Atitit db core tek 数据库核心技术 高级技术

目录

[第一章 存储引擎 1](#_Toc3585)

[第一节 行存储 json文档 1](#_Toc3795)

[第二节 Catalog kv schema 2](#_Toc8793)

[第二章 索引技术 2](#_Toc26309)

[第一节 Hash 2](#_Toc6583)

[第二节 Btree的实现 （多路 平衡树 有序树 2](#_Toc18604)

[第三节 Fulltext索引 3](#_Toc19232)

[第四节 倒排索引 vs 正派索引 3](#_Toc8638)

[第三章 Sql解析与优化技术 3](#_Toc8231)

[第一节 Druid 3](#_Toc2446)

[第二节 Sql优化 3](#_Toc7455)

[第三节 Sql优化 基于成本 基于规则RBO和CBO 3](#_Toc14862)

[第四章 非必备 3](#_Toc18316)

[第五章 事务机制 3](#_Toc28828)

[第一节 Undo日志 3](#_Toc17794)

[第二节 事务处理wal mvcc 3](#_Toc18563)

[第六章 多核执行 多线程 4](#_Toc9361)

[第七章 Cache技术 4](#_Toc25252)

[第一节 Cache缓存管理 4](#_Toc28007)

[第二节 查询cache 5](#_Toc2835)

[第三节 Query catch 5](#_Toc4451)

[第四节 Update cache 5](#_Toc21741)

[第五节 Sql hint 5](#_Toc27027)

[第八章 视图 sp udf 5](#_Toc9664)

[第一节 物化视图的实现 5](#_Toc31497)

[第九章 Event timer实现 6](#_Toc13616)

[第十章 分区 6](#_Toc4708)

[第十一章 Other 6](#_Toc27327)

[第一节 连接池 6](#_Toc6392)

# 存储引擎

## 行存储 json文档

## **Catalog kv schema**

与KV store有区别的是，数据库是有严格的schema的，所以多数存储引擎中的记录都是有结构的，很多KV store在作为数据库存储引擎时，都是在中间做一层转换，将上层处理的tuples以特定的编码方式转换为binary key-value，写入KVStore，并在读取到上层后，依靠schema解释为tuples格式供上层处理。

这种方法当然可以工作，但是诸多优化无法实施：a. 数据迭代必须是整行，即便只需要其中一列，序列化/反序列化开销是免不了的。b. project和filter的工作无法下放到存储层内部进行处理; c. 没有列信息，做按列编码，压缩也不可能。d. schema change只能暴力重整数据… 因此要做到真正的高效，越来越多的存储引擎选择完全感知schema，存储细微结构

# 索引技术

## Hash

## Btree的实现 （多路 平衡树 有序树

）

左边小右边大

Key，rowid

## Fulltext索引

## 倒排索引 vs 正派索引

# Sql解析与优化技术

## Druid

## Sql优化

## Sql优化 基于成本 基于规则RBO和CBO

# 非必备

# 事务机制

## Undo日志

## **事务处理wal mvcc**

**存储引擎之核心，保证数据库的ACID**。要保证D，大家的做法差不多，都是写WAL(Write Ahead Log)来做recovery，关键是如何高效的实现ACI，也就是所谓的多版本并发控制(MVCC)机制。

MVCC的完整实现比较复杂，暂不详细阐述，这里面的关键在于如何处理并发执行过程中的数据冲突(data race)，包括写写冲突，读写冲突；因为数据库的负载一般是读多写少的，要做到高效，只读事务不能被读写事务阻塞，这就要求我们的写不能直接去更新当前的数据，而是要有一套维护多版本数据的能力，当前的存储引擎管理多版本数据的办法无非两种：

写入数据原地更新，被更新的旧版本写到undo链中，写入代价大，事务处理复杂, 但是回收旧版本数据高效。  
写入数据不直接更新原来的数据，而是追加为新版本，写入代价小，但是读，尤其是扫描需要读取层次较多，更为严重的问题是回收旧版本的数据需要做compact，代价很大。  
前一种称为ARIES算法比大多数主流数据库存储引擎使用，后一种称为LSM-Tree的结构也被很多新存储引擎使用，受到越来越多的关注。

# 多核执行 多线程

# Cache技术

## Cache**缓存管理**

缓存的粒度一般是page，关键在于**缓存替换算法**。目前用的比较广泛的LRU,LFU,ARC..以及各种变种的算法都有在数据库中使用。另外还有一个是如何更有效的管理内存的问题，这点上，定长的page会比不定长的更有优势。

当然还要考虑**各种query pattern对cache的影响**，如果单行查询较多，选用更细粒度(比如row)的cache会更有效率，但是淘汰的策略会更复杂，很多新的研究开始尝试引入机器学习的方法来优化cache淘汰算法，以及有效的管理cache.

## 查询cache

## Query catch

## Update cache

## Sql hint

# 视图 sp udf

## 物化视图的实现

# Event timer实现

# 分区

# Other

## 连接池

ATITIT db perf enhs 数据库性能优化

阿里资深技术专家：优秀的数据库存储引擎应具备哪些能力？ - 知乎