文档编号	V1.1
文档负责人	曾志伟
文档名称	卡行天下数据库设计开发规范
文件状态	内部传阅

卡行天下数据库设计开发规范

审核人	
重要性	高
紧迫性	高
拟制人	曾志伟
提交日期	2015/10/21
最近一次修改日期	2018/04/19
需求变更控制时间点	

卡行天下供应链管理有限公司 KXTX Supply Chain Management Company Limited
上海市长宁区通协路 558 号汤泉国际大厦 B 幢 5F ZIP 200335
5F, Building B, Tangqiang, Tongxie Road No.558, SHANGHAI
4001-816-816

文档属性		
文件状态： [ ] 草　　稿 [√] 正式发布 []正在修改	文件名称：	卡行天下数据库设计开发规范
	当前版本：	1.1
	作　　者：	曾志伟
	文件密级：	[　]普通　　[√]秘密　　[　]绝密
文档参与人		
文档保存位置：SVN 路径 参考文档：文件名+SVN 路径		
文档版本		

SS 更新日期	内容说明	变更人	版本号
2015.10.09	卡行天下数据库设计开发规范	曾志伟	0.1
2016.04.27	卡行天下数据库设计开发规范	曾志伟	1.0
2017.06.20	卡行天下数据库设计开发规范	曾志伟	1.1
2017.12.19	卡行天下数据库设计开发规范	曾志伟	1.2
2017.04.17	卡行天下数据库设计开发规范	曾志伟	1.3

目录

一、表设计公共字段

1.1 业务表必含字段

【强制】所有与业务相关有 CRUD 操作的表，都需要含有以下 4 个字段。

序号	列名	数据类型	是否为空	默认值	注释
1	id	unsigned int unsigned bigint	not null		主键，新表设置尽量不用 uuid
2	create_time	datetime	not null		创建时间（框架上更新）
3	latest_time	timestamp	not null	CURRENT_TIMESTAMP	最后更新时间 ON UPDATE CURRENT_TIMESTAMP
4	is_delete	tinyint	not null	0	是否删除，0.未删除，1.已删除

1.2 可选字段

【推荐】以下为可选字段，注意默认值的统一。

序号	列名	数据类型	是否为空	默认值	注释
1	is_enable	tinyint	not null	1	是否启用，1 启用，0 禁用，默认 1
2	creater_id	int	not null		创建人
3	updater_id	int	not null		修改人

二、数据库对象命名规则

2.1 命名规范概述

2.1.1【强制】采用有意义的命名

命名只能使用英文字母、数字和下划线三类字符组合。尽量使用英文单词，不要使用拼音，多个单词组成时，中间以下划线 _ 分割。

尽可能地将命名描述的清楚些，但不要写的太长，具体尺度就在你的把握中。

2.1.2【强制】英文单词统一使用单数形式

无论表名或者字段名，统一使用英文的单数形式，列名使用单数形式大家都容易理解，但表名很多人习惯使用复数，但实际中使用单数形式更简洁，语法上也是单数表示一个类别。

2.1.3【强制】虽然 MySQL 在服务器端已经设置为不区分大小写，但创建修改表时，表名字段名都统一使用小写。

2.1.4【推荐】尽量不要超过 30 个字符，也即表名、字段、函数、过程、触发器、序列、视图等名称的长度均不要超过 30 个字符。

【禁止】:

- 禁止使用 MySQL 系统保留字
- 禁止对象名使用非字母开头
- 禁止使用除字母、数字、下画线外的其它字符
- 禁止使用下划线_开始或者结束
- 禁止使用拼音与英文混合命名，禁止直接使用中文命名

2.2 命名格式

2.2.1 【强制】数据库名

库名与应用名称尽量一致，相比表名和字段名，数据库名应该更加简洁化，使用字母、数字、下划线组合，不要使用横杠。

2.2.2 【强制】表名

表名前缀：表前缀用项目名称首字母缩写，如 tms，单词之间用下划线分开，单词都用单数形式，表名前缀避免使用 sys_做前缀。

表名称 = 表名前缀_ +表内容标识
如 tms 项目的表都以 tms 作为前缀，
项目表名：tms_order, tms_waybill, tms_config

【禁止】:
表名前缀禁止使用 sys_开头，不带前缀表名禁止使用 MySQL 数据库保留字。

2.2.3 【强制】关联表命名

多对多的两张表 A 和 B 设计时，需另外设计一张关联表。
A_rel_B，其中 A 和 B 的都去除前缀，如 tms_order 和 tms_waybill, 关联表为 order_rel_waybill。
当 A 和 B 表名太长时，除了去除前缀，A 和 B 表名应该进一步使用缩写，或者取表名的核心段，尽量使名字总长度不要超过 30 个字符。

关联表的主键一般是是相关实体表主键形成的复合主键，是多列。

2.2.4【强制】临时表 tmp_

临时表以 tmp_开头，后接功能描述。
尽量加上保留时间，如 tmp_tabname_keepto_20171220;

2.2.5【强制】字段名

- 命名应反映该段数据内容，多个单词时用下划线分隔;
- 非空字段，请加上 default 值;
- 同一字段在多个表中出现时，一定要确保其命名和数据类型保持一致;

2.2.6【强制】判断、标识性字段

凡表示"是否"一类的判断字段，都使用 is_开头。
如: 是否删除 is_delete, 是否付款 is_paid，是否启用 is_enable 等，类型都使用 tinyint，使用 0，1 表示，如 is_delete 中 0 表示未删除，1 表示已经删除。

状态、类别一类的字段，数据类型使用 tinyint，枚举数字从 1 开始，如定单状态 order_status、付款状态等，并将所有数字表示的状态或类别一一列举，如 1 表示什么，2 表示什么，3 表示什么…

【禁止】:
枚举数字不要使用数字 0，统一从 1 开始，如 1,2,3….

