用于更改一个或多个表属性。MySQL 和 GBase 8s 的 ALTER TABLE 语句非常类似，但还是有一定区别。MySQL 语句要经过修改才能在 GBase 8s 上成功运行。

区别比较如下：

| 修改表操作 | GBase 8s | MySQL |
|---|---|---|
| 新增列 | ALTER TABLE t1<br>ADD (c3 INT); | ALTER TABLE t1<br>ADD COLUMN c3 INT; |
| 修改现有列类型 | ALTER TABLE t1<br>MODIFY (c2 VARCHAR(20)); | ALTER TABLE t1<br>MODIFY c2 VARCHAR(20); |
| 删除列 | ALTER TABLE t1<br>DROP (c2); | ALTER TABLE t1<br>DROP COLUMN c2; |
| 将一列的下一个序列值更改为 1000 | ALTER TABLE t1<br>MODIFY (c4 SERIAL(1000)); | ALTER TABLE t1<br>AUTO_INCREMENT=1000; |
| 删除约束 | ALTER TABLE t1<br>DROP CONSTRAINT fk_symbol; | ALTER TABLE t1<br>DROP FOREIGN KEY fk_symbol; |
| 修改列名 | REANME COLUMN t1.c3 TO col3; | ALTER TABLE t1<br>CHANGE c3 col3 INT; |

# 3.3 索引

## 3.3.1 语法

GBase 8s 中建表索引与 MySQL 基本上是一致的。

```
CREATE [UNIQUE|DISTINCT] [CLUSTER] INDEX index-name
ON table-name (column-name [ASC|DESC],...) storage option [online]
```

语法示例：

| GBase 8s | MySQL |
|---|---|
| create index zip_ix on customer (zipcode) ; | create index zip_ix on customer (zipcode) ; |

在线创建索引：

```
create index idx_name on table_name(col_name1,colname2,...) ONLINE;
```

在创建索引时，使用 online 关键字，可以在用户访问这张表的同时，进行创建索引操作，而不会造成锁表。online 关键字在删除索引时同样有效。

## 3.3.2 索引限制

GBase 8s 的索引长度限制于使用的数据页大小相关

| 常用页大小 | 索引长度限制 |
|---|---|
| 2K | 387 字节 |
| 4K | 796 字节 |
| 8K | 1615 字节 |
| 12K | 2435 字节 |
| 16K | 3254 字节 |

每个索引分片的最大页数 2,147,483,647。

## 3.3.3 复合索引

GBase 8s 中一个复合索引最多可以使用 16 个字段做键值，或最多 341 个键值作为 SPL/JAVA UDR，102 个键值作为 C UDR 的返回值。这在不同的语言中有不同的上限限制。

### 3.3.4 其它

GBase 8s 可以根据应用需要，用 CLUSTER 选项来建立聚集索引，这个选项会对表中的数据按索引的顺序排序。每个表只能建立一个聚集索引。DML 语句的数量是判断能否可以建立聚集索引的重要因素，聚集索引上的大量的 DML 语句会导致性能问题。

## 3.4 约束

### 3.4.1 创建线束

语法

```
CREATE [TEMP] TABLE table-name
        (
        column-name
            {   datatype   }
            [table-constraint-definition][,...]
            [column-constraint-definition][,...]
            [NOT NULL] [UNIQUE [CONSTRAINT constr-name]][,...]
            [UNIQUE (unique-column-list) [CONSTRAINT constr-name]][,...]
        )
        [WITH NO LOG][IN DBspace-name] [EXTENT SIZE extent-size]
        [NEXT SIZE next-size] [LOCK MODE {PAGE | ROW }]

ALTER TABLE table-name ADD CONSTRAINT UNIQUE (old-column-name[,...])[table-constraint-definition]
```

GBase 8s 建立约束的语法与 MySQL 有所不同，主要区别在于 GBase 8s 的约束名称子句在后，而 MySQL 的在前。

| 操作 | GBase 8s | MySQL |
|---|---|---|
| 主键 | PRIMARY KEY(cust_num)<br>CONSTRAINT pk_number | CONSTRAINT pk_number<br>PRIMARY KEY(cust_num) |
| 检查 | CHECK (emp_code > 100)<br>CONSTRAINT ck_empcd | CONSTRAINT ck_empcd<br>CHECK(emp_code > 100) |
| 唯一约束 | UNIQUE(emp_no)<br>CONSTRAINT uq_empno | CONSTRAINT uq_empno<br>UNIQUE(emp_no) |
| 外键参考 | FOREIGN KEY(classes_id)<br>REFERENCES CLASSES(id)<br>CONSTRAINT fk_classes_id | CONSTRAINT fk_classes_id<br>FOREIGN KEY(classes_id)<br>REFERENCES CLASSES(id) |

### 3.4.2 删除约束

GBase 8s 修改约束的语法与 MySQL 有所不同。GBase 8s 总是以 DROP CONSTRAINT 子句删除约束。

| 操作 | GBase 8s | MySQL |
|---|---|---|
| 删除主键 | ALTER TABLE tab1 DROP CONSTRAINT pk_number; | ALTER TABLE tab1 DROP PRIMARY KEY; |
| 删除检查 | ALTER TABLE tab1 DROP CONSTRAINT ck_empcd; | ALTER TABLE tab1 DROP CHECK ck_empcd; |
| 删除约束 | ALTER TABLE tab1 DROP CONSTRAINT uq_empno; | ALTER TABLE tab1 DROP [INDEX \| KEY] uq_empno; |
| 删除外键 | ALTER TABLE tab1 DROP CONSTRAINT fk_classes_id; | ALTER TABLE tab1 DROP FOREIGEN KEY fk_classes_id; |

## 3.5 视图

创建视图语法：

```
CREATE VIEW view-name [(column-list)] AS SELECT statement [WITH CHECK OPTION]
```

创建视图示例：

```
CREATE VIEW v_dept (deptno, dname) AS
SELECT x0.deptno ,x0.dname FROM dept x0 ;
```

GBase 8s 建立视图的语句与 GBase 8s 基本相同。

由于视图并不是一张实体表，不能对其使用 ALTER 操作，如果有视图名或字段需要修改，需要将其重建。

GBase 8s 当前版本暂时还不支持物化视图。

## 3.6 触发器

MySQL 允许 SQL 语句、逻辑语句和存储过程在触发器中调用。GBase 8s 只允许 SQL 语句和存储过程在触发器内调用。所以：GBase 8s 触发器包含的逻辑语句必须转换为 GBase 8s 的存储过程，然后存储过程由触发执行，即触发存储过程。

GBase 8s 创建触发器常用语法

INSERT 触发器的语法：

```
CREATE TRIGGER trigger_name INSERT ON table_name
BEFORE [WHEN (condition)] (trig_action1,trig_action2,...)|
FOR EACH ROW [WHEN (condition)] (trig_action1,trig_action2,...)|
```

```
AFTER [WHEN (condition)] (trig_action1,trig_action2,...)|
[DISABLED|ENABLED]
```

DELET 触发器的语法：

```
CREATE TRIGGER trigger_name DELETE ON table_name
BEFORE [WHEN (condition)] (trig_action1,trig_action2,...)|
FOR EACH ROW [WHEN (condition)] (trig_action1,trig_action2,...)|
AFTER [WHEN (condition)] (trig_action1,trig_action2,...)|
 [DISABLED|ENABLED]
```

UPDATE 触发器的语法:

```
CREATE TRIGGER trigger_name UPDATE [OF (column,column,...)] ON table_name
BEFORE [WHEN (condition)] (trig_action1,trig_action2,...)|
FOR EACH ROW [WHEN (condition)] (trig_action1,trig_action2,...)|
AFTER [WHEN (condition)] (trig_action1,trig_action2,...)|
[DISABLED|ENABLED]
```

触发存储过程示例：

| GBase 8s | MySQL |
|---|---|
| -- 触发存储过程的过程部分<br>CREATE PROCEDURE proc_tri_sumofsalary()<br>REFERENCING NEW AS NEW FOR tb_emp8<br>  LET NEW.salary = 0;<br>END PROCEDURE;<br>-- 触发存储过程的触发器部分<br>CREATE TRIGER tri_sumofsalary INSERT ON tb_emp8<br>FOR EACH ROW(<br>  EXECUTE PROCEDURE proc_tri_sumofsalary() WITH<br>TRIGGER REFERENCES<br>); | CREATE TRIGGER tri_sumofsalary<br>BEFORE INSERT ON tb_emp8<br>FOR EACH ROW<br>SET NEW.salary = 0; |

## 3.7 序列

MySQL 中是没有序列的，但是 MySQL 有提供了自增长（AUTO_INCREMENT）来实现类似的目的，但也只是自增，而不能设置步长、开始索引、是否循环等。GBase 8s 可以直接使用 SERIAL 或 BIGSERIAL 或者 SERIAL8 字段代替，GBase 8s 的序列最大值为 bigint/int8 类型的最大值，即 9,223,372,036,854,775,807。

GBase 8s 支持 DML 语句(CREATE SEQUENCE, ALTER SEQUENCE, RENAME SEQUENCE, DROP SEQUENCE)来操纵序列，多用户可以并发访问同一个序列，每个序列占用 8 个字节，GRANT 和 REVOKE 语句可以用来更改序列的访问权限。可以为本地数据库中的序列建立同义词。用于调整序列值的 CURRVAL

和 NEXTVAL 在同义词中同样适用。

## 3.8 导出数据库结构

从 MySQL 数据库抽取表结构，可以使用 MySQL 自带的工具 mysqldump。

示例：

```
mysqldump -uUSERNAME -p DATABASENAME \
--events --routines --skip-add-drop-table \
--no-data --skip-comments > DATABASENAME.sql
```

生成的 SQL 文件是类似于下面这样的

```
/*!40101 SET character_set_client = @saved_cs_client */;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `Tab1` (
  `col1` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

在上文中提到过，GBase 8s 与 MySQL 语法上的差异，所以这里需要使用文本处理工具将这些 SQL 语句按照 GBase 8s 支持的格式进行修改。

GBase 8s 提供了非常简易的工具来完成此类操作。只需要执行命令

```
dbschema -d DATABASENAME -ss DATABASENAME.sql
```

即可完成导出。

## 3.9 导入数据库结构

**准备工作：**

1. 导入数据库结构前，必须按照 GBase 8s 语法格式修改创建对象脚本。
2. 创建数据库 database_name

**导入数据库结构：**

执行命令 dbaccess -e -m -a database_name database_name.sql 来将修改好的脚本文件执行，将数据库对象建立。

为了加快数据导入，建议索引和主外键约束等在这里先不要建立，等待数据全部导入后，再开 PDQ 一并建立。

# 4. 数据迁移

## 4.1 迁移过程示意

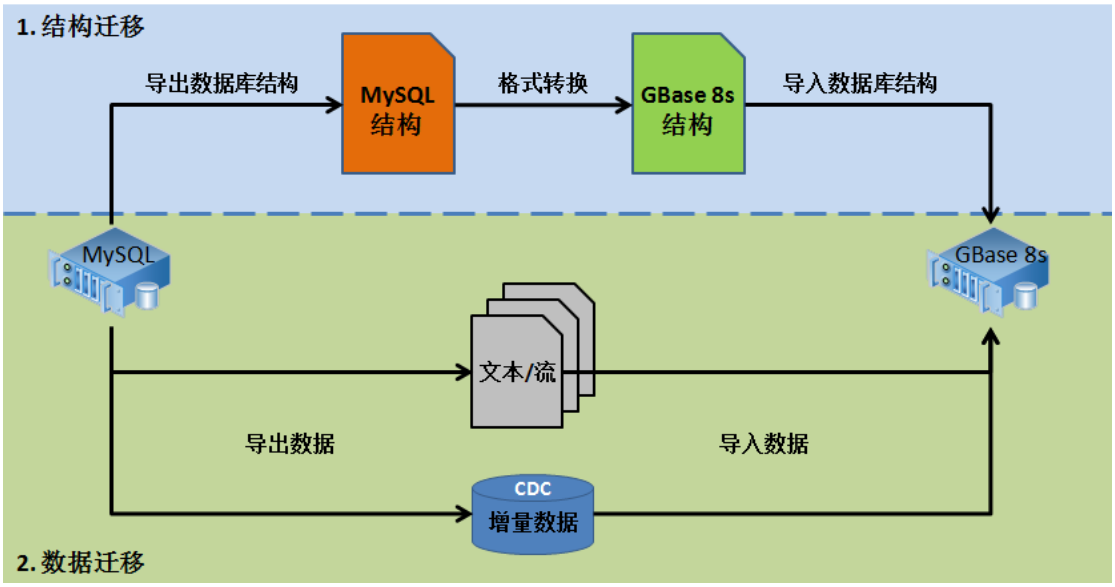


图 4.1.1 迁移示意图

如图 4.1.1，数据的迁移过程由两部分组成：

第一部分是迁移数据库的结构，主要包括数据库对象的迁移；

第二部分是将数据迁移到目标数据库中。

其中，第一部分我们在上一章节的 3.8、3.9 已经详细讲解。

真正的数据迁移是在第二部分完成的。对于数据的迁移，通常情况下是要暂停应用程序，这样可以保证迁移前后的数据一致性和完整性。使用常规方法，如果数据量很大的话，则需要较长的应用停机时间窗口。

在对时间窗口要求比较小的应用做迁移时，可以使用第三方工具(如 CDC)来进行增量数据的迁移，这样可以最大限度的减少停机时间，甚至可以实现 MySQL 到 GBase 8s 无缝迁移。

数据库结构和初始数据可以使用 GMTK（GBaseMigrationToolkit）工具来完成，工具可以按预设置的规划进行数据迁移，但仍需人工确认迁移后的数据库结构合理性。

## 4.2 文件格式

把数据从 MySQL 导出时，要将格式尽量调整为匹配 GBase 8s 的导入格式，这样可以最大程度上减小

数据在转换过程中的工作量。这里以 GBase 8s 的 load 工具格式为例，来说明 一下导入时对文件格式的要求。

GBase 8s 的 load 工具是一个常用的文本导入工具，它的字段间默认分隔符是“|”管道符。由于“|”在文本中是很少出现的字符，所以也推荐在导出时使用“|”来作为分隔符。每个字段后都应由“|”来作为结束标识，换行符作为行与行之间的分隔符。每个表的数据单独存储在一个文件中。

示例：

