制度管理信息表

制度基本信息项	
制度名称	XX 银行股份有限公司 MYSQL 开发使用规范
制度层级	□基本制度　□一般制度　■操作规范
发文部门	信息科技部
主要业务领域	信息科技
现行版本	2.0 版，2018
现行版本印发日期	2018 年 10 月 XX 日
主要关联制度	无
上次版本	1.0 版，2017
上次版本发文号	
上次印发日期	
密级	秘密
起草人	xx
签发人	xxx
修订信息	（对制度进行修订时填写）
本次修订要点	1、
	2、
	3、
	4、
	5、
历史修订记录	1、第 1 次修订：_____ ，制度名称：_____，文号：_____
	2、第 2 次修订：_____，制度名称：_____，文号：_____
	3、第 3 次修订：_____，制度名称：_____，文号：_____
	4、
	5、
备注	

目 录

XX 银行股份有限公司

MYSQL 开发使用规范

（2.0 版 2018 年）

1. 引言

1.1 目的

本文档旨在明确 MYSQL 数据库技术与应用的规划与设计、安装与部署、运行管理与维护等全生命周期各阶段的主要技术标准和指导原则，便于 MYSQL 数据库应用项目的统一建设和管理，增加 MSQL 数据库技术应用的规范性、性能保障和可维护性。

1.2 预期读者

XX 银行使用 MYSQL，MySQL 等项目相关数据库设计、开发管理和运行维护人员。

2. 设计规范

2.1 数据库设计原则

〈规范〉

[g1] MYSQL 面向的是 OLTP 的应用场景，并不适用于大多数复杂的 OLAP

[g2] 反范式设计：

- 不必强制满足第三范式，禁止使用外键
  - 外键用来保护参照完整性，可在业务端实现

- 适度的冗余设计，减少多表 join 查询，更适应 MPP 架构的横向扩展能力
- 直接基于 I/O 和查询进行优化

[g3] 充分考虑数据库整体安全设计，数据库管理和使用人员权限分离

〈建议〉

[j1] 充分考虑业务逻辑和数据分离，数据库只作为一个保证 ACID 特性的关系数据的持久化存储系统，禁止使用自定义函数、存储过程、触发器和视图

[j2] 充分考虑具体数据对象的访问频度及性能需求，结合主机、存储等需求，做好数据库性能设计

[j3] 充分考虑数据增长模型，决策是否采用 groupshard 模式

[j4] 充分考虑业务数据安全等级，设计合适的备份和恢复策略

2.2 数据库模型设计

2.2.1 基础规范

〈规范〉

[g1] 只使用 InnoDB 存储引擎，避免使用 MyISAM 引擎，二者对比，InnoDB 具有如下特性

- 完整的 ACID 支持
- 崩溃自检恢复
- 行级锁，高并发的保证
- 更能发挥多核 CPU 的性能
- 自带缓存池，更好的利用内存

[g2] 所有表使用统一的字符集，使用 UTF8 或 UTF8MB4 字符集

- UTF8MB4 是 UTF8 的超集
- UTF8 编码只支持 1-3 个字节，只支持 BMP 这部分的 Unicode 编码区，UTF8MB4：一个字符最多能有 4 字节，可以用来存 emoji 表情

[g3] 不在数据库中存储图片、二进制文件等大数据

[g4] 禁止在线上数据库上做数据库压力测试.