2.2.7【强制】主键命名
所有表都要建立主键，临时表、外部表、流水表，日志表有时可以不用主键。
主键名统一使用 id，这样一方面简短，另一方面，代码上凡是看到单独一个 id，都表示是当前表的主键。

2.2.8【强制】外键命名
数据库层不维护主外键关系，但是在字段命名上，要体现外键的业务逻辑。
外键命名使用： 主表名_id，主表名去掉前缀。
比如 tms_order 表的 id，当在 tms_waybill 上引用时，使用 order_id 作为外键名，同时在注释上说明这个字段是来自哪一张表的外键。

备注：当一张表的多个外键来自同一表的主键，请特别对待处理，如 update_id, create_id 都来自用户表 id。

2.2.9【推荐】非外键一类的号码 _no _sn
为了和外键区别开来，非外键一类的号码，都使用_no 或者_sn，如订单序列号使用 order_no，这样看到_id 一般是外键，看到_no 一般是序列号。

2.2.10 【强制】视图 vw_
vw_table_name_+内容标识，各部分以下划线_分割，多个表时，尽量将所有表都加入，若所有表名太多加起来太长，则取其中的主要的表+内容标识。

【禁止】:
- 视图创建语句中禁止使用 for update
- OLTP 系统的视图中禁止使用 order by
- 避免使用嵌套视图
- 需要分库分表的系统禁止使用 view，trigger,procedure 等，aliyun 上的 DRDS 也无权创建

2.2.11 索引
参考后面章节 ***三、索引的命名规范***

2.2.12 【强制】存储过程 sp_
存储过程以 sp_开头，后接功能描述。

2.2.13【强制】 函数 fn_
函数命名以 fn_开头，后接函数的功能描述。

2.2.14 【强制】包和包体 pkg_
包以 pkg_开头，后接功能描述。

2.2.15 【强制】系列 seq_

SEQUENCE 命名：seq_+table_name；表名连前缀也保留，即使用表的完整全名。
如 tms_order 表对应的系列，seq_tms_order

2.2.16 【强制】触发器 tr_
触发名= tr_ + 触发标识 + 相应的表名。
如：tr_ins_client，tr_del_client

触发类型	触发标识
Insert	ins
delete	del
update	upd

2.2.17 【强制】物化视图 mv_
MySQL 本身不支持物化视图，如果通过其它方式创建物化视图，统一使用 mv_开头，后接功能描述。

【禁止】:
分库分表的系统，禁止使用 view, function,procedure, trigger, package 等，aliyun 的 drds 上无权创建这些对象。

2.3 注释规范

2.3.1 【强制】所有表和字段都要写注释
每个表，每个字段都要有注释，说明其含义，注释优先使用中文，注意简洁，同时应描述清晰。

2.3.2 【推荐】注释尽可能详细、全面
创建每一数据对象前，应具体描述该对象的功能和用途。
传入参数的含义应该有所说明。如果取值范围确定，也应该一并说明。取值有特定含义的变量，如使用 tinyint 表示类型时，应给出每个值的含义，如定单状态 order_status、1 表示什么，2 表示什么，3 表示什么...

2.3.3 【强制】外键和冗余字段
外键，请在注释上标明来自哪张主表。
对于反范式的冗余字段请特别说明其维护方法，以避免时间久了出现遗漏或者数据不一致的情况。

2.4 索引的命名

以下为索引命名规范，在查看执行计划时，会显示索引名称，规范的索引名称有助于更快捷定位性能问题。

2.4.1【强制】主键命名：PK_表名
PK_开头，后面加表名，格式为：PK_表名 如果表名太长，请使用使用表名缩写
如:`alter table tabname add primary key pk_tabname(colname);`

表名缩写：索引命名时，对于很长的表名，使用表名缩写
由于尽量保持所有数据库对象名长度不超过30个字符，所以在索引中，缩写一般取每个名字段的第一个字母，如表ga_capital_assets_change缩写为gcac。如果表名很短，就写表的全名。

2.4.2【强制】外键索引 FK_所在表名_父表名

FK_开头，后面加字段所在表名，再加主表名，格式为：FK_所在表名_父表名， 如果表名太长，父表子表都使用缩写

注意： 由于我们不在数据库上维护索引，所以，外键的创建语法，使用普通索引的创建方法，命名规则也可以使用普通索引的命名规则，详细请参考 3.4。

2.4.3 【强制】唯一索引：UK_表名_列名

UK_开头，后面加表名，再加列名，格式为：UK_表名缩写_列名，表名太长请使用表名缩写。

如：`create unique index uk_tabname_colname on tabname(colname);`

2.4.4 【强制】普通索引命名：IDX_表名_列名

IDX_开头，后面加表名，再加列名，格式为：IDX_表名_列名，同样，表名太长请使用表名缩写。

如：要在 ga_capital_assets_change 表的 external_no 字段上建索引，则命名为

`create index idx_gcac_external_no on ga_capital_assets_change(external_no);`

2.4.5 【强制】复合索引 IDX_表名_列名缩写 1_列名缩写 2_列名缩写 3 如：idx_tabname_c1_c2_c3

IDX_开头，后面加表名，再加列名，格式为：IDX_表名_列名,复合索引创建方法和普通索引一样，主要是名字太长，一般表名列名都要缩写，如：alter table tabname add index idx_tabname_c1_c2_c3 (c1, c2, c3)