```
create table customer_log
  (   id char(14),
      update_date datetime year to second,
      tablename varchar(20),
      update_count float,
updated float    );
```

表 customer_log 的导出/导入格式如下：

```
20101013114153|2010-10-13 11:41:53|2|53.0|53.0|
20101013114153|2010-10-13 11:41:53|3|0.0|0.0|
20101015094917|2010-10-15 09:49:17|2|15.0|15.0|
20101015094917|2010-10-15 09:49:17|3|0.0|0.0|
20101015094918|2010-10-15 09:49:18|4|1.0|1.0|
20101015102622|2010-10-15 10:26:22|2|2.0|2.0|
20101015102622|2010-10-15 10:26:22|4|0.0|0.0|
20101015111103|2010-10-15 11:11:03|1|1.0|1.0|
```

## 4.3 数据导出

从MySQL提取数据库架构后，下一步需要考虑从MySQL中的每个表提取数据。为此，可以使用SELECT语句和 mysqldump 或 OUTFILE 选项。无论采用哪种数据提取方法，要在 MySQL 和 GBase 8s 之间实现平滑数据移动，必须要考虑以下事项：

1)、MySQL 中的卸载数据是文本格式。

2)、为 MySQL 中的每个表创建一个独立卸载文件(outfile)

3)、卸载文件中的每一行表示 MySQL 中的一行。

4)、MySQL 中的每一列/字段通过一个字符|分隔。

5)、MySQL 中的每一行通过“/n”字符或其他有效的行结束符终止。

可以使用类似 SQL 命令从 MySQL 表提取数据。以文本格式从 MySQL 表卸载数据：

```
SELECT * INTO OUTFILE '/tmp/tab1.unl'
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '|\n'
FROM tab1;
```

可以编写脚本来并发执行以上操作来提升数据从 MySQL 导出的效率。

导出的数据以 GBase 8s 导入要求的格式存储在/tmp/tab1.unl 中。

# 4.4 数据导入

## 4.4.1 注意事项

建立数据库时，以无日志模式建立，这样数据在导入时无需记录逻辑日志，导入效率会大幅提高。当数据导入完毕后，使用 ontape 命令将数据库改为需要的日志模式。

例如将某数据库日志模式改为 UNBUFF：

```
ontape -u DATABASE -s -L 0 -t /dev/null
```

在数据量大的迁移中，可以将建立索引、约束等脚本单独提取出来，在所有数据成功导入后，再打开 PDQ 建立。

## 4.4.2 导入工具 load

使用 load 工具进行导入：

load 是 GBase 8s 最基础和最常用的文本数据导入工具，支持多表并发导入，操作简单。

示例：

```
Load from /datam/tab1.unl insert into tab1;
```

## 4.4.3 外部表导入

使用 GBase 8s 外部表(external table)进行导入：

对于数据量大的表，传统的 load 导入方式会在迁移过程中占用大量的时间窗口，成为迁移效率的瓶颈。针对这个问题，对大表的导入，可以采用 GBase 8s 外部表的方式进行，

创建 External table：

语法

```
CREATE EXTERNAL TABLE table-name
(
column-name { datatype [DEFAULT default_opts] | <UDTs> } [
<external-column-defn> ] [,...]
)
USING   (DATAFILES("{DISK | PIPE} : file-path" [,...] )
[, <table-option> [...] ])
<external-column-defn>: EXTERNAL CHAR( size ) [ NULL 'null-string' [ NOT NULL ] ]
<table-options>:
  FORMAT format-type
  DEFAULT | DELUXE | EXPRESS
  ESCAPE 'escape-character'
  DELIMITER 'field-delimiter'
  RECORDEND 'record-delimiter'
  MAXERRORS num-errors
  REJECTFILE 'filename'
  NUMROWS num-rows
```

示例：

```
  create external table orders_ext
    ( order_num serial, order_date date, customer_num integer,ship_instruct char(40),
      backlog char(1), po_num char(10),ship_date date, ship_weight decimal(8,2),
      ship_charge money(6,2), paid_date date )
  using
    (
      datafiles   ("DISK:/opt/gbase/test/external_table/orders1.unl",
                   "DISK:/opt/gbase/test/external_table/orders2.unl" ),
      format "delimited",
      DELIMITER "|",
      rejectfile "/opt/gbase/test/external_table/orders_rejfile.err",
      maxerrors 100
    );
```

也可根据已有表结构建立相同结构的外部表：

```
  create external table orders_ext SAMEAS orders
  using
    (
        datafiles   ("DISK:/opt/gbse/test/external_table/orders1.unl",
                     "DISK:/opt/gbase/test/external_table/orders2.unl" ),
      format "delimited",
      DELIMITER "|",
      rejectfile "/opt/gbase/test/external_table/orders_rejfile.err",
      maxerrors 100
```

```
);
```

导入数据：

```
insert into orders select * from orders_ext;
```

外部表使用技巧：PDQ & 分片表 & Light append

打开 PDQ 功能，并行处理。

目的表是分片表能进行并行的 insert 和 select。

当导入表为 RAW TABLE 时，利用 Light append 进行快速数据导入。

## 4.5 校验

对数据行数进行校验

MySQL 数据库中各表的行数统计：

```
use information_schema;
select table_name,table_rows from tables
where TABLE_SCHEMA = 'DATABASENAME' order by table_rows desc;
```

GBase 8s 数据库中各表的行数统计：

```
update statistics;
select tabname,nrows from systables where tabid>99 and tabtpye='T' order by 2 desc;
```

# 5. 应用迁移

## 5.1 SQL

### 5.1.1 关键字

与 DDL 语句类似，GBase 8s 的 DML 语句与 MySQL 在绝大多数结构上是一致的。一些细节用法上的不同和对应功能的语法差异，是需要进行转换的。例如 UNITS,YEAR, MONTH, DAY, HOUR 这些 GBase 8s 中的关键字，是必须进行转换的。

**更多的关键字请参考附录 A。**

### 5.1.2 查询 SELECT

查询语句在被执行时，由数据库优化器来决定最佳的查询计划。GBase 8s 与 MySQL 使用着各自开发的不同的查询优化器，有着各自的方法和规则。同一个查询语句在不同的优化器下可能会选择不同的查询路径，语句的执行时间也会有快有慢。因此，迁移后的每个查询语句的性能都要经过测试，来确定是否需要进行必要的转换来达到预期的性能。

在一些框架或者连接池中，需要指定检测连接的语句 (如 druid 中的 validationQuery)，默认是 select 1，而 GBase 8s 需要完整的 SQL 语句：select 1 from dual; 。

### 5.1.3 合并 GROUP BY 子句

GBase 8s 中的 GROUP BY 子句限制较严格，非聚集函数字段均需要显式的在 GROUP BY 的列表中。同时 GROUP BY 列表中只能是字段名称或者是投影列序列，而不能是函数。

GROUP BY 列表示例：

<table>
<tr><th>GBase 8s</th><th>MySQL</th></tr>
<tr><td>select id,min(age),substr(name,1,4) myname<br>from tab1<br>group by id, myname;</td><td rowspan="2">select id,min(age),substring(name,1,4) myname<br>from tab1<br>group by id;</td></tr>
<tr><td>select id,min(age),min(subst(name,1,4))<br>from tab1<br>group by id;</td></tr>
</table>

投影列示例：

| GBase 8s | MySQL |
| --- | --- |
| select id,min(age),substr(name,1,4) myname<br>from tab1<br>group by **1,3**; | select id,min(age),substring(name,1,4) myname<br>from tab1<br>group by id,substring(name,1,4); |
| select id,min(age),subst(name,1,4) myname<br>from tab1<br>group by id,**myname**; | |

### 5.1.4 排序 ORDER BY 子句

GBase 8s 中的排序 ORDER BY 子句限制较严格，仅允许符合要求的排序规则。例如：MySQL 允许使用 字符类型 + 0 这样的组合形成的数值排序，而在 GBase 8s 中将会报错，如果需要相同排序的结果集，需要进行改造。

ORDER BY 排序示例：

| GBase 8s | MySQL |
| --- | --- |
| select id,name,sex,age<br>from tab3<br>order by **str2num(name)** + 0 asc; | select id,name,sex,age<br>from tab3<br>order by name + 0 asc; |

其中 str2num 自定义函数的内容如下：

```
-- str2num
drop function if exists str2num;
create function str2num(p1 varchar(255))
returns bigint;
  on exception
    return 0;
  end exception;
  return p1 + 0;
end function;
```

### 5.1.5 查询优化器（伪指令）

GBase 8s 支持在查询语句中指定它的执行计划，指定的格式有多种，例如：

