游戏数据库 TcaplusDB

最佳实践

产品文档

腾讯云

文档目录

最佳实践
表结构设计

最近更新时间：2021-11-25 15:29:16

数据库设计在整个软件开发的起到的举足轻重的作用，本文将介绍数据库设计应对的原则和方案。

表结构定义准则

- 字段名、表名不建议过长，推荐在32B内，不包含特殊字符。
- 字段名、表名命名规范化，尽量表达实际含义，避免过度简写。
- 注意选择适合的数据类型，避免丢失精度。
- 主键和分表因子的选择具有高度离散化，便于负载均衡和扩缩容处理。
- 索引并不是越多越好，要根据查询有针对性的创建，避免和主键定义一致。
- 业务中 IP 地址字段推荐使用 int 类型。
- 时间类型尽量选择 long 类型，存储秒。
- 逻辑上相关的字段需要保留操作的原子性，建议合并到一张表里。
- 如果性能很关键，可以进行适当的冗余数据设计。
- 主键和分表因子的选择具有高度离散化，便于扩容处理。
- 主键的可读性强，不建议采用二进制类型，便于查阅跟进问题。
- 主键字段大小不建议太大，尽可能使用较短的主键，有利于加快查询速度。
- 字段大小按照需要使用，不建议定义较大，实际使用较小的情况发生。
- 所有表和字段都需要添加注释。

游戏业务相关的设计原则

- 具有生成全局唯一的 ID 的需求，建议采用 increase 操作实现。
- 架构设计上避免数据库（DB）过载，如增加排队机制。
- 具有相关性的数据内容建议放在单个表里，规避出现数据不一致现象。
- 表的轻重需要分离，不建议所有的功能放在一个表中，独立的功能需要考虑独立的表。
- 表被频繁使用，且单表记录较大时，需要考虑设计简要表，避免直接获取原始数据表导致 DB 负载增加。
- 大厅聊天用共享内存即可，单局聊天实时 push，不建议存储 DB。如要支持离线私聊建议使用 DB。
- 善于使用 DB 提供的数组数据结构，如，历史战绩、邮件、举报记录等，即能保证数据淘汰也能支持按照插入顺序的 TopN 操作。
- 排行榜设计时，如果能使用排序组件则直接使用排序组件；如果需要 gameserver 实现，建议排行结果异步落地到 DB。

- 游戏中比较边缘且比较耗时的操作，建议采用单独的进程处理，规避影响 gameserver 主逻辑。
- 游戏业务开放周期长，逻辑复杂，在开发过程中，容易出现逻辑数据结构的变更，从可扩展性和易维护的角度考虑，建议游戏数据表设计时，一些易变数据结构设计作为数据表的 blob 来存储，序列化后再存储于数据库中，以避免因数据结构的变化而频繁修改数据库表。

TDR 表定义

- 主键字段，需要较高的离散度，便于 gameserver 发送请求时分发到多个接入层节点。
- 表定义时，建议索引键不要与主键完全一致，如果二者相同会消耗网络、磁盘资源。
- 普通字段定义数组时需要增加 refer 属性（count 是定义大小，refer 是实际使用大小），便于 count 大小以后扩展，并且减少数据的网络传输和磁盘占用。
- 普通字段推荐多采用一级字段，减少字段间嵌套，字段嵌套类型控制在3层以内。
- 不推荐主键字段采用 binary 类型，不利于排查问题。
- 单个表定义的索引最多8个，推荐2个 - 3个，请按照实际情况设置索引，索引定义过多反而会降低整体性能。

Protocol Buffers 表定义

- 主键字段需要较高的离散度，便于 gameserver 发送请求时分发到多个接入层节点。
- 支持定义嵌套的结构体类型 `非主键` 字段，嵌套深度太深将会影响数据访问性能。

不良设计示例

- 设计与需求不符
  修改量比较大，特别是项目临近上线阶段，修改成本更高。
- 性能低下
  含有大数据量的表之间的关联过多；没有合理的字段设计来用于查询而造成的 SQL 查询语句很复杂；对于大数据量的表没有采用有效的手段去处理；滥用视图等。
- 数据完整性丧失
  含有主外键关系的表之间关联字段的设计方式不合理，造成更新与删除操作后，程序容易出错或不完善；使用了已经删除或丢失掉的数据。
- 可扩展性太差
  表设计的与业务绑定的太紧密、单一，造成表的可拓展性、可修改性太差，无法满足新需求的要求。
- 非必要数据冗余量太大
  没用的垃圾数据存储过多，不仅占用资源，还影响查询效率。

- 不利于计算或统计
  缺少必要的联系性或统计性字段或用于计算统计的字段分散于多个表中，造成计算统计的步骤繁琐，甚至无法计算统计。
- 没有详尽的数据记录信息
  缺少必要的字段，造成无法跟踪数据变化、用户操作，也无法进行数据分析。
- 表之间的耦合性太大
  多张表之间的关联过于紧密，造成一张表发生变化而影响到其他表。
- 字段设计考虑不周
  字段长度过短或字段类型过于明确，造成可发挥、可拓展的空间太小。

数据库交互

最近更新时间：2021-09-17 17:21:30

游戏服务器端的最佳实践

1. 在部分字段读写的场景下推荐部分字段读取和更新，减少网络传输大小和磁盘读写次数，避免获取无效数据。
2. 对于写操作时响应包需要返回数据记录的，建议根据实际情况设置 result_flag，例如 update 后需要获取到修改后的数据记录，建议 result_flag 设置为2；如果不需要写操作返回数据记录，则 result_flag 设置为0。
3. 对于批处理的命令字，例如 batchget、listgetall、getbypartkey 等，建议按照 offset 和 limit 获取数据记录，limit 推荐是 200，gameserver 端开启分包返回。
4. list 相关的表，listaddafter 时需要注意队列满时配置淘汰规则，推荐采用队头插入、队尾删除或者队头删除、队尾插入的方式。
5. list 相关的表，例如 listreplace、listdelete、listbatchdelete 时，需要传递正确的 index。
6. gameserver 在读数据前，需要确保获取的数据记录 value 字段是有值的，例如 value 字段 A、B、C，本次 gameserver 获取的数据记录是A、B，则C字段是没有值的，请谨慎使用。
7. 推荐 gameserver 本地做超时控制，不推荐采用按照 gameserver 的回包做超时判断，需要 gameserver 本地主动的做超时判断。
8. gameserver 在遍历时，推荐从存储层从节点上获取数据，避免影响主节点的性能。
9. 不推荐对大字段进行 memset 操作，耗费更多的 CPU，推荐对大数据结构的部分字段设置为0的方式进行初始化。
0. gameserver 端在处理发往 Tcaplus 的请求和来自 Tcaplus 的响应时，建议采用分治的方法，处理部分发往 Tcaplus 的请求，再处理来自 Tcaplus 的响应包。

系统设计的最佳实践

1. 高频字段、需要一次性原子性操作的字段推荐独立成表。
2. 游戏服务器端在做数据回写时，建议进行流控、按照时间离散化处理。
3. 可以支持动态修改表结构，提供表的数据转换插件。
4. 回档支持全区全服、表级别、记录级别的冷备时间点回档、精确时间点回档。
5. 推荐采用分区分服模型，全区全服和分区分服都可以动态扩展接入层、存储层节点。
6. 具有业务关联的多个内容需要合并在单个表里，避免出现分布式事务问题。
7. 推荐开启压缩功能，包括请求包和响应包压缩、记录压缩功能。