有多字段联合索引时，WHERE 中过滤条件的字段顺序无需和索引一致，但如果有排序、分组就必须一致了。

2.4.6 【强制】其它类索引 IDX_表名_字段名

IDX_开头，其它索引，如 hash 索引，函数索引等，都使用前面说的普通索引命名方法。

【注意】:

【强制】超过 20 个长度的字符串，请考虑创建前缀索引。前缀索引的长度，一般略大于本列的平均长度，例如 colname 的平均长度为 15，那么前缀取稍大一点（如 16）就可以。或者根据选择性和实际访问量，原因请见*附录二*。

【禁止】:

禁止为超大数据类型创建索引，如 varchar 长度超过 200，考虑使用全文索引或者搜索引擎。

上面是索引的命名规则，对于什么情况下应该创建索引，请参考***附录一、适合及不适合建索引的情况***。

三、字段类型的选择

3.1 字段类型选择原则

- 【强制】尽量使用最小的数据类型，更小的数据类型，会占用更少的磁盘、内存和 CPU 缓存，处理时需要的 CPU 周期也更少，例如，存储’KXTX’时，使用 varchar(4)和 varchar(100)，占用磁盘空间是一样的，但 varchar(100)效率更低，使用时会耗用更大的内存，这对于排序和临时表操作影响比较大。

- 【强制】优先选择整型，而不是字符，整型比字符的代价更低，字符集在做比较时的校对规则比整型复杂，比如，用整型存储 IP 使用 int unsigned，手机号使用 bigint，都不要使用 varchar 类型。

- 【强制】整型类型，默认一般都加上 unsigned 属性，表示不允许负值，这样可以使正数的上限提高一倍。如果确认为非负数，必须加上 UNSIGNED。

- 【强制】小数类型为 decimal，禁止使用 float 和 double。
  说明： float 和 double 在存储的时候，存在精度损失的问题，很可能在值的比较时，得到不正确的结果。如果存储的数据范围超过 decimal 的范围，建议将数据拆成整数和小数分开存储。

- 【强制】如果存储的字符串长度几乎相等，使用 CHAR 定长字符串类型。varchar 类型会需要额外使用 1-2 字节记录字符串长度，如果列长度<=255 字节，则使用 1 字节表示，列长度>255 时，使用 2 字节。所以前面的 varchar(1)占用 2 字节。

- 【强制】用 INT UNSIGNED 存储 IPV4 地址，用 INET_ATON()将 IP 转换为数字、INET_NTOA()将数字转换为 IP，没必要使用 CHAR(15)来存储。

- 【强制】varchar 是可变长字符串，不预先分配存储空间，长度不要超过 5000，如果存储长度大于此值，定义字段类型为 TEXT，独立出来一张表，用主键来对应，避免影响其它字段索引效率。

- 【推荐】字段允许适当冗余，以提高性能，但是必须考虑数据同步的情况。冗余字段应遵循：
  1）不是频繁修改的字段。
  2）不是 varchar 超长字段，更不能是 text 字段。

- 【强制】主键采用整型，加上 UNSIGNED 属性，同时也加上 AUTO_INCREMENT。
- 【强制】在对小数进行精确计算时使用 decimal，如金额等账务数据。
- 【强制】对图片视频等大文件，不要存在数据库，而是存一条文件路径，实际文件存放文件系统上。
- 【强制】如无备注，所有的布尔值字段，如 is_hot、is_enable，都必须设置一个默认值，并设为 0；
- 【强制】所有数字类型字段，无特殊情况，都必须设置为非空，并设一个默认值 0；
- 【强制】所有字符类型字段，无特殊情况都必须设置为非空，并设一个默认值’’ 注意中间没有空格。

【禁止】:

- 禁止修改字段名,字段名一旦在数据库创建，禁止修改，有歧议时可以添加注释等方式。
- 小数类型为 decimal，禁止使用 float 和 double。
- 禁止修改字段属性和默认值等，如 int 修改为 varchar，null 修改为 not null,默认值 1 修改为 0 等，如特殊情况业务一定需要，需找部门总监以上负责人特别审核通过。
- 禁止将图片、excel、mp3、视频等大文件直接存放在数据库中。
- 禁止使用字符串作为主键，对于完全随机的字条串，例如 md5、uuid 产生的字符串，这些函数生成的和成的新值会任意分布在很大的空间内，并且 uuid 生成的索引是 int 索引大小的 27 倍，除了更耗空间，也更加消耗内存，导致临时表、insert 以及 select 语句变得更慢。

关于常用字段类型取值范围，请参考 ***附录二、常用类型所占字节以及取值范围***

3.2 Null 与空值的使用

【强制】所有字段定义中，默认都加上 NOT NULL 约束，除非业务要求必须为 NULL，但是一般很少有什么场景下必须要在数据库中存储 NULL 值。

【推荐】对于已经创建好的表，没有必要将字段全修改为 not null，但至少要在建索引的字段上设置 not null。

Null 与空值的区别，见*附录三、Null 与空值的区别*

四、SQL 语句编写规范

4.1 主要 SQL 规范

【强制】SQL 关键字使用大写，一些编辑器中可以设置关键词自动大写;
【强制】条件语句中，数据库字段放置在左边;
【强制】不等于统一使用<>;
【强制】在 OLTP 系统中，不要写复杂的 SQL 语句，关联表不能超过 3 张;
【推荐】拆分大的 DELETE 或 INSERT 语句，比如每次删除 1000 条;
【推荐】同一字段，将 or 改为 in(c1, … cn)，当 cn 很大时，or 会慢很多，注意控制 in 后面的个数;
【推荐】不同字段，将 or 改为 union/union all;
【推荐】减少子查询的使用，用 join 或其他更高效、可读性更好的方式替代;
【推荐】表的别名建议使用 a, b, c 格式表示;
【推荐】查询中用 Order By 排序时，优先使用主键列，索引列。排序操作会降低数据库的性能，应谨慎;
【推荐】 in 操作能避免则避免，若实在避免不了，需要仔细评估 in 后边的集合元素数量，控制在 1000 个之内。