```
select {+ ORDERED AVOID_FULL(e)}
empname, deptno
from employee e, department d
where e.dept_no = d.dept_no;
```

通过对 SELECT 后的注释部分加入“+”来植入指定的执行计划。

### 5.1.6 多值 INSERT

MySQL 中的 INSERT INTO tabname VALUES(...),(...)；多值插入在 GBase 8s 中是不支持的。

### 5.1.7 临时表

GBase 8s 中将临时表几乎当做一张常规表使用，当一个会话结束时，它遗留的临时表也自动被删除。并且不持 MySQL 中的全局临时表。

GBase 8s 不允许通过 CREATE TEMP TABLE ... AS SELECT ...的方式创建临时表，而 MySQL 允许使用该方式。

GBase 8s 中允许不同会话中创建同名临时表，而这在 MySQL 中是不允许的，任何时候同一用户的表名都唯一。

GBase 8s 中可以采用如下两种方式创建临时表：

| |
|---|
| 使用 SELECT INTO TEMP 语句隐式的创建临时表； |
| 例：select * from customer into temp tmp_customer with no log; |
| 使用 CREATE TEMP TABLE 语句显式的创建临时表； |
| 例：create temp table tmp_customer (c1 int) with no log; |

### 5.1.8 系统表

涉及到 MySQL 系统表的应用，必须进行移植，用 GBase 8s 对应的系统表将其替换，对于数据库系统表中无法对应的部分，需要对这部分程序进行删除。

GBase 8s 的绝大多数系统表名都以 sys 开头，若 MySQL 应用中的表与 GBase 8s 系统表重名，则必须在迁移时更换表名。

DUAL（空表）：GBase 8s 中的 DUAL 是与 MySQL 的 DUAL 表作用相同的类表结构。

## 5.2 运算符

运算符大多是通用的，存在部分操作符使用不同的写法。

### 5.2.1 算术运算符

| GBase 8s | MySQL | 差异描述 |
|---|---|---|
| + | + | 相同 |
| - | - | 相同 |
| * | * | 相同 |
| / | / 或者 DIV | / 相同，不支持 a DIV b |
| MOD(a, b) | % 或者 MOD 或者 MOD(a, b) | 只支持 MOD(a, b)函数，不支持 a % b 和 a MOD b |

差异在于除法和取余数两个运算符上，MySQL 支持更多写法。

### 5.2.2 比较运算符

| GBase 8s | MySQL | 差异描述 |
|---|---|---|
| = | = | 相同 |
| != 或者 <> | != 或者 <> | 相同 |
| > | > | 相同 |
| < | < | 相同 |
| <= | <= | 相同 |
| >= | >= | 相同 |
| BETWEEN a AND b | BETWEEN a AND b | 相同 |
| NOT BETWEEN a AND b | NOT BETWEEN a AND b | 相同 |
| IN | IN | 相同 |
| NOT IN | NOT IN | 相同 |
| | <=> | 不支持，需要修改 |
| LIKE | LIKE | 相同 |
| NOT LIKE | NOT LIKE | 相同 |
| REGEXP_LIKE(a, b) | a REGEXP b 或 a RLIKE b | 不相同，只支持 REGEXP_LIKE(a, b)函数 |
| IS NULL | IS NULL | 相同 |
| IS NOT NULL | IS NOT NULL | 相同 |

差异在于不支持 MySQL 的安全等于(<=>)；REGEXP 或者 RLIKE 操作符，需要替换为 REGEXP_LIKE 函数。

注：regexp_like 函数在某些 GBase 8s 版本中，需要手工创建，如下：