[g5] 禁止从测试，开发环境直连数据库

〈建议〉

[j1] 建议所有的表添加注释，除主键外其他字段都有添加注释，推荐采用英文标点，避免出现乱码。

[j2] 提前规划好单表规模，行数和大小

- 单表建议控制在 5000W 以下

[j3] 控制单行字段总长度，合理设置 innodb_page_size(16k)，尽量采用 COMPACT 行格式，避免行溢出。

2.2.2 数据库对象命名

数据库对象命名规范的适用范围为使用 MYSQL 的数据库设计、管理、开发人员，对于 MYSQL 产品自带的系统库表等对象不在本规范约束范围内。数据库对象命名时遵循如下规范：

〈规范〉

[g1] 在索引约束等对象命名时会同时包含表名、字段名等多个名字，此时可以使用缩写，缩写规则和字符数要统一

[g2] 只使用小写字母、数字和下划线的组合,并采用下划线分割

[g3] 库名，表名,字段名 长度不要超过 32 个字符

[g4] 不要使用 SQL（MYSQL）关键字

[g5] 对象名字至少包括：对象类型、父对象名、对象名

下表列出了不同数据库对象的命名规则规范示例：

表 1 数据库对象命名规则示例

数据库	样 式	实 例	说明
数据库（SCHEMA）	db_〈数据库名〉	db_user	表示这个数据库下都是用户相关的数据
表	tbl_〈表名〉	tbl_employees	表示存储雇员信息的表
主键	pk_〈表名〉_〈字段名〉[_	pk_emlo_id_name	表示 tbl_employees 表的 id

	字段名]		和 name 字段共同组成主键,在 MSQL，这里的主键名其实没有意义,系统会忽略这个名字,然后统一命名为：PRIMARY
唯一索引	uidx_<表名>_<字段名>[_字段名]_<编号>	uidx_empl_name_age_1	表示 tbl_employees 表的name 和age 字段创建的第一个索引
普通索引	idx_<表名缩写>_<列名缩写>[_列名缩写]_<编号>	idx_empl_name_age_2	表示 tbl_employees 表的name 和age 字段创建的第二个索引
视图	v_<视图名>	v_female	禁止使用
存储过程	sp_<存储过程名>	sp_add_empl	禁止使用
自定义函数	f_<函数名>	f_count_empl	禁止使用
触发器	trg_<触发器名>	trg_update_empl	禁止使用
临时表	tmp_<表名>	tmp_latecomer	
备份表	bak_<表名>_日期	bak_test_20160727	

〈建议〉

[j1] 对象命名要使用富有意义英文词汇，除约定俗成外，避免使用缩写

2.2.3. 实例配置

MYSQL 实例在申请后需要初始化，初始化过程中有 2 个重要指标需要指定：

- 字符集：指定后续库表默认的字符集。
- innodb_page_size：InnoDB 表文件存储中页的大小。页是 InnoDB 引擎在文件中存储的最小单位，原生 Mysql 的默认值为 16K，MYSQL 默认是 16K,该值的配置从以下几点考虑：

  1） 实例中主要表的单行数据大小

  2） InnoDB 表的 ROW_FORMAT 配置

  3） 是否使用 SSD 盘

在初始化后，还有些参数可以设置：

- 根据业务需要，可通过 SQL MODE 设置 STRICT_ALL_TABLESSTRICT_TRANS_TABLES 标志位开启 STRICT 模式，严格拒绝不符合字段类型定义的数据写入，两个标志位的区别：

  1） STRICT_ALL_TABLES：对于不支持事务的存储引擎，如果同时更新多

行数据，如果不 符合定义的数据不是第一行，则会导致该行前面的行数据更新，改行及后续行数据 不更新的情况

2) STRICT_TRANS_TABLES：对于不支持事务的存储引擎，如果不符合定义的数据是第 一行，则拒绝请求，否则按照普通模式运行

- MYSQL 默认会启动事务自动提交（AUTOCOMMIT），也建议保持开启
- MYSQL 默认的事务隔离级别是 REPEATABLE-READ，请根据需要决定是否调整为 READ-COMMITED（ORACLE 的默认级别）

2.2.4. 数据库

TDSQL 中数据库的概念和 Oracle 不同，可以认为是一个 SCHEMA，类似于一个表的文件夹，方便 DBA 的分类管理。创建数据库时，除了指定数据库名，强烈建议明确指定数据库默认的字符集参数，例如：