【强制】不要使用 count(列名)或 count(常量)来替代 count(*)， count(*)就是 SQL92 定义的标准统计行数的语法，跟数据库无关，跟 NULL 和非 NULL 无关。
说明： count(*)会统计值为 NULL 的行，而 count(列名)不会统计此列为 NULL 值的行。
【强制】 count(distinct col) 计算该列除 NULL 之外的不重复数量。注意 count(distinct col1, col2) 如果其中一列全为 NULL，那么即使另一列有不同的值，也返回为 0。
【强制】禁止使用存储过程，存储过程难以调试和扩展，更没有移植性。

【推荐】 TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少，但 TRUNCATE 无事务且不触发 trigger，有可能造成事故，故不建议在开发代码中使用此语句。

【禁止】:

- 禁止使用 select * from …. 相应地禁止 insert 中不指明字段;
- 页面搜索禁止左模糊或者全模糊，如果需要请走搜索引擎来解决。
- 条件语句中，数据库字段不允许出现在函数或者运算表达式中;
- 禁止程序中使用明文帐号和密码;
- 禁止开发人员添加 hint，使用 hint 必须经过 DBA 审核;
- 禁止使用 order by rand();
- 禁止多表跨库联合查询;
- 应用中禁止使用悲观锁;
- 应用代码禁止隐性数据类型转换;
- 应用中禁止 DDL 语句（权限上也会控制）;

- 禁止超过三个表以上的 join。需要 join 的字段，数据类型保持绝对一致；多表关联查询时，保证被关联的字段需要有索引。

4.2 将关联查询拆分

【推荐】尽量将多表关联查询拆分，尤其是对于数据量大，需要做分库分表的项目。将关联查询拆分成单表查询后，有以下优点，所以 OLTP 生产环境，请大家优先使用分解关联查询：

- 让缓存更高效，应用程序可以方便地缓存单表查询的结果;
- 查询分解后，单个查询可以减少锁竞争;
- 减少冗余数据的查询;
- 相当于在应用层实现了 hash join，而不是 MySQL 自带的 nest loop join;
- 最重要的一点，应用层做拆分，对于我们正在规划的分库分表，除了更好的查询性能，还会有更好的兼容性。

【禁止】:

在 OLTP 系统中，尤其是高并发系统，不要写复杂的 SQL 语句，禁用超过 3 张表以上的关联。

MySQL5.6 对子查询的优化有很大的改善，但仍然尽量避免使用子查询。

4.3 数据订正

【推荐】原则上禁止直接跑 SQL 修改数据库，稳定成熟的系统，应该很少有数据订正；

【推荐】如果一定要直接修改，常用的修改，请做成界面，授权客服修改；

【强制】如果还是一定要直接运行 SQL 修改，那么请严格按照以下流程，本流程所有环节都使用邮件。

流程详见另一份文件 *卡行天下生产环境数据订正流程 20160422.pdf*

以下【强制】

■数据订正时，必须先在 session 级设置 set autocommit=0，然后将所做的修改手动 commit;

■多库操作时，表名前加上库名;

■删除、修改记录时，必须有 where 唯一条件，如主键限制等;

■当有主从表时，要先删除从表记录，再删除主表记录;

■大的数据 insert、delete、update，拆分成多条，并且避开业务高峰执行;

■update 前，先 select 检查一下，单个字段的 update，先将原字段值保存下来作为备份;

■复杂的或者没把握的订正，请将开发人员叫到身边，一起执行;

■数据订正前一定要有备份，备份形式可以灵活，比如直接将 update 的字段 select 出来，或者将表 copy 到另一个名为 test01 的库，或者使用 RDS 的临时库。

■**【禁止】:** 数据库写入、修改等操作，尤其是批量写入时，禁止使用无线连接，必须使用有线网络。只有无线网络时，可以使用跳板机。

五、索引的使用规范

5.1. 【强制】业务上具有唯一特性的字段，即使是组合字段，也必须建成唯一索引。
说明： 不要以为唯一索引影响了 insert 速度，这个速度损耗可以忽略，但提高查找速度是明显的；另外，即使在应用层做了非常完善的校验和控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。

5.2. 【强制】超过三个表禁止 join。需要 join 的字段，数据类型保持绝对一致；多表关联查询时，保证被关联的字段需要有索引。
说明： 即使双表 join 也要注意表索引、 SQL 性能，过滤条件尽可能集中到驱动表。

5.3. 【强制】在 varchar 字段上建立索引时，必须指定索引长度，没必要对全字段建立索引，根据实际文本区分度决定索引长度。
说明： 索引的长度与区分度是一对矛盾体，一般对字符串类型数据，长度为 20 的索引，区分度会高达 90%以上，可以使用 count(distinct left(列名, 索引长度))/count(*)的区分度来确定。

5.4. 【强制】页面搜索严禁左模糊或者全模糊，如果需要请走搜索引擎来解决。
说明： 索引文件具有 B-Tree 的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索引。

5.5. 【推荐】如果有 order by 的场景，请注意利用索引的有序性。 order by 最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现 file_sort 的情况，影响查询性能。
正例： where a=? and b=? order by c; 索引： a_b_c
反例： 索引中有范围查找，那么索引有序性无法利用，如： WHERE a>10 ORDER BY b; 索引 a_b 无法排序。

5.6. 【推荐】利用覆盖索引来进行查询操作，来避免回表操作。
说明： 如果一本书需要知道第 11 章是什么标题，会翻开第 11 章对应的那一页吗？目录浏览一下就好，这个目录就是起到覆盖索引的作用。
正例： IDB 能够建立索引的种类：主键索引、唯一索引、普通索引，而覆盖索引是一种查询的一种效果，用 explain 的结果， extra 列会出现： using index.