```
create dba function gbasedbt.regexp_like(varchar(255),
                                          varchar(255),
                                          varchar(255) default '')
returning boolean with (PARALLELIZABLE)
external name '$GBASEDBTDIR/lib/liboracle.udr(regexp_like)'
language C;
```

### 5.2.3 位运算符

| GBase 8s | MySQL | 差异描述 |
|---|---|---|
| bitand(a,b) | & | 不相同，使用函数操作 |
| bitor(a,b) | \| | 不相同，使用函数操作 |
| bitxor(a,b) | ^ | 不相同，使用函数操作 |
| bitnot(a) | ~ | 不相同，使用函数操作；特别注意 MySQL 使用无符号 int8 |
| | << | 不支持 |
| | >> | 不支持 |

位运算符均需要改成使用函数，且不支持左位移和右位移。取反操作，MySQL 使用无符号的 int8 类型，GBase 8s 使用 integer 类型。

## 5.3 函数

依据现有的函数，可以通过比较简单的组合，编写成 MySQL 兼容函数。具体示例参考：附录 B GBase 8s 自定义 MySQL 兼容函数。

### 5.3.1 字符类函数

一些 MySQL 特有的函数，GBase 8s 不支持。一些 MySQL 的函数与 GBase 8s 的名称不一样，或者参数不一样。

| GBase 8s | MySQL | 差异描述 |
|---|---|---|
| ASCII(s)<br>*Java UDR: JASCII(s)* | ASCII(s) | 对于不属于 ASCII 值的字符解析不一样<br>可使用 Java UDR 的 JASCII（s）替换 |
| CHR(n) | CHAR(n) | 函数名称不一样，可替代 |
| CHAR_LENGTH(s)<br>CHARACTER_LENGTH(s) | CHAR_LENGTH(s)<br>CHARACTER_LENGTH(s) | 相同 |
| CONCAT(s1,s2)<br>s1 \|\| s2 \|\| … \|\| sn<br>*UDR:CONCAT(s1,s2,…s16)* | CONCAT(s1,s2…sn) | GBase 8s 只支持两个字符串合并<br>多于两个字符串，请使用 \|\| 连接<br>也可使用 UDR 的 CONCAT(s1,s2,…s16)<br>自定义函数参考附录 B |
| *UDR:CONCAT_WS(x,s1,s2…s16)* | CONCAT_WS(x, s1,s2…sn) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| | FIELD(s,s1,s2…) | 不支持该函数 |
| *UDR: FIND_IN_SET(s1,s2)* | FIND_IN_SET(s1,s2) | 不支持该函数，可通过自定义函数实现 |

| | | |
|---|---|---|
| | | 也可使用 Java UDR 的 FIND_IN_SET(s1,s2) |
| *UDR:FORMAT(x,n,loc)* | FORMAT(x,n,loc) | 不支持该函数，可通过自定义函数实现部分功能<br>自定义函数参考附录 B |
| *UDR: INSERT(s1,x,len,s2)* | INSERT(s1,x,len,s2) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| CHARINDEX(s1,s)<br>INSTR(s,s1) | LOCATE(s1,s)<br>POSITION(s1 IN s) | 函数名称不一样，可替代 |
| LOWER(s) | LCASE(s)<br>LOWER(s) | 存在别名 |
| LEFT(s,n) | LEFT(s,n) | 相同 |
| LPAD(s1,len,s2) | LPAD(s1,len,s2) | 相同 |
| LTRIM(s) | LTRIM(s) | 相同 |
| SUBSTR(s,n,len) | MID(s,n,len)<br>SUBSTRING(s,n,len)<br>SUBSTR(s, n, length) | 存在别名 |
| *UDR: REPEAT(s,n)* | REPEAT(s,n) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B<br>也可以使用 Java UDR 的 REPEAT(s,n) |
| REPLACE(s,s1,s2) | REPLACE(s,s1,s2) | 相同 |
| REVERSE(s) | REVERSE(s) | 相同 |
| RIGHT(s,n) | RIGHT(s,n) | 相同 |
| RPAD(s1,len,s2) | RPAD(s1,len,s2) | 相同 |
| RTRIM(s) | RTRIM(s) | 相同 |
| SPACE(n) | SPACE(n) | 相同 |
| *UDR: STRCMP(s1,s2)* | STRCMP(s1,s2) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| SUBSTRING_INDEX(s, delimiter, number) | SUBSTRING_INDEX(s, delimiter, number) | 相同 |
| TRIM(s) | TRIM(s) | 相同 |
| UPPER(s) | UCASE(s)<br>UPPER(s) | 相同 |
| WM_CONCAT(s) | GROUP_CONCAT(s) | 函数名称不一样，可以创建别名 |

字符串函数函数中：

MySQL 的 ASCII(s)函数需要使用 Java UDR 中的 JASCII（s）函数来替代；

MySQL 的 CONCAT()函数支持多个参数直接合并，而 GBase 8s 的 CONCAT()函数仅支持两个参数，可使用 CONCAT(CONCAT(s1,s2),s3)这样多级嵌套的方式实现，或者使用另一种双竖线的连接方式：s1||s2||s3 来实现。

MySQL 的 FIND_IN_SET(s1,s2)、REPEAT(s,n)、STRCMP(s1,s2)等函数，可通过自定义函数(含 Java

自定义函数)的方式来实现。

### 5.3.2 数值函数

| GBase 8s | MySQL | 差异描述 |
|---|---|---|
| ABS(x) | ABS(x) | 相同 |
| ACOS(x) | ACOS(x) | 相同 |
| ASIN(x) | ASIN(x) | 相同 |
| ATAN(x) | ATAN(x) | 相同 |
| ATAN2(n, m) | ATAN2(n, m) | 相同 |
| AVG(expression) | AVG(expression) | 相同 |
| CEIL(x) | CEIL(x)<br>CEILING(x) | 相同 |
| COS(x) | COS(x) | 相同 |
| COT(x) | COT(x) | 相同 |
| COUNT(expression) | COUNT(expression) | 相同 |
| DEGREES(x) | DEGREES(x) | 相同 |
| n / m | n DIV m | 函数名不一样 |
| EXP(x) | EXP(x) | 相同 |
| FLOOR(x) | FLOOR(x) | 相同 |
| GREATEST(expr1, expr2, expr3, ...) | GREATEST(expr1, expr2, expr3, ...) | 相同 |
| LEAST(expr1, expr2, expr3, ...) | LEAST(expr1, expr2, expr3, ...) | 相同 |
| LN(x) | LN(x) | 相同 |
| LOGN(x) | LOG(x) 或 LOG(base, x) | 函数名不一样 |
| LOG10(x) | LOG10(x) | 相同 |
| *UDR: LOG2(x)* | LOG2(x) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| MAX(expression) | MAX(expression) | 相同 |
| MIN(expression) | MIN(expression) | 相同 |
| MOD(x,y) | MOD(x,y) | 相同 |
| *UDR: PI()* | PI() | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| POW(x,y) | POW(x,y)<br>POWER(x,y) | 相同 |
| RADIANS(x) | RADIANS(x) | 相同 |
| *Java UDR: RANDOM()* | RAND() | 不支持该函数，可通过自定义函数实现<br>可以使用 Java UDR 的 RANDOM() |
| ROUND(x,y) | ROUND(x,y) | 相同 |
| SIGN(x) | SIGN(x) | 相同 |
| SIN(x) | SIN(x) | 相同 |
| SQRT(x) | SQRT(x) | 相同 |

| | | |
|---|---|---|
| SUM(expression) | SUM(expression) | 相同 |
| TAN(x) | TAN(x) | 相同 |
| TRUNC(x,y) | TRUNCATE(x,y) | 函数名不一样 |

数值函数相同的比较多，主要的差异是 LOG2()、PI()、RAND()函数不存在，可以使用自定义函数（包含 Java 自定义函数）来替代。

### 5.3.3 日期时间函数

MySQL 有较丰富的日期时间函数，GBase 8s 对应缺少很多，但大多数函数可以通过自定义函数的方式来实现。

| GBase 8s | MySQL | 差异描述 |
|---|---|---|
| d + n UNITS DAY<br>*UDR: ADDDATE(d,n)* | ADDDATE(d,n) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B，部分兼容 |
| t + INTERVAL(n) FIRST TO LAST<br>*UDR: ADDTIME(t,n)* | ADDTIME(t,n) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B，部分兼容 |
| TODAY<br>*UDR: CURDATE()*<br>*UDR: CURRENT_DATE()* | CURDATE()<br>CURRENT_DATE() | 函数或表达式名不同，可通过自定义函数实现<br>自定义函数参考附录 B |
| CURRENT HOUR TO SECOND<br>*UDR: CURTIME()*<br>*UDR: CURRENT_TIME()* | CURTIME()<br>CURRENT_TIME() | 函数或表达式名不同，可通过自定义函数实现<br>自定义函数参考附录 B |
| CURRENT YEAR TO SECOND<br>SYSDATE<br>*UDR: CURRENT_TIMESTAMP()*<br>*UDR: NOW()* | CURRENT_TIMESTAMP()<br>LOCALTIME()<br>LOCALTIMESTAMP()<br>NOW()<br>SYSDATE() | 函数或表达式名不同，可通过自定义函数实现<br>自定义函数参考附录 B |
| DATE(x) | DATE(x) | 相同 |
| *UDR: DATETIFF(d1,d2)* | DATEDIFF(d1,d2) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| *UDR: DATE_ADD(d,n)* | DATE_ADD(d，INTERVAL expr type) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| GBASE_TO_CHAR(d,fmt)<br>*UDR: DATE_FORMAT(d,fmt)* | DATE_FORMAT(d,f) | 函数名称不一样，参数不一样<br>自定义函数参考附录 B |
| *UDR: DATE_SUB(d,n)* | DATE_SUB(date,INTERVAL expr type) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| DAY(d)<br>*UDR:DAYOFMONTH(d)* | DAY(d)<br>DAYOFMONTH(d) | 相同 |
| GBASE_TO_CHAR(d, '%A')<br>*UDR: DAYNAME(d)* | DAYNAME(d) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |

| | | |
|---|---|---|
| *UDR:DAYOFWEEK(d)* | DAYOFWEEK(d) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| *UDR: DAYOFYEAR(d)* | DAYOFYEAR(d) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| EXTEND(d, type) | EXTRACT(type FROM d) | 函数名称不一样，参数不一样 |
| *UDR: FROM_DAYS(n)* | FROM_DAYS(n) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| GBASE_TO_CHAR(d, '%H')<br>*UDR: HOUR(t)* | HOUR(t) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| LAST_DAY(d) | LAST_DAY(d) | 相同 |
| *UDR: MAKEDATE(year, day-of-year)* | MAKEDATE(year, day-of-year) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| *UDR: MAKETIME(hh24, mi, ss)* | MAKETIME(hour, minute, second) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| *UDR: MICROSECOND(date)* | MICROSECOND(date) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| GBASE_TO_CHAR(d, '%M')<br>*UDR: MINUTE(t)* | MINUTE(t) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| GBASE_TO_CHAR(d, '%B')<br>*UDR: MONTHNAME(d)* | MONTHNAME(d) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| MONTH(d) | MONTH(d) | 相同 |
| *UDR: PERIOD_ADD(p, n)* | PERIOD_ADD(period, number) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| *UDR: PERIOD_DIFF(p1, p2)* | PERIOD_DIFF(period1, period2) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| QUARTER(d) | QUARTER(d) | 相同 |
| GBASE_TO_CHAR(d, '%S')<br>*UDR: SECOND(t)* | SECOND(t) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| *UDR: SEC_TO_TIME(s)* | SEC_TO_TIME(s) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| GBASE_TO_DATE(s,fmt) | STR_TO_DATE(string, format_mask) | 函数名称不一样 |
| *UDR: SUBDATE(d,n)* | SUBDATE(d,n) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| *UDR: SUBTIME(t,n)* | SUBTIME(t,n) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| GBASE_TO_CHAR(t, '%T')<br>*UDR: TIME(e)* | TIME(expression) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| GBASE_TO_CHAR(t,fmt)<br>*UDR: TIME_FORMAT(t,fmt)* | TIME_FORMAT(t,f) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| *UDR: TIME_TO_SEC(t)* | TIME_TO_SEC(t) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| *UDR: TIMEDIFF(t1,t2)* | TIMEDIFF(time1, time2) | 不支持该函数，可通过自定义函数实现 |

| | | |
|---|---|---|
| | | 自定义函数参考附录 B |
| | TIMESTAMP(expression, interval) | 不支持该函数，因与内部函数冲突 |
| *UDR: TO_DAYS(d)* | TO_DAYS(d) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| *UDR: WEEK(d,mode)* | WEEK(d,mode) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| WEEKDAY(d) | WEEKDAY(d) | 参数名称相同，返回值有差异。<br>MySQL 周一是 0，周日对应 6<br>GBase 8s 周日是 0，周六对应 6 |
| *UDR: WEEKOFYEAR(d)* | WEEKOFYEAR(d) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |
| YEAR(d) | YEAR(d) | 相同 |
| *UDR: YEARWEEK(d,mode)* | YEARWEEK(date, mode) | 不支持该函数，可通过自定义函数实现<br>自定义函数参考附录 B |

MySQL 的 DATE_FORMAT(d,fmt)等日期时间格式化函数和 GBASE 8s 的 GBASE_TO_CHAR(d,fmt)等日期时间格式化函数的 fmt（格式化参数）上有区别，参考如下：

| GBase 8s | MySQL | 描述及差异 | 示例 |
|---|---|---|---|
| %a | %a | 缩写星期名 | 三 或者 Wed |
| %b 或 %h | %b | 缩写月份名称 | 1 月 或者 Jan |
| %m | %c | 月，数值 | 1 |
| 无对应 | %D | 带有英文前缀的月中的天 | 11th |
| %d | %d | 月的天，数值(00-31) | 11 |
| %e | %e | 月的天，数值(0-31) | 11 |
| %F5 | %f | 微秒 | 123456 |
| %H | %H | 小时 (00-23) | 21 |
| %I | %h | 小时 (01-12) | 09 |
| %I | %I | 小时 (01-12) | 09 |
| %M | %i | 分钟，数值(00-59) | 59 |
| 无对应 | %j | 年的天 (001-366) | 011 |
| %H | %k | 小时 (0-23) | 21 |
| %I | %l | 小时 (1-12) | 09 |
| %B | %M | 月份名称全称 | 一月 或者 January |
| %m | %m | 月，数值(00-12) | |
| %p | %p | AM 或 PM | 上午 或者 AM |
| 需拼接 | %r | 时间，12-小时（hh:mm:ss AM 或 PM） | 09:40:30 AM |
| %S | %S | 秒(00-59) | 59 |
| %S | %s | 秒(00-59) | 59 |
| %T | %T | 时间, 24-小时 (hh:mm:ss) | 21:40:30 |
| 无对应 | %U | 周 (00-53) 星期日是一周的第一天 | 02 |
| 无对应 | %u | 周 (00-53) 星期一是一周的第一天 | 02 |

| | | | |
|---|---|---|---|
| 无对应 | %V | 周 (01-53) 星期日是一周的第一天，与 %X 使用 | 02 |
| 无对应 | %v | 周 (01-53) 星期一是一周的第一天，与 %x 使用 | 02 |
| %A | %W | 星期名 | 周三 或者 Wednesday |
| %w | %w | 周的天 （0=星期日, 6=星期六） | 3 |
| 无对应 | %X | 年，其中的星期日是周的第一天，4 位，与 %V 使用 | 2023 |
| 无对应 | %x | 年，其中的星期一是周的第一天，4 位，与 %v 使用 | 2023 |
| %Y | %Y | 年，4 位 | 2023 |
| %y | %y | 年，2 位 | 23 |

### 5.3.4 高级函数

| GBase 8s | MySQL | 差异描述 |
|---|---|---|
| *Java UDR: BIN(x)* | BIN(x) | 不支持该函数，可通过自定义函数实现<br>可以使用 Java UDR 的 BIN(x) |
| | BINARY(s) | 不支持该函数 |
| CASE expression<br>WHEN condition1 THEN result1<br>WHEN condition2 THEN result2<br>...<br>WHEN conditionN THEN resultN<br>ELSE result<br>END | CASE expression<br>WHEN condition1 THEN result1<br>WHEN condition2 THEN result2<br>...<br>WHEN conditionN THEN resultN<br>ELSE result<br>END | 相同 |
| CAST(x AS type) | CAST(x AS type) | 相同，但需注意 type 不指定长度时的默认值 |
| COALESCE(expr1, expr2, ...., expr_n) | COALESCE(expr1, expr2, ...., expr_n) | 相同 |
| DBINFO('sessionid') | CONNECTION_ID() | 函数名称不一样 |
| *Java UDR: CONV(x,f1,f2)* | CONV(x,f1,f2) | 不支持该函数，可通过自定义函数实现<br>可以使用 Java UDR 的 CONV(x,f1,f2) |
| | CONVERT(s, type)<br>CONVERT(s USING cs) | 不支持该函数 |
| USER<br>CURRENT_USER | CURRENT_USER()<br>SESSION_USER()<br>SYSTEM_USER()<br>USER() | 一个是函数，一个是表达式 |
| DBINFO('dbname') | DATABASE() | 函数名称不一样 |
| DECODE(expr,when,v1,v2)<br>CASE expr WHEN when THEN v1 ELSE v2 END | IF(expr,v1,v2) | 不支持该函数，必须使用其他实现方式代替 |

| | | |
|---|---|---|
| NVL(v1,v2) 或 ISNULL(v1,v2)<br>*Java UDR: IFNULL(v1,v2)* | IFNULL(v1,v2) | 函数名称不一样<br>可以使用 Java UDR 的 IFNULL(v1,v2) |
| | ISNULL(expr) | 存在函数名称一样，含义不一样的内置函数 |
| DBINFO('sqlca.sqlerrd1')<br>DBINFO('serial8')<br>DBINFO('bigserial') | LAST_INSERT_ID() | 函数名称不一样，起决于序列的数据类型 |
| NULLIF(expr1, expr2) | NULLIF(expr1, expr2) | 相同 |
| DBINFO('version', 'full') | VERSION() | 函数名称不一样 |
| *Java UDR: encryptAES(x,y)* | AES_ENCRYPT(x,y) | 函数名称不一样，加密解密需配对<br>可以使用 Java UDR: encryptAES(x,y) |
| *Java UDR: decryptAES(x,y)* | AES_DECRYPT(x,y) | 函数名称不一样，加密解密需配对<br>可以使用 Java UDR: encryptAES(x,y) |
| *Java UDR: HEXSTR(x)* | HEX(x) | 存在函数名称一样，含义不一样的内置函数，需通过自定义函数代替<br>可以使用 Java UDR 的 HEXSTR(x) |
| *Java UDR: UNHEX(x)* | UNHEX(x) | 不支持该函数，可通过自定义函数实现<br>可以使用 Java UDR 的 UNHEX(x) |

GBase 8s 与 MySQL 在高级函数上比较大的区别：函数名称不一样，函数名称一样功能或者结果不一样。

GBase 8s 完全不支持 BINARY(s)、CONVERT(s USING cs)函数，仅能通过逻辑上调整语句写法。

MySQL 中的 CAST(x AS type)，若 type 为 CHAR 或者 VARCHAR 类似，迁移到 GBase 8s 时，需要指定 CHAR(n)或 VARCHAR(n)的长度，因为 GBase 8s 默认长度为 1。

MySQL 中的 IF(expr,v1,v2)函数，如果 expr 为表达式或者值测试时，必须使用 CASE WHEN 或 DECODE()函数来代替，因为在 GBase 8s 中，函数中出现等号" =" ，表示的是显式的指定形参的值，从而造成冲突。同时 IF 还是保留关键字，无法创建同名的函数。

MySQL 中的 ISNULL(expr)，HEX(x)在 GBase 8s 中存在同名函数，但含义不一样，故迁移时必须创建其他名称的函数，或者更换写法。

# 5.4 存储过程语言 SPL(Stored Procedure Language)

## 5.4.1 存储过程 SPL 概览

GBase 8s 的存储过程和方法都可以看做是 UDR(user-defined routines)，一个 UDR 可以使用 SPL 或是其他外部语言，例如 C 或 JAVA。存储过程就是一个不返回值的程序。SPL 语句只能在存储过程中使用，SPL 程序以可执行的格式被解析，优化，存储在系统目录表中。

GBase 8s 的存储过程支持输入，输出和输入输出参数，并且可以用在 SQL 语句中任何可以使用表达式的地方。

变量定义和赋值

DEFINE， LET

流程控制

分支控制 IF

循环控制 FOR，FOREACH，WHILE, LOOP

EXIT， CONTINUE

函数调用与返回

CALL， SYSTEM ，RETURN

错误处理和调试

TRACE， ON EXCEPTION， RAISE EXCEPTION

## 5.4.2 参数限制

GBase 8s 的存储过程参数上限是 341 个，使用 JAVA 编写的 UDR 也适用这个限制,C 语言编写的 UDR 最多支持 102 个参数。Java 编写的 UDR 中 DECIMAL 数据类型的参数不能超过 9 个。C 语言编写的 UDR 中如果返回不透明数据类型的值，必须在 C 宿主变量的定义中指定 opaque_type。

# 5.5 开发环境

## 5.5.1 语言环境

GBase 8s 可以支持许多语言、文化和代码集。所有特定于文化的信息汇集于单个环境中，称为 Global Language Support（GLS）语言环境。除了 ASCII 美国英语之外，GLS 允许您在其他语言环境中工作并在 SQL 数据和标识中使用非 ASCII 字符。可以使用 GLS 功能来与特定语言环境定制保持一致。语言环境文件包括特定于文化的信息，如货币和日期格式以及整理顺序。

GBase 8s 通过 DB_LOCALE 和 CLIENT_LOCALE 来设置数据库的语言本地化支持设置。正确设置 GLS 语言环境相关变量(DB_LOCALE, CLIENT_LOCALE)，保证 GBase 8s 数据库服务器、客户端能正确的支持中文字符和支持使用中文的对象名。DB_LOCALE 和 CLIENT_LOCALE 的值由四部分组成（第 4 部分为可选），字符集不区分大小写：

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| < 语言 > _ | < 国家和地区 > . | < 字符集名 / 字符集编码 > | [@modifier] |

举例说明：