〈规范〉

```
CREATE DATABASE `db_user` CHARSET='utf8' COLLATE='utf8_bin';
```

2.2.5. 表

2.2.5.1. 表参数设置

- InnoDB 表的默认 ROW_FORMAT 为 COMPACT，请根据情况指定合适的值：

1) COMPACT：在不产生行溢出的情况下，一行数据存储在一个数据页中，数据读取效率最好

2) DYNAMIC：至少存储在 2 个页中，数据页只存储溢出页的指针，数据存在溢出页中，这种结构的索引存储最集中，索引访问效率较高

3) REDUNDANT：老的格式，不要使用

4) COMPRESSED：会对数据进行压缩后存储，最高的空间利用率，不过会增加 CPU 和内存消耗，增加系统响应时间，降低吞吐量，谨慎使用

- 强烈建议明确指定表默认的字符集参数

- 文件存放路径等物理相关的参数是不允许指定的

2.2.5.2. 表主外键、约束设计

<规范>

[g1] InnoDB 表必须指定主键

[g2] 禁止使用外键

[g3] 不要使用 CHECK 约束，MYSQL（Mysql）虽然不报错，但会忽略，可以使用 ENUM 类型代替

<建议>

[j1] 建议不使用具有实际意义的字段做主键，如果一定要使用，确保该字段具有现实世界的唯一且不变性（例如：银行交易流水号），以尽量避免主键修改

[j2] 组成主键的字段总长度越短，效率越高。整数类型的字段做主键最合适

[j3] 如果存在多个唯一键，考虑最常用的唯一键作为主键

[j4] 建议使用自增字段作为主键

[j5] 字段属性尽量加上 NOT NULL 约束以及默认值，使用 NULL 值会导致如下副作用：

- 含义不明，对很多运算符不管用，增加复杂度。例如 a!=5 是无法匹配到 a 为 NULL 的行
- 很难进行查询优化
- 含 NULL 的复合索引失效

Demo:

* c1 VARCHAR(16) DEFAULT NULL -- 不建议
* c1 VARCHAR(16) DEFAULT NOT NULL -- 不建议
* c1 VARCHAR(16) NOT NULL DEFAULT '' -- 建议
* c1 int(11) NOT NULL DEFAULT 0 -- 建议

2.2.5.3. 表结构设计

<规范>

[g1] 单表 2000W 数据量以下的禁止使用分区表

<建议>

[j1] 将大字段、访问频率低的字段拆分到单独的表中存储,分离冷热数据

[j2] 推荐使用 HASH 进行散表,表名后缀使用十进制数,数字必须从 0 开始

[j3] 按日期时间分表需符合 YYYY[MM][DD][HH]格式,例如 2013071601。年份必须用 4 位数字表示，按日散表 user_20110209，按月散表 user_201102

[j4] 采用合适的分库分表策略。例如千库十表、十库百表等

2.2.6. 字段

2.2.6.1. 字段类型设计

<规范>

[g1] 禁止在数据库中存储明文密码

<建议>

[j1] 将大字段、访问频率低的字段拆分到单独的表中存储，分离冷热数据

[j2] 建议使用 UNSIGNED 存储非负数值

[j3] 建议使用 INT UNSIGNED 存储 IPV4

[j4] 用 DECIMAL 代替 FLOAT 和 DOUBLE 存储精确浮点数。例如与货币、金融相关的数据

[j5] INT 类型固定占用 4 字节存储,例如 INT(4)仅代表显示字符宽度为 4 位,不代表存储长度

[j6] 区分使用 TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT 数据类型。例如取值范围为 0-80 时,使用 TINYINT UNSIGNED

[j7] 建议使用 TINYINT 来代替 ENUM 类型

[j8] 尽可能不使用 TEXT、BLOB 类型

[j9] 使用 VARBINARY 存储大小写敏感的变长字符串或二进制内容

[j10] 使用尽可能小的 VARCHAR 字段。VARCHAR(N)中的 N 表示字符数而非字节数

[j11] 区分使用 DATETIME 和 TIMESTAMP。存储年使用 YEAR 类型；存储日期使用 DATE 类型，存储时间(精确到秒)建议使用 TIMESTAMP 类型