5.7. 【推荐】利用延迟关联或者子查询优化超多分页场景。
说明： MySQL 并不是跳过 offset 行，而是取 offset+N 行，然后返回放弃前 offset 行，返回行，那当 offset 特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行 SQL 改写。
正例： 先快速定位需要获取的 id 段，然后再关联：
SELECT a.* FROM 表 1 a, (select id from 表 1 where 条件 LIMIT 100000,20) b where a.id=b.id

5.8. 【推荐】 SQL 性能优化的目标：至少要达到 range 级别， 要求是 ref 级别， 如果是 consts 最好。
说明：
1）consts 单表中最多只有一个匹配行（主键或者唯一索引），在优化阶段即可读取到数据。
2）ref 指的是使用普通的索引。（normal index）
3）range 对索引进范围检索。
反例： explain 表的结果，type=index，索引物理文件全扫描，速度非常慢，这个 index 级别比较 range 还低，与全表扫描是小巫见大巫。

5.9. 【推荐】建组合索引的时候，区分度最高的在最左边。
正例： 如果 where a=? and b=? ， a 列的几乎接近于唯一值，那么只需要单建 idx_a 索引即可。
说明： 存在非等号和等号混合判断条件时，在建索引时，请把等号条件的列前置。如： where a>? and b=? 那么即使 a 的区分度更高，也必须把 b 放在索引的最前列。

5.10.【参考】创建索引时避免有如下极端误解：
1）误认为一个查询就需要建一个索引。
2）误认为索引会消耗空间、严重拖慢更新和新增速度。
3）误认为唯一索引一律需要在应用层通过“ 先查后插” 方式解决。

六、范式与冗余

6.1 数据库设计前三范式

第一范式就是原子性，字段不可再分割。即数据库表中的所有字段值都是不可分解的原子值。第一范式是对关系模式的基本要求，不满足第一范式的数据库就不是关系数据库。

第二范式就是完全依赖，没有部分依赖。数据库表中不存在非关键字段对任一候选键的部分依赖，也即所有非关键字段都完全依赖于任意一组候选关键字。

第三范式就是没有传递依赖。在第二范式的基础上，数据表中如果不存在非关键字段对任一候选关键字段的传递依赖则符合第三范式
备注：对于设计范式，一般没有特别严格的要求，完全按照规范化设计的系统几乎是不可能的，可以合理存在一些反范式设计。

6.2 反范式设计

严格遵守范式设计出来的数据库，虽然思路清晰，结构合理。但范式越高，设计出来的表可能越多，关系可能越复杂，性能却不一定会很好，因为表一多，就增加了关联性。特别是在高可用的 OLTP 数据库中，这一点表现得很明显。

所以在数据库设计时，应当保持一定的冗余数据来提高程序效率，增加冗余字段时，请在注释上写明，该冗余字段来自哪张主表的字段，以便程序后期维护。

需要注意的是，冗余的负面影响是数据的一致性，当数据库设计有冗余时，为了保持数据一致性，会给编程增加一定的工作量。当性能与范式发生冲突时，通常考虑性能优先。

6.3 明确应用业务的性质（OLTP or OLAP）

在为业务做数据库设计时，应该首先要明确分析出正在设计的应用程序是什么类型的，即它是 “事务处理

型”（Transactional）还是“分析型”（Analytical）。本规范大多是针对 OLTP 业务类型的。

对于事务处理型业务(OLTP)，这种类型的应用程序的 DML 操作更多，即更关注数据的增删查改（CRUD，Creating/Reading/Updating/Deleting）。

对于分析型业务(OLAP)，这种类型的应用程序查询操作更多，更关注数据分析、报表、趋势预测等等功能。这一类的数据库的插入和更新操作相对来说是比较少的，它们主要的目的是更加快速地查询、分析数据。

七、字符集与校对规则

7.1 字符集 UTF-8

字符集是指一种从二进制编码到某类字符符号的映射。UTF-8 包含全世界所有国家需要用到的字符，是一种多字节编码，它存储一个字符会使用变长的字节数（1-3 个字节）。

数据库全都统一使用 UTF-8 字符集。使用 UTF-8 时汉字占 3 个字节，英文和数字占 1 个字节。

所有项目的字符存储与表示，默认均使用 utf-8 编码，使用字符计数方法时请注意：
SELECT LENGTH("卡行天下"); 返回为 12
SELECT CHARACTER_LENGTH("卡行天下"); 返回为 4
如果要使用表情，那么使用 utfmb4 来进行存储，注意它与 utf-8 编码。

7.2 校对规则 utf8_general_ci

校对是指一组用于某个字符集的排序规则，校对规则都使用 utf8_general_ci，其中的 ci 是 case insensitive，即大小写不敏感。

MySQL 数据库会在服务器端统一将字符集设置为 UTF-8，校对规则设置为 utf8_general_ci，请在客户端也统一使用这个设置，否则会发生隐式转换，不设置的话默认就是 UTF8 和 utf8_general_ci。

八、存储引擎与事务隔离级别

8.1 存储引擎 innodb

字符集是指一种从二进制编码到某类字符符号的映射，UTF-8 包含全世界所有国家需要用到的字符，是一种多字节编码，它存储一个字符会使用变长的字节数（1-3 个字节）。使用 UTF-8 时汉字占 3 个字节，英文和数字占 1 个字节。