```
CLIENT_LOCALE=en_us.8859-1
CLIENT_LOCALE=en_us.819
# 以上两个为同一字符集：819 为 8859-1 的编码
DB_LOCALE=zh_cn.gb
```

**数据库服务端**

在创建数据库时（为了统一系统数据库与应用数据库的字符集，在创建数据库实例时），请按如下步骤设置数据库的 DB_LOCALE 值。

1)、设置环境变量 DB_LOCALE

```
export DB_LOCALE=zh_CN.utf8
```

2)、创建数据库

```
create database database_name;
```

3)、验证当前数据库字符集

```
SELECT dbs_collate FROM sysmaster:sysdbslocale
WHERE dbs_dbsname = 'database_name';
```

**客户端**

4)、当我们使用 ODBC/JDBC 连接数据库时，我们需要在连接信息中正确设置语言环境变量：

DB_LOCALE 和 CLIENT_LOCALE。

5)、设置语言环境变量

DB_LOCALE=zh_CN.utf8

CLIENT_LOCALE=zh_CN.utf8

特别注意，在创建数据库前请设置好环境变量 DB_LOCALE 为规划的字符集，因为数据库一旦创建就不能修改其字符集，除非重新创建。

在默认情况下 GBase 8s 将使用 en_us.8859-1 字符集。

支持的简体中文字符集有：

| 字符集名称 | 字符集编码 | 说明 |
|---|---|---|
| utf8 | zh_cn.57372 | UTF-8 |
| GB18030-2000 | zh_cn.5488 | 国标 GB18030-2000 |
| gb | zh_cn.57357 | 国标 GB2312 |

### 5.5.2 JDBC

**安装配置 JDBC**

JDBC Driver 安装软件集成在 CSDK 软件安装包中，也可以下载单独的 JDBC 安装包。安装完 JDBC Driver 后，需要将文件 gbasedbtjdbc.jar 添加到环境变量 CLASSPATH 中。

**设置 JDBC Driver 环境变量**

```
export CLASSPATH=${CLASSPATH}:${GBASEDBTDIR}/jdbc/lib/gbasedbtjdbc.jar
```

**设置 JDBC 连接字符串**

在 Java 程序中使用 JDBC 连接数据库，首先应该加载使用的 JDBC 类，JDBC Driver 的类名为 com.gbasedbt.jdbc.Driver。

建立 Java 程序与 GBase 8s 数据库的连接需要使用 DriverManager.getConnection() 方法，该方法中的 URL 参数为一个数据库的连接字符串，指定数据库的连接信息。使用不同的 JDBC 所需的 URL 参数也不相同。

使用 JDBC Driver 连接数据库，连接字符串的格式如下：

```
jdbc:gbasedbt-sqli://[{ip-address|host-name}:{port-number|service-name}][/dbname]:
```

```
GBASEDBTSERVER=servername[{;user=user;password=password]
|CSM=(SSO=database_server@realm,ENC=true)} [;name=value[;name=value]...]
```

其中， “jdbc:gbasedbt-sqli”指定使用的 JDBC 为 JDBC Driver；

“{ip-address|host-name}”为数据库服务器的 IP 地址主机名；

“{port-number|service-name}”为数据库服务器监听客户端连接的端口号或服务名；

“dbname”为数据库名；

“servername”为数据库实例名。

URL 示例：

```
jdbc:gbasedbt-sqli://10.13.147.9:9088/sysmaster:gbasedbtSERVER=demoserver
jdbc:gbasedbt-sqli://9.125.66.130:6346/db18030:gbasedbtSERVER=instance_name;
NEWCODESET=gb18030,gb18030-2000,5488;DB_LOCALE=zh_cn.gb18030-2000;
CLIENT_LOCALE=zh_cn.gb18030-2000;
```

**环境变量**

GBase 8s 常用环境变量：

| 字符集名称 | 说明 |
|---|---|
| GBASEDBTDIR | 数据库软件安装路径 |
| GBASEDBTSERVER | 数据库实例名称 |
| ONCONFIG | 配置文件名称，位于$GBASEDBTDIR/etc 目录下 |
| PATH | 命令读取路径（必须包含$GBASEDBTDIR/bin） |
| DB_LOCALE | 数据库字符集（与实例数据库使用字符集应当一致） |
| CLIENT_LOCALE | 客户端字符集（与数据库字符集应当一致） |
| DBDATE | 日期格式（如：Y4MD-） |

# 附录

## 附录 A GBase 8s ANSI 保留字

| | | |
|---|---|---|
| **A** | | |
| ABSOLUTE | ALLOCATE | ATTACH |
| ACCESS | ALTER | AUDIT |
| ACCESS_METHOD | AND | AUTHORIZATION |
| ADD | ANSI | AUTO |
| AFTER | ANY | AUTOFREE |
| AGGREGATE | APPEND | AVG |
| ALLIGNMENT | AS | AVOID_EXECUTE |
| ALL | ASC | AVOID_SUBQF |
| ALL_ROWS | AT | |
| **B** | | |
| BEFORE | BOOLEAN | BY |
| BEGIN | BOTH | BYTE |
| BETWEEN | BUFFERED | |
| BINARY | BUILTIN | |
| **C** | | |
| CACHE | CLOSE | CONNECTION |
| CALL | CLUSTER | CONST |
| CANNOTHASH | CLUSTERSIZE | CONSTRAINT |
| CARDINALITY | COARSE | CONSTRAINTS |
| CASCADE | COBOL | CONSTRUCTOR |
| CASE | CODESET | CONTINUE |
| CAST | COLLATION | COPY |
| CHAR | COLLECTION | COSTFUNC |
| CHAR_LENGTH | COLUMN | COUNT |
| CHARACTER | COMMIT | CRCOLS |
| CHARACTER_LENGTH | COMMITTED | CREATE |
| CHECK | COMMUTATOR | CURRENT |
| CLASS | CONCURRENT | CURSOR |

| | | |
|---|---|---|
| CLIENT | CONNECT | CYCLE |
| **D** | | |
| DATABASE | DECLARE | DIAGNOSTICS |
| DATAFILES | DECODE | DIRTY |
| DATASKIP | DEFAULT | DISABLED |
| DATE | DEFERRED | DISCONNECT |
| DATETIME | DEFERRED_PREPARE | DISTINCT |
| DAY | DEFINE | DISTRIBUTEBINARY |
| DBA | DELAY | DISTRIBUTEREFERENCES |
| DBDATE | DELETE | DISTRIBUTIONS |
| DBMONEY | DELIMITER | DOCUMENT |
| DBPASSWORD | DELUXE | DOMAIN |
| DEALLOCATE | DEREF | DONOTDISTRIBUTE |
| DEBUG | DESC | DORMANT |
| DEC | DESCRIBE | DOUBLE |
| DEC_T | DESCRIPTON | DROP |
| DECIMAL | DETACH | DTIME_T |
| **E** | | |
| EACH | ESCAPE | EXPLAIN |
| ELIF | EXCEPTION | EXPLICIT |
| ELSE | EXCLUSIVE | EXPRESS |
| ENABLE | EXEC | EXPRESSION |
| END | EXECUTE | EXTEND |
| ENUM | EXECUTEANYWHERE | EXTENT |
| ENVIRONMENT | EXISTS | EXTERNEXTERNAL |
| ERROR | EXIT | |
| **F** | | |
| FAR | FIXED | FOUND |
| FETCH | FLOAT | FRACTION |
| FILE | FLUSH | FRAGMENT |
| FILLFACTOR | FOR | FREE |
| FILTERING | FOREACH | FROM |
| FIRST | FOREIGN | FUNCTION |

| | | |
|---|---|---|
| FIRST_ROWS | FORMAT | |
| FIXCHAR | FORTRAN | |
| **G-H** | | |
| GENERAL | GOTO | HAVING |
| GET | GRANT | HIGH |
| GK | GROUP | HOLD |
| GLOBAL | HANDLESNULLS | HOUR |
| GO | HASH | HYBRID |
| **I** | | |
| IF | INDICATOR | INTERNAL |
| IFX_INT8_T | GBASEDBT | INTERNALLENGTH |
| IFX_LO_CREATE_SPEC_T | INIT | INTERVAL |
| IFX_LO_STAT_T | INNER | INTO |
| IMMEDIATE | INSERT | INTERVL_T |
| IMPLICIT | INSTEAD | IS |
| IN | INT | ISCANNONICAL |
| INCREMENT | INT8 | ISOLATION |
| INDEX | INTEG | ITEM |
| INDEXES | INTEGER | ITERATOR |
| **J-K** | | |
| JOIN | KEEP | KEY |
| **L** | | |
| LABELEQ | LAST | LOCALLOCATOR |
| LABELGE | LEADING | LOCK |
| LABELGLB | LEFT | LOCKS |
| LABELGT | LET | LOG |
| LABELLE | LEVEL | LONG |
| LABELLT | LIKE | LOW |
| LABELUB | LIST | LOWER |
| LABELTOSTRING | LISTING | VARCHAR |
| LANGUAGE | LOC_T | |
| **M** | | |
| MATCHES | MEDIUM | MODERATE |

| | | |
|---|---|---|
| MAX | MEMORY_RESIDENT | MODIFY |
| MAXERRORS | MIDDLE | MODULE |
| MAXLEN | MIN | MONEY |
| MAXVALUE | MINUTE | MONTH |
| MDY | MINVALUE | MOUNTING |
| MEDIAN | MODE | MULTISET |
| **N** | | |
| NAME | NOCYCLE | NORMAL |
| NCHAR | NOMAXVALUE | NOT |
| NEGATOR | NOMIGRATE | NOTEMPLATEARG |
| NEW | NOMINVALUE | NULL |
| NEXT | NON_RESIDENT | NUMERIC |
| NO | NONE | NVARCHAR |
| NOCACHE | NOORDER | NVL |
| **O** | | |
| OCTET_LENGTH | OPAQUE | OPTION |
| OF | OPCLASS | OR |
| OFF | OPEN | ORDER |
| OLD | OPERATIONAL | OUT |
| ON | OPTICAL | OUTER |
| ONLY | OPTIMIZATION | |
| **P** | | |
| PAGE | PLI | PRIVATE |
| PARALLELIZABLE | PLOAD | PRIVILEGES |
| PARAMETER | PRECISION | PROCEDURE |
| PASCAL | PREPARE | PUBLIC |
| PASSEDBYVALUE | PREVIOUS | PUT |
| PDQPRIORITY | PRIMARY | |
| PERCALL_COST | PRIOR | |
| **R** | | |
| RAISE | REMAINDER | RETURNS |
| RANGE | RENAME | REUSE |
| RAW | REOPTIMIZATION | REVOKE |

| | | |
|---|---|---|
| READ | REPEATABLE | ROBIN |
| REAL | REPLICATION | ROLE |
| RECORDEND | RESERVE | ROLLBACK |
| REF | RESOLUTION | ROLLFORWARD |
| REFERENCES | RESOURCE | ROUND |
| REFERENCING | RESTART | ROUTINE |
| REGISTER | RESTRICT | ROW |
| REJECTFILE | RESUME | ROWID |
| RELATIVE | RETAIN | ROWIDS |
| RELEASE | RETURN | ROWS |
| RETURNING | ROWNUM | |
| **S** | | |
| SAMEAS | SHARE | START |
| SAMPLES | SHORT | STATIC |
| SCHEDULE | SIGNED | STATISTICS |
| SCHEMA | SIZE | STDEV |
| SCRATCH | SKALL | STEP |
| SCROLL | SKINHIBIT | STOP |
| SECOND | SKSHOW | STORAGE |
| SECONDARY | SMALLFLOAT | STRATEGIES |
| SECTION | SMALLINT | STRING |
| SELCONST | SOME | STRINGTOLABEL |
| SELECT | SPECIFIC | STRUCT |
| SELFUNC | SQL | STYLE |
| SEQUENCE | SQLCODE | SUBSTR |
| SERIAL | SQLCONTEXT | SUBSTRING |
| SERIAL8 | SQLERROR | SUM |
| SERIALIZABLE | SQLWARNING | SUPPORT |
| SERVERUUID | STABILITY | SYNC |
| SESSION | STACK | SYNONYM |
| SET | STANDARD | SYSTEM |
| **T** | | |
| TABLE | TO | TOP |

| | | |
|---|---|---|
| TEMP | TODAY | TRIGGERS |
| TEXT | TRACE | TRIM |
| THEN | TRAILING | TRUNCATE |
| TIME | TRANSACTION | TYPE |
| TIMEOUT | TRIGGER | TYPEDEF |
| **U** | | |
| UNCOMMITTED | UNITS | USAGE |
| UNDER | UNLOCK | USE_SUBQF |
| UNION | UNSIGNED | USER |
| UNIQUE | UPDATE | USING |
| **V** | | |
| VALUE | VARIABLE | VIEW |
| VALUES | VARIANCE | VIOLATIONS |
| VAR | VARIANT | VOID |
| VARCHAR | VARYING | VOLATILE |
| **W** | | |
| WAIT | WHERE | WORK |
| WARNING | WHILE | WRITE |
| WHEN | WITH | WHENEVER |
| WITHOUT | | |
| **X-Y** | | |
| XLOAD | XUNLOAD | YEAR |

# 附录 B GBase 8s 自定义 MySQL 兼容函数

自定义 MySQL 兼容函数，可能需要根据实际情况调整。

## B.1 字符类函数

FIND_IN_SET (s1,s)函数：在 s 字符串集中查找 s1 的位置