[j12] 所有字段均定义为 NOT NULL

下表是推荐使用的主要数据类型取值范围和存储需求：

表 2 主要数据类型取值范围和存储需求

类型	值范围	存储需求
TINYINT[(M)]	-128 到 127 或 0 到 255	1 个字节
SMALLINT[(M)]	-32768 到32767 或 0 到65535	2 个字节
INT[(M)]	-2147483648 到 214748364 或 0 到4294967295	4 个字节
BIGINT[(M)]	-9223372036854775808 到9223372036854775807 或 0 到 18446744073709551615	8 个字节
DECIMAL[(M[,D])]	整数最大位数（M）为 65，小数位数最大（D）为30	变长
DATE	'1000-01-01' 到 '9999-12-31'	3 个字节
TIME[(<microsecond precision>)]	'-838:59:59.999999'到 '838:59:59.999999'	5 个字节
DATETIME[(microsecond precision)]	'1000-01-01 00:00:00.000000' 到'9999-12-31 23:59:59.999999'	8 个字节
TIMESTAMP[(<microsecond precision)]	'1970-01-01 00:00:01' (UTC) 到'2038-01-19 05:14:07' (UTC)	4 个字节
CHAR[(M)]	0<M<=255	M 的整数倍，和字符集设置有关
VARCHAR[(M)]	0<M<65532/N	N 的取值和实际字符集设置有关
TEXT[(M)]	0<M<65535/N	N 的取值和实际字符集设置有关

2.2.7. 索引

<规范>

[g1] 索引名必须全部使用小写

[g2] 非唯一索引按照“idx_字段名称[_字段名称]_<编号>”进用行命名。例如：

idx_age_name_1

[g3] 唯一索引按照“uidx_字段名称[_字段名称]_<编号>”进用行命名。例如：uidx_age_name_2

[g4] 组合索引包含所有字段名,过长的字段名可以采用缩写形式。例如：idx_age_name_add_3

[g5] 禁止冗余索引，重复索引

[g6] 禁止使用外键

<建议>

[j1] 单张表中索引数量不超过 5 个

[j2] 单个索引中的字段数不超过 5 个

[j3] 唯一键由 3 个以下字段组成,并且字段都是整形时,可使用唯一键作为主键;其他情况下，建议使用自增列或发号器作主键

[j4] 表必须有主键,推荐使用 UNSIGNED 自增列作为主键

[j5] 联表查询时,JOIN 列的数据类型必须相同,并且要建立索引

[j6] 不在低区分度列上建立索引,例如“性别”

[j7] 选择区分度大的列建立索引。组合索引中,区分度大的字段放在最前。

[j8] 不对过长的 VARCHAR 字段建立索引，建议优先考虑前缀索引，或添加 CRC32 或 MD5 伪列并建立索引

[j9] 合理创建联合索引，(a,b,c)相当于(a)、(a,b)、(a,b,c)

[j10] 合理使用覆盖索引减少 IO，避免排序

[j11] MYSQL 的 InnoDB 引擎支持全文索引，目前仅限于英文

2.2.8. 分页设计

分页是应用中最常见的访问模型,我们用下面几种分页方式的实际测试情况来看看如何设计合理的分页模型：

<建议>

```
/*
```

```
 * id 是表 post 的主键
 */
 MySQL> SELECT sql_no_cache * FROM post LIMIT 20000,10;
 10 row in set (0.13 sec)

 MySQL> SELECT sql_no_cache * FROM post LIMIT 80000,10;
 10 rows in set (0.58 sec)

 MySQL> SELECT sql_no_cache id FROM post LIMIT 80000,10;
 10 rows in set (0.02 sec)

 MySQL> SELECT sql_no_cache * FROM post WHERE id>=323423 LIMIT 10;
 10 rows in set (0.01 sec)

 MySQL> SELECT * FROM post WHERE id >= ( SELECT sql_no_cache id FROM
 post LIMIT 80000,1 ) LIMIT 10 ;
10 rows in set (0.02 sec)
```