所有项目都使用 innodb 存储引擎，innodb 适用于几乎 99%的 MySQL 应用场景，MySQL5.5 起默认存储引擎改为 innodb，MySQL5.7 起，连系统表也使用 innodb 了。

8.2 事务隔离模式 read commited

MySQL 的默认隔离级别是 Repeatable Read（可重读）它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。所有项目我们统一使用 Read Committed（读取提交内容）。

Read repeatable 和 Read committed 最大区别

- read repeatable:开启一个事务,读一个数据,而后再次读,这 2 次读的数据是一致的(行级锁且是锁间隙);
- read committed:开启一个事务,读一个数据,而后再次读,这 2 次可能不一样的,因为在这 2 次读之间可能有其他事务更改这个数据,这也就是读提交,每次读到的数据都是已经提交的(行级锁,不锁间隙)

修改事务隔离模式：

- 全局修改 my.cnf

```
transaction-isolation = READ-COMMITTED
```

- 当前 session 修改

```
set session transaction isolation level READ COMMITTED;
```

九、DRDS 开发规范

9.1 确定拆分字段

分库分表，首先要从字段中选出拆分键，即分库/分表用的字段。拆分键目前暂时只支持单个字段。对代码层，要求查询时一定要带上拆分键，或者至少 95%以上的查询要带上拆分键。拆分键有以下 2 种：

分库键：DRDS 根据分库键的值将数据水平拆分到后端的每一个 RDS 分库里。键值相同的数据，一定会位于同一个 RDS 数据库里。

分表键：每一张逻辑表都可以定义自己的分表键，键值相同的数据，一定会位于同一个 RDS 数据表里。

货运圈目前用到的常见的业务拆分规则，有以下三种：

使用 id 加时间，id+time

```
如:     dbpartition BY HASH(user_id,id2)
    tbpartition BY MM(create_time) tbpartitions 12
```

只用 id 拆分

```
如:     dbpartition BY UNI_HASH(`trade_entity_id`)
    tbpartition BY UNI_HASH(`trade_entity_id`) tbpartitions 8
```

只用时间拆分

```
如:     dbpartition BY YYYYMM_OPT(`create_time`)12*4   48
```

```
    tbpartition BY YYYYMM_OPT(`create_time`) tbpartitions 3
```

【禁止】禁止拆分键的值是否可以为 NULL

目前 DRDS 中，拆分键值为 NULL 是允许的，但要注意的是：

1．拆分键值为 NULL 的记录，都会被路由到同样的一个分片

2．如果 NULL 值的记录占比非常高，那这个分区就会出现热点

9.2 分布式事务

由于 DRDS 底层是多台数据库，所以如果有跨库的事务，要申请打开分布式事务功能，这个功能需要找 DBA 申请开通。

手工处理事务的典型 SQL 语句步骤如下：

```
set autocommit=false //开启事务
select last_txc_xid() //注册一个 GTS 事务
insert/update/delete 等业务 SQL
commit 或者 rollback //全局提交或回滚
set autocommit=true //恢复自动提交
```

Spring 事务管理器来处理事务的典型代码如下：

```
@Transactional
public void update(JdbcTemplate jdbcTemplate) {
  //开启一个 GTS 分布式事务
  jdbcTemplate.execute("select last_txc_xid()");
  // insert/update/delete 等业务 SQL
}
```

什么情况？ 一个事务有多条 begin select / update /insert …..单条 insert 全不需要

多个 DRDS 实例的 GTS 接入（略）

https://help.aliyun.com/document_detail/53318.html

使用 GTS 会在以下两个层面对性能产生影响：

1)事务处理链路上会增加对 GTS 服务器的 RPC 调用，在系统负载良好的情况下，RT 会上升 2-3ms

2)由于 GTS 会消耗 CPU，因此系统容量会下降 20-30%

9.3 广播表的使用

广播表是指将这个表复制到每个分库上，在分库上通过同步机制实现数据一致，有秒级延迟。这样做的好处是可以将 JOIN 操作下推到底层的 RDS（MySQL），来避免跨库 JOIN。由于需要将表复制到每一个库，如果有 24 个库，那么广播表就会复制成 24 份，所以一般要求广播表尽量小，一般尽量在万行级以下。

```
CREATE TABLE t(
  id int,   name varchar(30),
  primary key(id) ) ENGINE=InnoDB DEFAULT CHARSET=utf8 BROADCAST;
```

查看某个广播表每个分库中的数据量：

```
    /!TDDL:SCAN*/SELECT COUNT(1) FROM table_name
```

DRDS 表连接优化–广播表驱动

以下的三张表 JOIN（其中表 t1 是广播表，t2 和 t3 是拆分表），查询执行时间约 15 秒：

```
SELECT t1.name FROM t2 i
JOIN t3 b ON i.sellerId = b.sellerId JOIN t1 a ON b.province = a.id
WHERE a.id < 110107 LIMIT 0, 10;    执行时间 15 秒
```

如果调整一下 JOIN 的顺序，将广播表放在最左边作为 JOIN 驱动表，则整个 JOIN 查询在 DRDS 中会被下推为单库 JOIN 查询：

```
SELECT t1.name FROM t1 a
JOIN t3 b ON b.province = a.id JOIN t2 i ON i.sellerId = b.sellerId
WHERE a.id < 110107 LIMIT 0, 10; 执行时间小于 0.1 秒
```