```
-- find_in_set
drop function if exists find_in_set;
create dba function find_in_set(str varchar(255),strlist varchar(8192))
returns int with (not variant)
```

```
  define currp  int;
  define lastp  int;
  define sepnum int;
  define strlen int;
  define strtmp varchar(8192);

  let lastp  = 1;
  let sepnum = 0;
  let strtmp = strlist || ",";
  let strlen = length(strtmp);
  FOR currp = 1 TO strlen
    IF substr(strtmp,currp,1) = ',' THEN
      let sepnum = sepnum + 1;
      IF substr(strtmp,lastp,currp - lastp) = str THEN
        RETURN sepnum;
      END IF;
      let lastp = currp + 1;
    END IF;
  END FOR;
  RETURN 0;
end function;
```

INSERT(s1,x,len,s2)函数：用 s2 替代 s1 中 x 位开始的 len 长度的字符串

```
-- INSERT(s1,x,len,s2)
drop function if exists INSERT(lvarchar,int,int,lvarchar);
create dba function INSERT(s1 lvarchar,x int,len int,s2 lvarchar)
returns lvarchar with (not variant)
  on exception
    return s1;
  end exception;
  if x < 1 or len < 1 then
    return s1;
  end if;
  return substr(s1,1,x-1) || s2 || substr(s1,x+len-1);
end function;
```

REPEAT(s,n)函数：复制 n 份 s

```
-- REPEAT(s,n)
drop function if exists REPEAT(lvarchar,int);
create dba function REPEAT(s lvarchar, n int)
returns lvarchar with (not variant)
  define tmpstr lvarchar;
  define tmppos int;
  on exception
```

```
    return s;
  end exception;
  let tmpstr = s;
  let tmppos = 1;
  while tmppos < n
    let tmpstr = tmpstr || s;
    let tmppos = tmppos + 1;
  end while;
  return tmpstr;
end function;
```

STRCMP(str1,str2)函数：比较 str1 和 str2，相同返回 0，大于返回 1，小于返回-1

```
-- STRCMP(str1,str2)
drop function if exists STRCMP(lvarchar,lvarchar);
create dba function STRCMP(str1 lvarchar, str2 lvarchar)
returns int with (not variant)
  on exception
    return null;
  end exception;
  if str1 = str2 then
    return 0;
  elif str1 > str2 then
    return 1;
  else
    return -1;
  end if;
end function;
```

GROUP_CONCAT(s)聚集函数：WM_CONCAT()函数别名

```
-- group_concat
drop aggregate if exists group_concat;
create aggregate group_concat with
(
  init = str_sum_init,
  iter = str_sum_iter,
  combine = str_sum_combine
);
```

FORMAT(x,n,loc)函数：格式化数值 x

```
-- format(x,n [,loc])
drop function if exists format(decimal,int,varchar);
create function format(x decimal(32,12), n int, loc varchar(10) default 'en_us')
returns varchar(40) with (not variant)
  define str varchar(40);
```

```
  define lstr varchar(40);
  define rstr varchar(40);
  define llen int;
  on exception
    return null;
  end exception;
  let str = round(x,n);
  let llen = instr(str,'.');
  let lstr = substr(str,1,llen - 1);
  if lower(loc) = 'en_us' then
    let lstr = gbase_to_char(lstr,'###,###,###,###,###,###,###');
  end if;

  -- had fraction
  if n > 0 then
    let rstr = substr(str, llen, n + 1);
    let str = lstr || rstr;
  else
    let str = lstr;
  end if;
  return ltrim(str);
end function;
```

CONCAT(s1,s2,...s16)函数：字符串连接函数（仅支持 16 个字段）

```
-- concat(s1,s2,...s16)
drop function if exists concat(varchar,varchar,varchar,varchar,
                               varchar,varchar,varchar,varchar,
                                 varchar,varchar,varchar,varchar,
                                 varchar,varchar,varchar,varchar);
create function concat(s1 varchar(255) default '',s2 varchar(255) default '',
                       s3 varchar(255) default '',s4 varchar(255) default '',
                       s5 varchar(255) default '',s6 varchar(255) default '',
                       s7 varchar(255) default '',s8 varchar(255) default '',
                       s9 varchar(255) default '',s10 varchar(255) default '',
                       s11 varchar(255) default '',s12 varchar(255) default '',
                       s13 varchar(255) default '',s14 varchar(255) default '',
                       s15 varchar(255) default '',s16 varchar(255) default '')
returns varchar(255) with (not variant)
  on exception
    return null;
  end exception;
  return s1 || s2 || s3 || s4 || s5 || s6 || s7 || s8 || s9 || s10 || s11 || s12 || s13 || s14 || s15
|| s16;
end function;
```

CONCAT_WS(del,s1,s2,...s16)函数：字符串连接函数，使用分隔符 del（仅支持 16 个字段）

```
-- concat_ws(del,s1,s2,...s16)
drop function if exists concat_ws(varchar,varchar,varchar,varchar,varchar,
                                  varchar,varchar,varchar,varchar,
                                    varchar,varchar,varchar,varchar,
                                    varchar,varchar,varchar,varchar);
create function concat_ws(del varchar(255), s1 varchar(255) default null,
        s2 varchar(255) default null,s3 varchar(255) default null,s4 varchar(255) default null,
        s5 varchar(255) default null,s6 varchar(255) default null,s7 varchar(255) default null,
        s8 varchar(255) default null, s9 varchar(255) default null,s10 varchar(255) default null,
        s11 varchar(255) default null,s12 varchar(255) default null,s13 varchar(255) default null,
        s14 varchar(255) default null,s15 varchar(255) default null,s16 varchar(255) default null)
returns varchar(255) with (not variant)
  define str varchar(255);
  on exception
    return null;
  end exception;
  if del is null then
    return null;
  end if;
  if s1 is null then
    let str = nvl(s2,'');
  else
    let str = s1 || del || s2;
  end if;
  let str = str || nvl2(s3,del||s3,'') || nvl2(s4,del||s4,'');
  let str = str || nvl2(s5,del||s5,'') || nvl2(s6,del||s6,'')
  let str = str || nvl2(s7,del||s7,'') || nvl2(s8,del||s8,'');
  let str = str || nvl2(s9,del||s9,'') || nvl2(s10,del||s10,'');
  let str = str || nvl2(s11,del||s11,'') || nvl2(s12,del||s12,'');
  let str = str || nvl2(s13,del||s13,'') || nvl2(s14,del||s14,'');
  let str = str || nvl2(s15,del||s15,'') || nvl(s16,'');
  return str;
end function;
```

## B.2 数值函数

PI()函数：返回 PI 值，有效数字 20 位。

```
-- PI()
drop function if exists PI();
```

```
create dba function PI()
returns decimal(32,20) with (not variant)
  return 3.141592653589793116;
end function;
```

LOG2(N)函数：返回 LOG2(N)值，有效数字 20 位。

```
-- LOG2(N)
drop function if exists LOG2(decimal);
create dba function LOG2(N decimal(32,20))
returns decimal(32,20) with (not variant)
  return LOGN(N)/LOGN(2);
end function;
```

## B.3 日期时间函数

ADDDATE(d,n)函数：日期 d 上增加天数或者时间间隔

```
-- ADDDATE(d,n)
-- 参数为数字，默认为天
drop function if exists ADDDATE(datetime year to second, int);
create dba function ADDDATE(d datetime year to second,n int)
returns datetime year to second with (not variant)
  on exception
    return null;
  end exception;
  return d + n units day;
end function;

-- 参数为 interval，注意使用本数据库的写法，例如：interval(10 11:12:13) day to second
drop function if exists ADDDATE(datetime year to second, interval);
create dba function ADDDATE(d datetime year to second, n interval day to second)
returns datetime year to second with (not variant)
  on exception
    return null;
  end exception;
  return d + n;
end function;
```

ADDTIME(t,n)：日期时间 t 上增加秒数或者时间间隔

```
-- ADDTIME(t,n)
-- 参数为数字，默认为秒
drop function if exists ADDTIME(datetime year to second, int);
```

```
create dba function ADDTIME(t datetime year to second,n int)
returns datetime year to second with (not variant)
  on exception
    return null;
  end exception;
  return t + n units second;
end function;

-- 参数为 interval，注意使用本数据库的写法，例如: interval(10 11:12:13) day to second
drop function if exists ADDTIME(datetime year to second, interval);
create dba function ADDTIME(t datetime year to second, n interval day to second)
returns datetime year to second with (not variant)
  on exception
    return null;
  end exception;
  return t + n;
end function;
```

CURDATE()和 CURRENT_DATE()函数：返回当前日期

```
-- CURDATE()/CURRENT_DATE()返回当前日期
drop function if exists curdate();
create dba function CURDATE()
returns varchar(10) with (not variant)
  return gbase_to_char(today,'%Y-%m-%d');
end function;

drop function if exists CURRENT_DATE();
create dba function CURRENT_DATE()
returns char(10) with (not variant)
  return gbase_to_char(today,'%Y-%m-%d');
end function;
```

CURTIME()和 CURRENT_TIME()函数：返回当前时间

```
-- CURRENT_TIME()/CURTIME()返回当前时间
drop function if exists CURTIME();
create dba function CURTIME()
returns char(8) with (not variant)
  return gbase_to_char(current year to second, '%H:%M:%S');
end function;

drop function if exists CURRENT_TIME();
create dba function CURRENT_TIME()
returns char(8) with (not variant)
  return gbase_to_char(current year to second, '%H:%M:%S');
```

```
end function;
```

CURRENT_TIMESTAMP()函数：返回当前日期和时间，返回类型是 datetime

```
-- CURRENT_TIMESTAMP()返回当前日期和时间
drop function if exists CURRENT_TIMESTAMP();
create dba function CURRENT_TIMESTAMP()
returns datetime year to second with (not variant)
  return current year to second;
end function;
```

NOW()函数：返回当前日期和时间，返回类型是 datetime

```
-- NOW()返回当前日期和时间
drop function if exists NOW();
create dba function NOW()
returns datetime year to second with (not variant)
  return current year to second;
end function;
```

DATEDIFF(d1,d2)函数：计算日期 d1->d2 之间相隔的天数

```
-- DATEDIFF(d1,d2)计算日期 d1->d2 之间相隔的天数
drop function if exists DATEDIFF(datetime year to second, datetime year to second);
create dba function DATEDIFF(d1 datetime year to second, d2 datetime year to second)
returns int with (not variant)
  return substr(d1 - d2,1,10);
end function;
```

DATE_ADD(d,n)函数：日期时间 d 增加时间间隔 interval

```
-- DATE_ADD(d,n)
-- 参数为 interval，注意使用本数据库的写法，例如：interval(10 11:12:13) day to second
drop function if exists DATE_ADD(datetime year to second, interval);
create dba function DATE_ADD(d datetime year to second, n interval day to second)
returns datetime year to second with (not variant)
  on exception
    return null;
  end exception;
  return d + n;
end function;
```

DATE_FORMAT(d,fmt)函数：日期时间格式转换函数，与 GBASE_TO_CHAR()函数类似，差异是 fmt 格式