上面的结果很明显，要尽量避免直接使用 LIMIT m,n 这种分页方式

2.3. 数据库安全设计

数据库用户安全设计原则：

1) 数据库用户权限授权按照最小分配原则
2) 数据库用户要分为管理、应用、维护、备份四类用户

数据库用户权限设计规范：

1) 除 MYSQL 和核心维护人员外，其他用户不能拥有 SUPER 权限账号
2) 避免使用简单密码
3) MYSQL 的权限最新支持到字段级别，权限的主体对象要按最小原则控制
4) 开发、测试和生产环境中用户权限设置要保持一致

严格禁止在数据库中存储任何形式的密码明文。

3. 开发规范

3.1. SQL 编码规范

<建议>

[j1] 每行不要超过 80 个字符，超过就换行缩进

[j2] 使用两个空格来缩进代码

[j3] 关键词要大写，比如：SELECT

[[j4] 常数符号要大写，比如：NULL

[j5] 合理使用注释，建表语句前要对表的用途进行详细注释

[j6] 每个字段后使用 COMMNET 子句添加字段的注释

[j7] 建表语句最后面使用 COMMENT 子句添加对表的一句话注释

[j8] 去掉多余的扩号，例如：

```
((a AND b) AND c OR (((a AND b) AND (c AND d))))
```

应该优化为：

```
(a AND b AND c) OR (a AND b AND c AND d)
```

[j9] 去掉重叠的条件，例如

```
(B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
```

应该优化为：

```
B=5 OR B=6
```

3.2. SQL 语句规范

3.2.1. 索引和分区

<建议>

[j1] 尽量不要尝试使用 USE INDEX 和 IGNORE INDEX 进行索引的选择.

[j2] 查询条件尽量使用索引

[j3] 注意字段类型，避免类型转换（隐式转换）。类型转换除了会增加 CPU 消耗，如果转换失败，还会导致索引失效

[j4] 对于复合索引，查询条件必须包含所有前缀字段才管用，例如索引(a,b,c)，查询条件必须是 a 或者 a、b 或者 a、b、c 才能使用到索引

3.2.2 SELECT 列和 WHERE 条件

<规范>

[g1] 禁止让数据库做算术运算和函数运算，交给应用层来做，例如：

```
SELECT a FROM tbl WHERE id*10=100;
```

[g2] SELECT 只获取必要的字段，禁止使用 SELECT *

[g3] 禁止使用负向查询,例如 not in、!=、not like

[g4] LIKE 子句的条件中，%不要是第一个字符，禁止使用%前导查询,例如：like “%abc”，无法利用到索引，尽量靠后。更复杂的需求考虑使用全文索引

[g5] 使用 IN 代替 OR。SQL 语句中 IN 包含的值不应过多，应少于 1000 个

[g6] OR 条件大于 3 个：

- 不同字段的，使用 UNION ALL 代替
- 相同字段的，用 IN 代替

[g7] 禁止使用 order by rand( )

[g8] 禁止隐式转换。数值类型禁止加引号；字符串类型必须加引号

〈建议〉

[j1] 尽量使用UNION ALL，而不是UNION。UNION 会做去重和排序

[j2] WHERE 子句使用的原则：尽量使用索引，尽量简单，尽量匹配更少的行

[j3] WHERE 子句中多使用等值操作符，少使用非等值操作符，不等值操作符通常会导致索引失效。

[j4] 就算有索引，WHERE 子句匹配的行数不要超过表的 30%，否则效率还是很低，InnoDB 引擎还很有可能直接放弃使用索引，采用全表扫描

[j5] 尽量使用 WHERE 子句代替 HAVING 子句，例如：

```
SELECT id,COUNT(*) FROM tbl GROUP BY id HAVING age>=30;
```