9.4 DRDS 的限制

使用 DRDS 有一些注意事项需要重点注意，代码层需要一定修改，如上面的分布式事务，广播表等。

以下是 DRDS 使用时的一些注意事项和限制：

- insert 操作无影响，select、update 操作必须带上拆分键。如不能带拆分键，带上主键、唯一索引键，或者其它过滤性很强的索引也可以
- DRDS 不支持 store procedure、package、view、trigger 等，创建时会报错
- 多表关联，如果是拆分表+非拆分表关联，非拆分表一定要设成广播表，否则会存在跨库查询
- 使用 DRDS 的 OLTP 系统，对关联表要求更严，禁止三张表以上的关联
- DRDS 对多表关联的要求，虽然 OLAP 会松一些，但请注意广播表的影响
- 为了性能考虑，DRDS 中，UNION 默认为 UNION ALL，如需去重，请显式使用 UNION DISTINCT
- DRDS 层不会对 SQL 结果进行任何缓存
- DRDS 中，拆分键的值不允许修改，直接修改会报错（可以采用 insert+delete 的变通方法）

- DRDS 中，使用 GTS（TXC），分布式事务的默认超时时间是 10 分钟
- 执行 INSERT_SELECT 的时候，关掉事务（避免因跨库操作而报错）
- 禁止拆分键为 null 值，详情见 9.1
- 更新删除全表时，请加 HINT/*TDDL:FORBID_EXECUTE_DML_ALL=false*/delete from t

更多 DRDS 使用方法和规范，请参考另一单独文档 *MySQL 使用 DRDS 分库分表.pdf*

附录一、适合及不适合建索引的情况

a）适合建索引的情况

- where 后面使用频繁的搜索列，但不一定是所有 where 后面的列都要建索引；
- 虽然数据库上没有维护主外关系，但如果该字段相当于某张表的外键，一般要在该字段加索引；
- 两表 join 时，所匹配 on 和 where 的字段应建立合适的索引；
- group by, order by 后面字段，要考虑建索引，但不一定是全部建索引；
- 索引要建在选择性高，重复记录少的字段上；
- 索引尽量建在小的字段上，如果碰到大的字段(超过 20 字符)，考虑使用前缀索引。

- 前缀索引；

超过 20 个长度的字符串，请考虑创建前缀索引，因为较小的索引涉及的磁盘 I/O 较少，更为重要的是，对于较短的键值，索引高速缓存中的块能容纳更多的键值。前缀索引的长度，一般略大于本列的平均长度，例如 colname 的平均长度为 23，那么前缀取稍大一点（如 24）就可以，或者根据选择性和实际访问量。前缀索引兼顾索引大小和查询速度，但是其缺点是不能用于 ORDER BY 和 GROUP BY 操作，也不能用于覆盖索引。 创建方法如下：

```
create index idx_tabname_colname on tabname(colname(24)) ;
alter table tabname add key(colname(24) );
```

选择性的计算方法如下：

```
SELECT count(DISTINCT(left(colname,5)))/count(*) AS Selectivity FROM tabname;
```

- 哈希索引

对于一些比较长的的字段，比如存储 url 的字段，这类数据一般比较长，这时可以使用 crc32 做哈希，再在这个 hash 值上建索引，性能会大幅提升。

如当字段 url=http://www.kxtx.cn/Innovation.aspx 时，这时可以取 crc32("http://www.kxtx.cn/Innovation.aspx")= 110365867，不要在 url 上直接建索引，而是在哈希值 110365867 上建索引。

- 复合索引

如果 where 后面有多个列，那么建一个复合索引，会比建多个单列索引更高效。复合索引很重要的问题是如何安排列的顺序，比如 where 后面用到 c1, c2 这两个字段，那么索引的顺序是(c1,c2)还是(c2,c1)呢，正确的做法是，重复值越少的越放前面，比如一个列 95%的值都不重复，那么一般可以将这个列放最前面。另外，复合索引的字段数尽量不要超过 3 个，一旦超过，要慎重考虑必要性。

使用复合索引时，第一列一定要用上。建立索引 key(c1,c2,c3)相当于建立了 key(c1), key(c,c2)和 key(c1,c2,c3)三个索引。比如 key(c1,c2,c3)，那么只支持 c1； c1,c2； c1,c2,c3 三种组合进行查找。

多字段联合索引时，WHERE 中过滤条件的字段顺序无需和索引一致，但如果有排序、分组则就必须一致了。

- 尽量保证 distinct, group by, order by 后面所带的列名相同，尽量不要使用包含多列的排序。

- 在 group by, order by 列上建索

如 `select … from tabname order by c2;`
在 order by 列上建索引可以实现利用索引进行 order by 优化。

- WHERE + ORDER BY 情况下创建索引，例如下面 sql 建立联合索引(c1,cn)来实现 order by 优化

如：`select c1,c2,… from tabname where c1=xxx order by cn;`

b）不适合建索引的情况

- 唯一性太差的字段不适合建立索引，如标识性字段 is_delete，或者表示状态类型等一类的字段；
- 只出现在 select 中，不会出现在 where 或者 order by 条件中的字段不要建索引；
- 一般不要出现，同一字段既是单字段索引，又出现在复合索引中；
- 同一张表，不要建过多复合索引；
- '%xxx'或者'%xxx%'一类的模糊查询，不要创建普通索引，可以考虑建全文索引，但'xxx%'可以用到索引；
- 不要过度索引，索引会影响 DML(insert、delete、update)的性能，对于 DML 操作频繁的表，一般一张表不要超过 6 个索引，具体还需视表的大小和业务情况适当调整；
- 尽量避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序。

附录二、常用类型所占字节以及取值范围

存储类型：整形

tinyint	1 字节	-128 ~ 127 (或 0 ~ 255)
smallint	2 字节	-2^15 (-32,768) 到 2^15-1 (32,767)
mediaint	3 字节	-2^23(-8388608) 到 2^23-1 (8388607)
int	4 字节	-2^31 (-2,147,483,648) 到 2^31-1 (2,147,483,647) (或 0-42.9 亿)
bigint	8 字节	-2^63 (-9223372036854775808) 到 2^63-1 (9223372036854775807)