```
-- DATE_FORMAT(d,fmt)
-- fmt 有差异需转换
drop function if exists DATE_FORMAT(datetime year to second, varchar);
create dba function DATE_FORMAT(d datetime year to second, fmt varchar(40))
```

```
returns varchar(40) with (not variant);
  define tmpfmt varchar(40);
  on exception
    return null;
  end exception;
  let tmpfmt = replace(fmt,'f','F5');
  let tmpfmt = replace(tmpfmt,'h','I');
  let tmpfmt = replace(tmpfmt,'k','H');
  let tmpfmt = replace(tmpfmt,'l','I');
  let tmpfmt = replace(tmpfmt,'M','B');
  let tmpfmt = replace(tmpfmt,'s','S');
  let tmpfmt = replace(tmpfmt,'W','A');
  let tmpfmt = replace(tmpfmt,'i','M');
  return gbase_to_char(d,tmpfmt);
end function;
```

DATE_SUB(d,n)函数：日期时间 d 减少时间间隔 interval

```
-- DATE_SUB(d,n)
-- 参数为 interval，注意使用本数据库的写法，例如：interval(10 11:12:13) day to second
drop function if exists DATE_SUB(datetime year to second, interval);
create dba function DATE_SUB(d datetime year to second, n interval day to second)
returns datetime year to second with (not variant)
  on exception
    return null;
  end exception;
  return d - n;
end function;
```

DAYNAME(d)函数：返回日期时间 d 对应的星期几

```
-- DAYNAME(d)返回日期 d 是星期几，如 Monday,Tuesday
drop function if exists DAYNAME(datetime year to second);
create dba function DAYNAME(d datetime year to second)
returns varchar(20) with (not variant)
  on exception
    return null;
  end exception;
  return gbase_to_char(d, '%A');
end function;
```

DAYOFWEEK(d)函数：返回日期时间中的本周第几天

```
-- DAYOFWEEK(d)函数：指定日期时间 d 为当周的第几天，字符型
-- DAYOFWEEK(d)计算日期 d 是本周的第几天
drop function if exists DAYOFWEEK(datetime year to second);
create dba function DAYOFWEEK(d1 datetime year to second)
```

```
returns int with (not variant)
  return weekday(d1) + 1;
end function;
```

DAYOFMONTH(d)函数：返回日期时间中的 day

```
-- DAYOFMONTH(d)计算日期 d 是本月的第几天
drop function if exists DAYOFMONTH(datetime year to second);
create dba function DAYOFMONTH(d datetime year to second)
returns int with (not variant)
  return day(d);
end function;
```

FROM_DAYS(n)函数：返回指定 n 天相对于 0000-00-00 的对应的日期值（注 0001-01-01 为 366）

```
-- FROM_DAYS(n) 返回日期值
drop function if exists FROM_DAYS(int);
create dba function FROM_DAYS(n int)
returns char(10) with (not variant)
  on exception
    return '0000-00-00';
  end exception;
  if n < 366 then
    return '0000-00-00';
  else
    return gbase_to_char(gbase_to_date('0001-01-01','%Y-%m-%d') + (n - 366) units day, '%Y-%m-%d');
  end if;
end function;
```

HOUR(t)函数：返回日期时间 t 中的 hour

```
-- HOUR(t) 返回当前小时数
drop function if exists HOUR(datetime year to second);
create dba function HOUR(t datetime year to second)
returns int with (not variant)
  on exception
    return null;
  end exception;
  return gbase_to_char(t,'%H');
end function;
```

MAKEDATE(year, dayofyear)函数：返回指定 year 和 dayofyear 天数的字符型日期值

```
-- MAKEDATE(year, day-of-year)
drop function if exists MAKEDATE(int, int);
create dba function MAKEDATE(d int, doy int)
returns char(10) with (not variant)
```

```
  on exception
    return null;
  end exception
  if doy < 0 then
    return null;
  else
    return gbase_to_char(gbase_to_date(d || '-01-01','%Y-%m-%d') + doy units day,'%Y-%m-%d');
  end if;
end function;
```

MAKETIME(hh24,mi,ss)函数：返回字符型时间值

```
-- MAKETIME(hh24, mi, ss)
drop function if exists MAKETIME(int, int, int);
create dba function MAKETIME(hh24 int, mi int, ss int)
returns char(8) with (not variant)
  on exception
    return null;
  end exception;
  if hh24 < 0 or hh24 > 23 or mi < 0 or mi > 59 or ss < 0 or ss > 59 then
    return null;
  else
    return lpad(hh24,2,'0') || ':' || lpad(mi,2,'0') || ':' || lpad(ss,2,'0');
  end if;
end function;
```

MICROSECOND(d)函数：返回指定日期时间 d 的微秒数。需注意 GBase 8s 默认仅有 5 位，故需后加 0。

```
-- MICROSECOND(date)
drop function if exists MICROSECOND(datetime year to fraction(5));
create dba function MICROSECOND(d datetime year to fraction(5))
returns int with (not variant)
  return gbase_to_char(d, '%F5') || '0';
end function;
```

MINUTE(t)函数：返回日期时间 t 中的 minute

```
-- MINUTE(t) 返回当前分钟数
drop function if exists MINUTE(datetime year to second);
create dba function MINUTE(t datetime year to second)
returns int with (not variant)
  on exception
    return null;
  end exception;
  return gbase_to_char(t,'%M');
end function;
```

MONTHNAME(t)函数：返回指定日期时间 t 的月份名称

```
-- MONTHNAME(t) 返回当前月份名称
drop function if exists MONTHNAME(datetime year to second);
create dba function MONTHNAME(d datetime year to second)
returns varchar(20) with (not variant)
  on exception
    return null;
  end exception;
  return gbase_to_char(d,'%B');
end function;
```

PERIOD_ADD(p,n)函数：指定年月 YYYYmm 或者 YYmm（数字格式）增加 n 月后的年月 YYYYmm

```
-- PERIOD_ADD(p,n)
drop function if exists PERIOD_ADD(int, int);
create dba function PERIOD_ADD(p int, n int)
returns int with (not variant)
  define tmpstr char(6);
  on exception
    return null;
  end exception;
  let tmpstr = lpad(p,6,'0');
  if p < 9999 then
    if substr(tmpstr,3,2) > 49 then
      let tmpstr = '19' || substr(tmpstr,3);
    else
      let tmpstr = '20' || substr(tmpstr,3);
    end if;
  end if;
  return gbase_to_char(gbase_to_date(tmpstr,'%Y%m') + n units month,'%Y%m');
end function;
```

PERIOD_DIFF(p1,p2)函数：指定两个年月 YYYYmm 或者 YYmm（数字格式）的差值（月份差）

```
drop function if exists PERIOD_DIFF(int, int);
create dba function PERIOD_DIFF(p1 int, p2 int)
returns int with (not variant)
  define tmpstr1 char(6);
  define tmpstr2 char(6);
  on exception
    return null;
  end exception;
  let tmpstr1 = lpad(p1,6,'0');
  let tmpstr2 = lpad(p2,6,'0');
  if p1 < 9999 then
```

```
    if substr(tmpstr1,3,2) > 49 then
      let tmpstr1 = '19' || substr(tmpstr1,3);
    else
      let tmpstr1 = '20' || substr(tmpstr1,3);
    end if;
  end if;
  if p2 < 9999 then
    if substr(tmpstr2,3,2) > 49 then
      let tmpstr2 = '19' || substr(tmpstr2,3);
    else
      let tmpstr2 = '20' || substr(tmpstr2,3);
    end if;
  end if;
  return (substr(tmpstr1,1,4) - substr(tmpstr2,1,4)) * 12 + (substr(tmpstr1,5) - substr(tmpstr2,5));
end function;
```

SECOND(t)函数：返回指定日期时间 t 中的 second 值

```
-- SECOND(t) 返回当前秒数
drop function if exists SECOND(datetime year to second);
create dba function SECOND(t datetime year to second)
returns int with (not variant)
  on exception
    return null;
  end exception;
  return gbase_to_char(t,'%S');
end function;
```

SEC_TO_TIME(s)函数：指定当天的秒数 s，转换成对应的字符型时间值

```
-- SEC_TO_TIME(s)
drop function if exists SEC_TO_TIME(int);
create dba function SEC_TO_TIME(s int)
returns char(8) with (not variant)
  return gbase_to_char(gbase_to_date('00:00:00','%H:%M:%S') + s units second, '%H:%M:%S');
end function;
```

SUBDATE(d,n)函数：指定日期时间 d 减去 n 天后的时间

```
-- SUBDATE(d,n)
-- 参数为数字，默认为天
drop function if exists SUBDATE(datetime year to second, int);
create dba function SUBDATE(d datetime year to second,n int)
returns datetime year to second with (not variant)
  on exception
    return null;
  end exception;
```

```
  return d - n units day;
end function;
```

SUBTIME(t,n)函数：指定日期时间 t 减去 n 秒后的时间

```
-- SUBTIME(t,n)
-- 参数为数字，默认为秒
drop function if exists SUBTIME(datetime year to second, int);
create dba function SUBTIME(t datetime year to second,n int)
returns datetime year to second with (not variant)
  on exception
    return null;
  end exception;
  return t - n units second;
end function;
```

TIME(t)函数：返回指定日期时间 t 的字符型时间值（hh24:mi:ss）

```
-- TIME(e)
-- datetime
drop function if exists TIME(datetime year to second);
create dba function TIME(t datetime year to second)
returns char(8) with (not variant)
  on exception
    return '00:00:00';
  end exception;
  if t is null then
    return null;
  end if;
  return gbase_to_char(t,'%T');
end function;

-- 仅 time
drop function if exists TIME(datetime hour to second);
create dba function TIME(t datetime hour to second)
returns char(8) with (not variant)
  on exception
    return '00:00:00';
  end exception;
  if t is null then
    return null;
  end if;
  return gbase_to_char(t,'%T');
end function;
```

TIME_FORMAT(d,fmt)函数：日期时间 d 格式 fmt 转换，与 GBASE_TO_CHAR()函数类似

```
-- TIME_FORMAT(d,fmt)
-- fmt 有差异需转换
drop function if exists TIME_FORMAT(datetime hour to second, varchar);
create dba function TIME_FORMAT(d datetime hour to second, fmt varchar(40))
returns varchar(40) with (not variant);
  define tmpfmt varchar(40);
  on exception
    return null;
  end exception;
  let tmpfmt = replace(fmt,'f','F5');
  let tmpfmt = replace(tmpfmt,'h','I');
  let tmpfmt = replace(tmpfmt,'k','H');
  let tmpfmt = replace(tmpfmt,'l','I');
  let tmpfmt = replace(tmpfmt,'M','B');
  let tmpfmt = replace(tmpfmt,'s','S');
  let tmpfmt = replace(tmpfmt,'W','A');
  let tmpfmt = replace(tmpfmt,'i','M');
  return gbase_to_char(d,tmpfmt);
end function;
```

TIME_TO_SEC(t)函数：指定时间 t 转换成秒数

```
-- TIME_TO_SEC(t)
drop function if exists TIME_TO_SEC(datetime hour to second);
create dba function TIME_TO_SEC(d datetime hour to second)
returns int with (not variant);
  on exception
    return null;
  end exception;
  return gbase_to_char(d,'%H') * 3600 + gbase_to_char(d,'%M') * 60 + gbase_to_char(d,'%S');
end function;
```

TIMEDIFF(t1,t2)函数：返回两个日期时间 t1 和 t2 之间的字符型时间间隔值

```
-- TIMEDIFF(t1,t2)
-- 仅 time
drop function if exists TIMEDIFF(datetime hour to second,datetime hour to second);
create dba function TIMEDIFF(d1 datetime hour to second, d2 datetime hour to second)
returns varchar(20) with (not variant);
  on exception
    return null;
  end exception;
  return d1 - d2;
end function;

-- datetime
```

```
drop function if exists TIMEDIFF(datetime year to second,datetime year to second);
create dba function TIMEDIFF(d1 datetime year to second, d2 datetime year to second)
returns varchar(20) with (not variant);
  define tmpitv varchar(20);
  define tmpstr varchar(20);
  define tmp_d varchar(20);
  on exception
    return null;
  end exception;
  let tmpitv = d1 - d2;
  let tmp_d = substr(tmpitv,1,10);
  let tmpstr = (abs(substr(tmpitv,1,10))*24 + substr(tmpitv,11,2))::int || substr(tmpitv,13,6);
  if instr(tmp_d,'-') > 0 then
    return '-' || tmpstr;
  else
    return tmpstr;
  end if;
end function;
```

TO_DAYS(d)函数：返回日期时间值相对于 0000-00-00 的天数（0001-01-01 为 366）