应该替换为：

```
SELECT id,COUNT(*) FROM tbl WHERE age>30 GROUP BY id;
```

[j6] 一个表的 ORDER BY 和 GROUP BY 的组合都不应该超过 3 种，否则从业务逻辑考虑进行优化或者分成多张表

[j7] 如果不需要排序，GROUP BY 子句写成：GROUP BY column ORDER BY NULL

[j8] WHERE 子句尽量使用主键

[j9] InnoDB 表尽量避免使用类似 COUNT(*)的全表扫描查询，如果有高频的 count(*)类的操作，从设计上考虑另外用一张表存这个计数值

[j10] 避免使用 JOIN 和子查询，必要时推荐用 JOIN

3.2.3 多表 JOIN

〈建议〉

[j1] 多表 join 时，各表的顺序按照各表在 WHERE 子句条件下返回的行数从小到大排列，行数最小的在最左边

[j2] 避免大表间的 JOIN，一般表的记录数不超过 10W 条的情况下，才建议使用 JOIN

3.2.4 DML 语句

<规范>

[g1] INSERT 语句必须指定字段列表，禁止使用 INSERT INTO TABLE( )

[g2] 禁止在 UPDATE 语句中，讲“,”写成 AND，非常危险，例如：

```
UPDATE tbl SET fid=fid+1000, gid=gid+1000 WHERE id > 2;
```

如果写成

```
UPDATE tbl SET fid=fid+1000 AND gid=gid+1000 WHERE id > 2;
```

此时“fid+1000 AND gid=gid+1000”将作为值赋给 fid，且没有 Warning

[g3] 清空一张表用 TRUNCATE，而不是 DELETE

[g4] 禁止单条 SQL 语句同时更新多个表

<建议>

[j1] UPDATE、DELETE 等语句尽量带上 WHERE 子句，且使用索引字段，最好使用主键

[j2] 使用 INSERT……ON DUPLICATE KEY update(INSERT IGNORE)来避免不必要的查询

[j3] INSERT、UPDATE、DELETE 语句中不要使用不确定值的函数，例如：RAND( )和 NOW( )

[j4] 多条 INSERT 语句要合并成一条批量提交，一次不要超过 500 行数据

3.2.5. 其它

<规范>

[g1] 禁止在主，从库上执行后台管理和统计类功能的 QUERY，必要时申请统计类从库

<建议>

[j1] 使用 prepared statement，可以提升性能并避免 SQL 注入

[j2] 减少与数据库交互次数，尽量采用批量 SQL 语句

[j3] 拆分复杂 SQL 为多个小 SQL，避免大事务

[j4] 获取大量数据时,建议分批次获取数据,每次获取数据少于 2000 条，结果集应小于 1M,

[j5] SQL 中避免出现 now( )、rand( )、sysdate( )、current_user( )等不确定结果的函数

[j6] 建议使用合理的分页方式以提高分页效率

[j7] 程序应有捕获 SQL 异常的处理机制，必要时通过 rollback 显式回滚

[j8] 重要 SQL 必须被索引：update、delete 的 where 条件列、order by、group by、distinct 字段、多表 join 字段

[j9] 使用 EXPLAIN 判断 SQL 语句是否合理使用索引,尽量避免 extra 列出现:Using File Sort、Using Temporary

[j10] 合并 DDL,如果要对表进行 DDL 操作，尽量将对一张表的 DDL 在一条 SQL 语句完成，例如：要给表 t 增加一个字段 b，然后给已有字段 a 建立索引，可以用下面一条语句完成：

```
ALTER TABLE t ADD COLUMN b VARCHAR(10), ADD INDEX idx_a(a);
```