整型类型指定宽度一般对应用没有意义，int(1),int(10),int(20)对于存储和计算来说是相同的。

存储类型：FLOAT，DOUBLE，DECIMAL（NUMERIC）

占用大小：FLOAT 是 4 字节 0 到 23 的单精度，DOUBLE 是 8 字节 24 到 53 的双精度。

区别：1、FLOAT 和 DOUBLE 支持标准浮点运算进行近似计算。

2、DECIMAL 进行 DECIMAL 运算，CPU 并不支持对它进行直接计算。浮点运算会快一点，因为计算直接在 CPU 上进行。

3、DECIMAL 只是一个存储格式，在计算时会被转换为 DOUBLE 类型。

4、DECIMAL(18,9)使用 9 个字节，小数点前 4 个字节，小数点 1 个字节，小数点后 4 个字节。

5、DECIMAL 需要额外的空间和开销，只有对小数进行精确计算的时候才使用它，如保存金融数据。

6、DECIMAL(M,D)在 MySQL5.6 中，M 的范围是 1-65，D 是 0-30.

存储类型：字符

固定长度的字符串类型采用 CHAR，非固定长度的字串类型采用 VARCHAR，char 最大长度是 255 字节，varchar 最大长度为 65535 字节（注意是字节不是字符）。

在 utf8 字符集下，char/varchar 在表中，英文和数字占 1 字节，汉字占 3 字节。

在 utf8 字符集下，char/varchar 在索引中，不管是英文、数字还是汉字，全占三字节，比如使用 uuid 时索

引的长度是 36*3+2=110，是使用 int 大小的 110/4= 27 倍。这也是为何在建索引时，当长度超过 20 时，就要考虑使用前缀索引了。

如无特殊需要，尽量避免使用 text, clob, long 等类型，text, clob 处理更复杂，性能比 varchar 更差，有些场合还会有一些使用限制。Innodb 会将过长的 varchar 存储为 blob.
varchar 在磁盘上存储数据是变长的，但在内在和临时表中会占设计和全部长度，如使用 varchar(100)存储 hello 时，在表中是占用 5 字节，但在内存和临时表中，占用的是 100 字节。

附录三、Null 与空值的区别

- MySQL 中，null 是未知的，且占用空间的。null 使得索引、索引统计和值都更加复杂，并且影响优化器的判断。
- 空值('')是不占用空间的，注意空值的''之间是没有空格。
- 在进行 count()统计某列的记录数的时候，如果采用的 NULL 值，会被系统自动忽略掉，但是空值是会进行统计到其中的。
- 判断 null 使用 is null 或者 is not null，但判断空字符使用 =''或者 <>''来进行处理。
- 对于 timestamp 数据类型，如果插入 NULL 值，则出现的值是当前系统时间。插入空值，则会出现 '0000-00-00 00:00:00' 。
- 对于已经创建好的表，普通的列将 null 修改为 not null 带来的性能提升比较小，所以调优时没有必要特意一一查找并 null 修改 not null。
- 对于已经创建好的表，如果计划在列上创建索引，那么尽量修改为 not null，并且使用 0 或者一个特殊值或者空值''。
- 对于新表通常情况也全都指定为 not null，并指定 default ''或者其它默认值。

附录四、常见命名及缩写

a）常用表名/列名

user	-	用户
goods	-	商品、产品等一切可交易网站的物品都用此命名
goods_gallery	-	物品的相册
goods_cate	-	物品的分类，除了单独作为表名，其他地方分类单词一律用缩写 cate
article	-	文章、新闻、帮助中心等以文章形式出现的，一般都用此命名
cart	-	购物车
feedback	-	用户反馈
order	-	订单
site_nav	-	包括页头和页尾导航
site_config	-	系统配置表
admin	-	后台用户
role	-	后台用户角色

access - 后台操作权限，相当于 action
role_access - 后台角色对应的权限
affix - 附件

category - 分类
attr - 属性
status - 状态
mode - 形式（模式）
add_time - 添加时间、上架时间等
lastest_time - 最后操作时间，如登录、修改记录
expire_time - 过期时间
name - 商品名称、商家名称等，不要跟 title 混用，title 只用于文章标题、职称等
price - 价格
desc - 描述、简介，比如 goods_desc
details - 详情、文章内容等
sort_id - 排序
telephone - 座机号码
mobile - 手机号码
phone - 当不区分手机和座机时，请用 phone 命名
address - 地址，单独出现不要用 addr 缩写，组合出现时需用缩写，比如 mac 地址，mac_addr
zipcode - 邮编
region - 地区，大的区域，比如记录杭州市、温州市等
area - 区域，小的，比如上城区，江干区等
avg_cost - 人均消费
amount - 金额
audit - 审核
source - 来源
total - 汇总
phase - 阶段
archive - 归档
evaluate - 评估

b）常用缩写

img	-	image	
cate	-	category	分类
attr	-	attribute	属性
type	-	type	类型
addr	-	address	
tel	-	telephone	
zip	-	zipcode	
src	-	source	
proj	-	project	
ver	-	version	

cur	–	current	
tot	–	total	
fin	–	finance	财务
supp	–	supplier	供应商
amt	–	amount	金额
def	–	define	
dict	–	dictionary	
org	–	organization	
reg	–	registration	
capt	–	capital	
invt	–	investment	
tmpl	–	templet	模板
tmp	–	temporary	临时
rept	–	report	
prod	–	product	
exec	–	execute	
bdgt	–	budget	预算
appr	–	approval	批复，核准
app	–	application	
info	–	information	