```
-- TO_DAYS(d) 返回日期值
drop function if exists TO_DAYS(datetime year to day);
create dba function TO_DAYS(d datetime year to day)
returns int with (not variant)
  on exception
    return null;
  end exception;
  return d::date::int + 693960;
end function;
```

WEEK(d,mode)函数：指定日期时间 d 是第几周（周日为第一天，mode 值无效）

```
-- WEEK(d,mode)
drop function if exists WEEK(datetime year to second, int);
create dba function WEEK(d datetime year to second, mode int default 0)
returns int with (not variant)
  on exception
    return null;
  end exception;
  return trunc(1 + (d - TRUNC(d, 'YEAR')) / 7);
end function;
```

WEEKOFYEAR(d)函数：指定日期时间 d 是当前的第几周（周日为第一天）

```
-- WEEKOFYEAR(d)
drop function if exists WEEKOFYEAR(datetime year to second);
```

```
-- day of year, first week is 01-01 ~ 01-07
create dba function WEEKOFYEAR(d datetime year to second)
returns int
with (not variant)
  return trunc(1 + (d - TRUNC(d, 'YEAR')) / 7);
end function;
```

YEARWEEK(d,mode)函数：返回指定日期时间 d 是当前的第几周，格式为年份+周数

```
drop function if exists YEARWEEK(datetime year to second, int);
create dba function YEARWEEK(d datetime year to second, mode int default 0)
returns int with (not variant)
  on exception
    return null;
  end exception;
  return year(d)*100 + trunc(1 + (d - TRUNC(d, 'YEAR')) / 7);
end function;
```

WEEKOFMONTH(d)函数：指定日期时间 d 为当月的第几周

```
-- WEEKOFMONTH(d) 计算当月第几周
drop function if exists weekofmonth(datetime year to second);
-- day of month, first week is 01~07
create dba function weekofmonth(d1 datetime year to second)
returns int with (not variant)
with (not variant)
  return trunc(1 + (d1 - TRUNC(d1, 'MONTH')) / 7);
end function;
```

DAYOFYEAR(d)函数：指定日期时间 d 为当年的第几天，格式为（000）字符型

```
-- DAYOFYEAR(d)计算日期 d 是本年的第几天
drop function if exists DAYOFYEAR(datetime year to second);
create dba function DAYOFYEAR(d1 datetime year to second)
returns char(3) with (not variant)
with (not variant)
  return lpad(trunc(1 + (d1 - TRUNC(d1, 'YEAR'))),3, '0');
end function;
```

UNIX_TIMESTAMP(dt)函数：将日期时间 dt 转换为自 1970-01-01 08:00:00 开始的秒数

```
drop function if exists UNIX_TIMESTAMP(datetime year to second);
-- unix_timestamp
create dba function UNIX_TIMESTAMP(datestr datetime year to second)
returns bigint with (not variant);
```

```
  define rc_char varchar(30);
  if datestr < datetime(1970-01-01 08:00:00) year to second then
    let rc_char = datetime(1970-01-01 08:00:00) year to second - datestr;
    return 0 - ((replace(substr(rc_char,1,9),'-')*86400)::bigint + substr(rc_char,17,2)*1 +
substr(rc_char,14,2)*60 + substr(rc_char,11,2)*3600);
  else
    let rc_char = datestr - datetime(1970-01-01 08:00:00) year to second;
    return ((substr(rc_char,1,9)*86400)::bigint + substr(rc_char,17,2)*1 + substr(rc_char,14,2)*60
+ substr(rc_char,11,2)*3600);
  end if;
end function ;
```

FROM_UNIXTIME(P)函数：返回自 1970-01-01 08:00:00 开始 p 秒的日期时间值

```
drop function if exists from_unixtime;
-- from_unixtime
create dba function from_unixtime(p bigint)
returns varchar(20) with (not variant);
  define v_day int;
  define v_hour int;
  define v_min int;
  define v_sec int;
  define v_dt datetime year to second;
  on exception
    return null;
  end exception;

  let v_day = p / 86400;
  let v_dt = datetime(1970-01-01 08:00:00) year to second + v_day units day;

  let v_hour = mod(p,86400) / 3600;
  let v_dt = v_dt + v_hour units hour;

  let v_min = mod(p,3600) / 60;
  let v_dt = v_dt + v_min units minute;

  let v_sec = mod(p,60);
  let v_dt = v_dt + v_sec units second;

  return to_char(v_dt,'yyyy-mm-dd hh24:mi:ss');
end function;
```

TIMESTAMPDIFF(unit,d1,d2)函数：返回指定日期时间 d1 和 d2 指定单位 unit 的间隔

```
-- TIMESTAMPDIFF(unit, d1, d2)
drop function if exists TIMESTAMPDIFF(varchar, datetime year to second, datetime year to second,
```

```
boolean);
-- SECOND,MINUTE,HOUR,DAY; d2>d1 为正数
create dba function TIMESTAMPDIFF(unit varchar(10), d1 datetime year to second, d2 datetime year to
second, isabs boolean default 'f')
returns int8 with (not variant)
  define tmp_num int8;
  define tmp_str varchar(20);
  define tmp_flag char(1);
  define tmp_y int;
  define tmp_m int;
  define tmp_d int;

  let tmp_str = d2 - d1;
  let tmp_num = substr(tmp_str,1,9)::int8;
  let tmp_flag = 'f';
  if tmp_num < 0 then
    let tmp_flag = 't';
  end if;
  case lower(unit)
    when 'day' then
      let tmp_num = tmp_num;
    when 'hour' then
      let tmp_num = abs(tmp_num) * 24 + substr(tmp_str,11,2)::int8;
    when 'minute' then
      let tmp_num = (abs(tmp_num) * 24 + substr(tmp_str,11,2)::int8) * 60 + substr(tmp_str,14,2);
    when 'second' then
      let tmp_num = ((abs(tmp_num) * 24 + substr(tmp_str,11,2)::int8) * 60 + substr(tmp_str,14,2))
* 60 + substr(tmp_str,17,2);
    else
      let tmp_num = 0;
  end case;

  if tmp_flag = 't' and isabs = 'f' then
    return '-' || tmp_num;
  else
    return tmp_num;
  end if;
end function;
```

# 附录 C GBase 8s Java UDR 函数

**创建 Java UDR 过程步骤:**

1，将共享库 gbasedbt-judr-x.x.x.jar 改名为 judr.jar，放置于数据库服务器上

比如：放到$GBASEDBTDIR/udr 目录下，则 judr.jar 的位置为$GBASEDBTDIR/udr/judr.jar

2，设置数据库的 JVP 环境

ONCONFIG 配置文件涉及到 JVP 的相关参数：

1)、VPCLASS 开启 jvp，用于处理 Java UDR，默认不开启，取消前面的注释符'#'

```
# 如果 java udr 使用量多，可以额外增加几个 jvp，即增加 num
VPCLASS  jvp,num=1
```

2)、JVPARGS 增加 -Dfile.encoding=UTF-8，JVPCLASSPATH 增加 judr.jar

```
# 修改配置参数 JVPARGS 和 JVPCLASSPATH
JVPARGS -Dcom.ibm.tools.attach.enable=no;-Dfile.encoding=UTF-8
JVPCLASSPATH
$GBASEDBTDIR/extend/krakatoa/krakatoa.jar:$GBASEDBTDIR/extend/krakatoa/jdbc.jar:$GBASEDBTDIR/udr/
judr.jar
```

重启数据库生效，通过 onstat -g glo 确认已经增加的 jvp。

3，创建相应的 Java UDR 例程

JASCII(s)函数：返回字符对应的 ASCII 码值

```
-- ascii
drop function if exists jascii(lvarchar);
create dba function jascii(lvarchar)
returns int with (not variant)
  external name 'com.gbasedbt.judr.mysql.MysqlFunc.ascii(java.lang.String)'
language java;
```

RANDOM()函数：返回 0-1 之间的随机数

```
-- random 随机函数 (0-1 之间)
drop function if exists random();
create dba function random()
returns float
  external name 'com.gbasedbt.judr.math.Math.randdouble()'
language java;
```

BIN(x)函数：返回数值 x 对应的字符型二进制值

```
-- bin(x)，返回二进制
drop function if exists BIN(int);
```

```
create dba function BIN(x int)
returns lvarchar with (not variant)
  external name 'com.gbasedbt.judr.math.Math.dec2bin(int)'
language java;
```

CONV(x,from,to)函数：进制转换函数，支持 2，8，10，16 进制间转换

```
-- conv 进制转换函数
drop function if exists conv(lvarchar,int,int);
create dba function conv(lvarchar,int,int)
returns lvarchar
with (not variant)
  external name 'com.gbasedbt.judr.math.Math.conv(java.lang.String,int,int)'
language java;
```

IFNULL(str1,str)函数：如果 str1 为 null，返回 str2

```
-- ifnull
drop function if exists ifnull(lvarchar,lvarchar);
create dba function ifnull(lvarchar,lvarchar)
returns lvarchar
with (not variant, handlesnulls)
  external name 'com.gbasedbt.judr.mysql.MysqlFunc.ifnull(java.lang.String,java.lang.String)'
language java;
```

ENCRYPTAES(str, key)函数：AES 加密函数

```
-- encryptAES，指定 key
drop function if exists encryptAES(lvarchar,lvarchar);
create dba function encryptAES(lvarchar,lvarchar)
returns lvarchar with (not variant)
  external name 'com.gbasedbt.judr.crypt.Crypt.encryptAESHex(java.lang.String,java.lang.String)'
language java;

-- 默认 key
drop function if exists encryptAES(lvarchar);
create dba function encryptAES(lvarchar)
returns lvarchar with (not variant)
  external name 'com.gbasedbt.judr.crypt.Crypt.encryptAESHex(java.lang.String)'
language java;
```

DECRYPTAES(str, key)函数：AES 解密函数

```
-- decryptAES (PKCS5)，指定 key
drop function if exists decryptAES(lvarchar,lvarchar);
create dba function decryptAES(lvarchar,lvarchar)
returns lvarchar with (not variant)
```

```
  external name 'com.gbasedbt.judr.crypt.Crypt.decryptAES(java.lang.String,java.lang.String)'
language java;

-- 默认 key
drop function if exists decryptAES(lvarchar);
create dba function decryptAES(lvarchar)
returns lvarchar with (not variant)
  external name 'com.gbasedbt.judr.crypt.Crypt.decryptAES(java.lang.String)'
language java;
```

其他函数，可使用 UDR 也可以使用 Java UDR，视情况使用：

JUUID()函数：兼容 mysql 的 UUID()函数

```
-- uuid
drop function if exists juuid;
create dba function juuid()
returns lvarchar
  external name 'com.gbasedbt.judr.mysql.MysqlFunc.UUID()'
language java;
```

UNIX_TIMESTAMP()函数：返回当前或者指定日期时间的相对于 1970-01-01 08:00:00 的秒数

```
-- unix_timestamp
drop function if exists unix_timestamp();
create dba function unix_timestamp()
returns int with (not variant)
  external name 'com.gbasedbt.judr.mysql.MysqlFunc.unix_timestamp()'
language java;

drop function if exists unix_timestamp(lvarchar);
create dba function unix_timestamp(lvarchar)
returns int
with (not variant)
  external name 'com.gbasedbt.judr.mysql.MysqlFunc.unix_timestamp(java.lang.String)'
language java;
```

FROM_UNIXTIME()函数：返回给定秒数相对于 1970-01-01 08:00:00 的日期时间

```
-- from_unixtime
drop function if exists from_unixtime(int);
create dba function from_unixtime(int)
returns lvarchar
with (not variant)
  external name 'com.gbasedbt.judr.mysql.MysqlFunc.from_unixtime(long)'
language java;
```

GBASE

南大通用数据技术股份有限公司
General Data Technology Co., Ltd.

微博二维码　微信二维码

技术支持热线：400-817-9696