3.3 行为规范

<规范>

[g1] 表结构变更必须通知 DBA 进行审核

[g2] 禁止有 super 权限的应用程序账号存在

[g3] 禁止有 DDL、DCL 权限的应用程序账号存在

[g4] 批量导入、导出数据必须通过 DBA 审核，并在执行过程中观察服务

[g5] 批量更新，如 UPDATE、DELETE 操作，必须 DBA 进行审核，并在执行过程中观察服务

[g6] 产品出现非数据库导致的故障时，如被攻击，必须及时通 DBA，便于维护服务稳定

[g6] 业务部门程序出现 BUG 等影响数据库服务的问题，必须及时通知 DBA，便于维护服务稳定

[[g7] 出现业务部门人为误操作导致数据丢失，需要恢复数据的，必须第一时间通知 DBA，并提供准确时间点、,误操作语句等重要线索

[g9] 不要在 MySQL 数据库中存放业务逻辑

<建议>

[j1] 重要项目的数据库方案选型和设计建议提前通知 DBA 参与

[j2] 提交线上建表改表需求,必须详细注明涉及到的所有 SQL 语句(包括 INSERT、DELETE、UPDATE)，便于 DBA 进行行审核和优化

[j3] 业务部门推广活动或上线新功能,建议提前通知 DBA 进行服务和访问量评估，并留出必要时间以便 DBA 完成扩容

3.4 DB connector

我们建议用 mysql 官方的 connector,下载地址：

http://dev.mysql.com/downloads/connector/j/

3.5.MYSQL NOSHARD 与 GROUP SHARD 版本选择

当满足以下条件时，可以考虑使用 GROUP SHARD：

- 表中数据会持续增长，在可预见的时间内，超过单机最大存储量
- 表的读写量很大，超过了单机最大吞吐量
- 充分考虑了分库分表会导致的在表设计和使用上的各种限制

关于GROUP SHARD 版本在使用上的限制，请参考：DCDB for MYSQL 开发指南.pdf

MYSQL GROUP SHARD 版本上线时，我们会更新 GROUP SHARD 版本的 DB 开发和设计规范，目前上线的是 NOSHARD 版本

4. 用户与权限规范

4.1. 用户角色和权限介绍

MYSQL 的使用者应该分为集群管理者、数据库实例管理者和数据库使用者：

- 集群管理者主要通过运维管理平台完成以下工作：
  - 管理整个集群的资源池，负责资源的上下架和合理的分配
  - 为实例管理者分配数据库实例
  - 协助实例管理者进行实例管理
  - 负责管理所有实例的冷备和 BINLOG 备份数据
  - 关注集群和实例的运行指标，保证集群和各实例的正常运行
- 数据库实例管理者主要通过数据库管理平台完成以下工作：
  - 提交新建、扩容、删除实例的申请
  - 负责维护实例的个性化参数
  - 负责管理数据库实例的账号和密码
  - 关注实例的运行指标，进行数据库性能管理
  - 通过查看慢查询分析、错误日志、慢查询日志等信息，进行数据库的性能优化分析

数据库使用者通过 SQL 客户端连接数据库，提交 SQL 语句，存取数据

集群管理者,数据库实例管理者会 MYSQL 运维规范中详细介绍，本开发规范中，主要为读者介绍 3 种数据库使用者角色(应用写用户，只读用户，应用发布用户)。

- 应用写用户权限：INSERT，DELETE，UPDATE，SELECT
- 只读用户权限：SELECT
- 应用发布用户权限：CREATE，ALTER，DROP，INSERT，DELETE，UPDATE，SELECT

数据库使用者角色权限粒度默认分配到库，特殊权限，需单独申请，如 FILE 权限，等等。

5. 建库建表示例

5.1 建库

```
-- 1. create database demo

CREATE DATABASE IF NOT EXISTS db_sakila DEFAULT CHARACTER SET utf8 COLLATE utf8_bin;



USE db_sakila;
```

(-- 绿色部分是我们标注的注释，是 demo 中解释说明用途，开发过程中可以不用写。)

