MySQL数据库备份优化&数据架构设计

---

1. 线上数据备份恢复策略实施

1.1. 备份流程的设计

- 备份工具
- 备份方式

1、考虑数据量：做备份工具的选型

- 数据量较小的情况下：mysqldump逻辑备份，导出的是SQL
- 如果我们数据量非常大：xtrabackup

2、考虑我们的时间点补偿

- 从上一个全量备份时点到现在这个阶段的数据
- Binlog：statement，**row**，mixed

3、做我们的备份方案

- 全量备份 crontab -e （cron语法）定时执行备份脚本，时间选择在业务量小的时点（记录我们的备份的position）
- 增量：binlog（statement和mixed模式，我们的SQL都是线性且可执行的）
- statement和mixed的增量适用于正常情况下数据库无法打开使用或数据文件损坏

- row特别适合单表数据异常恢复
- 一定要确保线上不会出现直接执行的SQL

1.2. 数据恢复流程

案例1：statement，mixed

student（id,score）：4点100行数据，第二天8点的时候更新一条score，所有人score全变成90

- mysqldump：可以去到文件里找到student表把insert全部拿出来
- xtrabackup：先要恢复到一个新数据库
- 把4-8点间所有涉及这个student表的DML除select外，还最后我出错的这个SQL外，所有SQL全部拿出来
- 按照这个顺序：4点student所有数据执行->执行Binlog里的SQL

案例2：row

- 找到position，通过mysqlbinlog导出变更的数据，只能用代码去调整他的记录变更为恢复的SQL
- 如果数据库不可用需要恢复
    - 文件转移到新数据库：物理备份
    - 如果数据文件无法启动
    - mysqldump全量+row模式下的增量（调整起来就非常费劲）

在备份的基础上，如果条件允许一定要做主从HA

- 主机：row
- 从机：statement

1.3. statement模式下数据恢复

作业：自己drop database一次，并通过binlog进行恢复

恢复的过程也记录进binlog了

是否需要记录?

2. MySQL索引优化

- SQL及索引：高质量的SQL，避免索引失效
- 数据库表结构：范式，至少要符合3NF
- 系统配置MySQL，Linux
- 硬件

2.1. Explain执行计划分析

explain的用法

```
explain select * from employee where age=40 and name='张飞'
```

explain的作用

- 查看表的读取顺序
- 读取操作类型
- 哪些索引可用
- 表之间关联
- 每张表有哪些索引被优化器执行

type

- system
- const
- eq_ref
- ref
- range
- index
- ALL

> 查询的效果从上到下越来越差

2.2. 索引命中策略分析

- 最左匹配原则
- 在索引字段上加入函数：不走索引
- is null/is not null：不走索引
- 覆盖索引：key-value都在索引里，如果select columns直接使用的是索引列就直接使用覆盖索引
- 只要索引条件和or挨着：就用不上
- != / <> ： 是否能应用索引

2.3. 索引分析总结

优势：

1、提高查询速度

2、表连接的时候，加速连接

3、保证数据唯一：唯一索引

劣势

1、修改和增加数据时会提升开销

2、索引还会占用物理空间

3、在进行大量的insert或update、delete时，速度会变慢

适合建立索引

1、数据差异化较大

2、频繁查询的列，where条件里经常用到的

3、常用的表关联字段

4、查询中统计或分组的字段

不适合的

1、记录值特别少

2、值变化特别小，重复率高

3、经常增删改的表

2.4. 数据库出现问题后如何死而不僵

```
mysql> show processlist;
mysql> kill pid;
```

3. 数据库架构设计

做架构到底是在做什么?

抽象能力

抽象-->具象

- 逻辑设计：
  - 1、具体内容：设计数据库的一个逻辑结构，与具体的DBMS无关，主要反映业务逻辑
  - 2、设计步骤：用关系模型
  - 3、使用工具来模型化：E-R图
    - 矩形：实体对象 1:m，n:m，1:1
    - 椭圆：属性
    - 线：关系的连接
    - 菱形：关系
  - 4、实体关系模型
    - 通过表格实现：字段名，类型，长度，约束
    - 实体的实例化和泛化
  - 5、至少满足3NF
- 物理设计
  - 对具体数据库进行选型：oracle，mysql
  - 表的字段及存储结构
- 实际工作中：都是并行的

3.1. 数据库命名规范

- 所有数据库对象名称：小写加下花线分割
  - MySQL对象名称在默认情况下是大小写敏感
  - MySQL的对像其实都是一个文件，而linux文件名是大小写敏感
  - Dbname / dbname，MyTable / mytable
  - 开发非常麻烦

- 所有MySQL数据库对象名称禁止使用MySQL保留关键字
    - 一定要提前准备一份对应版本的关键字表
    - 建表的时候没问题，但SQL查询就挂了``
- 所有的数据库对象名称：见名知义，但最长不要超过32个字符（不要中英文混合）
- 所有临时表命名：tmp_tablename_20191215
- 所有的备份表：bak_tablename_20191215
- 索引：idx_ pk_
- 所有存储类型相同的列名以及长度必须保持一致
    - order：product_title 50
    - erp_instock: product_title 50