5.2 建表

```
-- 2. create table demo

-- Table "tbl_actor"

CREATE TABLE tbl_actor (

  actor_id                SMALLINT              UNSIGNED NOT NULL AUTO_INCREMENT,

  first_name              VARCHAR(45)           NOT NULL DEFAULT '' COMMENT 'The Actor first name',

  last_name               VARCHAR(45)           NOT NULL DEFAULT '' COMMENT 'The Actor last name',

  last_update             TIMESTAMP             NOT NULL DEFAULT    CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT 'Last update time',

                          --  A timestamp,The range is '1970-01-01 00:00:01.000000' UTC to '2038-01-19 03:14:07.999999'

  PRIMARY KEY pk_actor_id(actor_id),      -- 主键

  KEY idx_actor_last_name_1 (last_name)   -- last_name 上建立非唯一索引
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT 'Actors table' ;



-- Table "tbl_film"

CREATE TABLE tbl_film (

  film_id                    INT                 UNSIGNED NOT NULL AUTO_INCREMENT,

  title                                  VARCHAR(50)              NOT NULL DEFAULT ''
COMMENT 'The film title',

  release_year                       YEAR                          NOT NULL DEFAULT '0000'
COMMENT 'The film release year',

                             -- 'YYYY',rang is from 1901 to 2155

  release_date                     DATE                        NOT NULL DEFAULT '1000-01-01'
COMMENT 'The film release date',

                             -- 'YYYY-MM-DD',A date. The supported range is '1000-01-01' to
'9999-12-31'

  release_datetime                 DATETIME                    NOT NULL DEFAULT '1000-01-01
00:00:00.000000'   COMMENT 'The film release datetime',

                             --   'YYYY-MM-DD   HH:MM:SS[.fraction]',A   date   and   time
combination,   The   supported   range   is   '1000-01-01   00:00:00.000000'   to   '9999-12-31
23:59:59.999999'

  language_id                      TINYINT                   UNSIGNED NOT NULL DEFAULT 0
COMMENT 'REFERENCES language (language_id)',

                             -- 8-bit unsigned int [0, 255]

  rental_rate                           DECIMAL(4,2)             NOT NULL DEFAULT 4.99
COMMENT 'rental rate',

  replacement_cost                  DECIMAL(5,2)              NOT NULL DEFAULT 19.99
COMMENT 'replacement cost',

  rating                             ENUM('G','PG','PG-13','R','NC-17')        DEFAULT 'G'
COMMENT 'rating',
```

```
  special_features              SET('Trailers','Commentaries','Deleted Scenes','Behind the
Scenes') DEFAULT NULL,

                                    -- Can take zero or more values from a SET

                                    -- But only one value from ENUM

  film_md5                          CHAR(32)                    NOT NULL DEFAULT ''
COMMENT 'The film MD5',

  last_update                TIMESTAMP            NOT NULL DEFAULT    CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP COMMENT 'Last update time',

  PRIMARY KEY pk_film_id(film_id),

  KEY idx_film_title_1(title),          -- title 上建立非唯一索引, 索引序号是 1

  UNIQUE KEY uidx_film_md5_2(film_md5)  -- film_md5 上建立唯一索引, 索引序号是 2

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT 'Film table' ;



-- Table "tbl_film_actor"

CREATE TABLE tbl_film_actor (

  id                    INT                 UNSIGNED NOT NULL AUTO_INCREMENT,

  actor_id                SMALLINT              UNSIGNED NOT NULL    COMMENT 'REFERENCES
tbl_actor (actor_id)',

                        -- 字段类型注意和 tbl_actor 表的 actor_id 一致。

  film_id                 INT                   UNSIGNED NOT NULL    COMMENT 'REFERENCES
tbl_film (film_id)',

                        -- 字段类型注意和 tbl_film 表的 film_id 一致。

  PRIMARY KEY film_actor_id(id) ,

  KEY idx_fa_actor_film_id_1(actor_id, film_id)             -- actor_id, film_id 上建立联
合索引

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT 'Film actor object-relation table' ;
```

6. 附则

本规范由信息科技部负责解释、修订。

本规范自发布之日起执行。