3.2. 数据库设计规范

- 正常情况下建议使用innoDB，v5.6版本后默认都是innoDB
- 字符集
    - UTF-8
    - 统一字符集避免乱码
    - UTF-8的字符集是一个汉字3个字节：varchar(255) UTF-8 255*3=765字节
- 加入注释
- 控制单表的数据量大小：行
    - 对于日志数据，进行归档
    - 对于业务数据进行分库分表
- 分区表谨慎使用
- 控制表宽度
    - 虽然表没有行限制，但列最多4096
    - 如果列多了，占用内存和I/O会非常大
- 禁止在表中建立预留字段：varchar_column，order_second_no，remark，memo
    - varchar类型
    - 违背上面的命名规则
    - 时间久了，不看业务代码，完全是魔鬼字段
- 禁止在数据库里存放图片、文件、二进制文件
    - 如果要用blob、Text存大文件，select colums....
    - 如何避免select * ，外键表单独放单文件
- 禁止对线上环境进行压力测试
    - 会产生大量的垃圾数据和日志文件
- 禁止从开发环境、测试环境连接生产数据库

3.3. 数据库索引设计规范

- 单张表索引数量建议不超过5个，如果列多可以适当增加
    - 索引过多：SQL在进行优化器评估的时候会有更大的开销
    - 绝对不允许给表的每一列都建立索引
- 每个innodb表都必须有一个主键，innoDB表就是一个索引组织表

    - 表数据的实际存储顺序只能有一种，innoDB是按照主键进行存放的
    - 如果没有主键，mysql会优先选择第一个非空唯一索引来做主键
    - 如果上面这个没有，mysql会自动生成一个36个字节的主键，但性能不好
    - 不能使用更新频繁的列和联合索引做主键，主键不断变，数据的存放顺序就会不断变化
    - 不要使用UUID、MD5、HASH等做主键，不能保证这些值是按顺序增长的。如果生成较小的字符串就会导致不断变化数据存储的位置，影响I/O性能
- 要在哪些列上建立索引：没有最好只有最适合
    - explain
    - where后
    - join的连接lie
    - 筛选项最大的放在索引做左侧
- 避免建立冗余和重复索引
- 对于频繁查询的数据列，优先考虑使用覆盖索引
- 尽量避免加入外键约束
    - 因为外键写入的时候会降低存储效率
    - 但要给这些关联字段加索引

3.4. 数据库字段设计规范

- 优先选择符合存储需要的最小数据类型
    - INT来存放时间戳
    - varchar(255) '长袖衬衫'
- 避免使用TEXT，BLOB数据类型
    - 如果非要使用可以单独拉出来做关联表
    - 这两个类型上没有默认值
- 避免使用ENUM数据类型
    - 修改则需要使用ALTER语句
- 尽可能把所有列定义为NOT NULL
    - 如果为NULL，索引需要额外的空间来保存
- 日期格式尽量不要用字符串保存
    - 不能用日期函数进行计算和比较
    - 用字符串占用的空间更多

1970-01-01 00:00:00-2038-01-19 03:14:07

int来保存：4294967295

- 财务相关的数据用Decimal类型来进行计算

3.5. 数据库SQL开发规范

- 在程序中使用PrepareStatement，#{}
    - 降低词法和语法分析器的重复执行
    - 防止SQL注入
- 合理和充分的利用表上的索引
    - 避免前后%

    - 使用left join或not exists来优化not in（not in无法使用索引）
- 程序连接不同数据库使用不同的账号，禁止跨库操作
    - 应用A---B 应用D---C：跨库访问最好调用业务层
    - 如果账号被注入，也只注入一个库
- 禁止使用select *（但是依旧这么做）
- 禁止使用不含列名的insert into tableName values('','','');
- 避免使用子查询，可以把子查询优化为join操作
    - 子查询的结果集无法使用索引
    - 子查询会产生临时表操作，如果查询量大则会严重影响效率
- 避免使用join关联太多表
    - 大查询拆小查询，由我们的程序来去做关联和合并
    - 进行表数据冗余
    - 有一定的转换
- 减少同数据库的交互次数
- 使用in代替or，in能用索引，or用不上
- 禁止用order by rand()
- where中不要对列进行函数计数：列无法使用索引了
- UNION ALL和UNION
    - 如果我们的数据明显不重复，就使用UNION ALL

程序=数据结构+算法

3.6. 数据库操作行为规范

- 大批量的数据操作会严重造成数据延迟
    - 数据分批执行
    - v5.7 format_binlog：row
- 对大表结构的操作会导致锁表
    - 对于大表的操作：pt-online-schema-change （PERCONA）
    - 原理
        - 创建新表结构
        - 复制旧表数据到新表
        - 在原表上加入触发器确保数据同步
        - 所有操作进行完毕后对原表进入一个很短的时间锁
        - 把原表名进行修改，再改新表名
        - 删除原表名
        - 原子性操作分割进行了
- 禁止对普通用户授予super权限
- 对于程序遵循最小权限原则