Atitit 存储引擎核心技术 总结

目录

[1. 表的存储有三个文件：结构+数据+索引 1](#_Toc22742)

[2. 页式管理 数据存储格式 1](#_Toc32289)

[2.1. 数据存取的选择：行存储还是列存储？ 1](#_Toc4690)

[2.2. 选择数据在磁盘和内存中的聚集方式 段页式存储管理 2](#_Toc21880)

[2.3. 直接使用索引存储结构(比如B+Tree)直接索引到记录 3](#_Toc10506)

[2.4. Kv式存储 3](#_Toc17078)

[2.5. Lsm索引块存储 3](#_Toc13879)

[2.6. 矩阵式存储 3](#_Toc24853)

[2.7. 八．图结构存储 3](#_Toc15239)

[2.8. 六．对象与块存储 3](#_Toc6523)

[3. Atitit 数据库核心技术index索引技术 btree hash lsm fulltxt 4](#_Toc31861)

[4. 读写更新api 4](#_Toc20220)

[4.1. 简单的实现kv模型 5](#_Toc20280)

[5. Ref 5](#_Toc26751)

[5.1. Atitit.数据库存储引擎的原理与attilax 总结 5](#_Toc2664)

# 表的存储有三个文件：结构+数据+索引

.frm文件，存储表的结构

.myd文件，存储表的数据

.myi文件，存储表的索引

# 页式管理 数据存储格式

与现有的大多数存储引擎一样，InnoDB 使用页作为磁盘管理的最小单位；数据在 InnoDB 存储引擎中都是按行存储的，每个 16KB 大小的页中可以存放 2-200 行的记录。

## 数据存取的选择：行存储还是列存储？

数据按行存储(NSM)，对事务处理比较友好，因为事务数据总是完整行写进来， 多用于OLTP场景。

按列存储(DSM)，把tuples中相同的列值物理上存储在一起，这样只需要读取需要的列，在大规模数据扫描时减少大量I/O。另外列存做压缩的效果更好，适合OLAP场景，但是事务处理就不那么方便，需要做行转列。所以大部分AP数据库事务处理效率都不怎么高，某些甚至只支持批量导入。

混合存储(FSM)，行列混合布局

## 选择数据在磁盘和内存中的聚集方式 段页式存储管理

以按行存储为例，大部分存储引擎使用固定大小的页面(page)来存储连续的若干行。当然，数据行如何连续排列，有堆表(随机)和索引组织表(按索引序)两种，现在较为流行的LSM-Like的存储引擎使用不定大小的页面(称为DataBlock)，只支持按主键索引序聚集；这两种方式主要区别在于前者被设计为可更新的，每个page中会留有空间，后者是只读的，数据紧密存储不带padding，便于压缩。两者的区别实际上是因为事务处理机制有较大的区别导致的，后面再论。

## 直接使用索引存储结构(比如B+Tree)直接索引到记录

对于In-Memory Database来说，数据组织的方式会有较大区别，因为不需要在内存和持久化存储中交换数据，内存中一般不会使用page形式，而是直接使用索引存储结构(比如B+Tree)直接索引到记录(tuples)，无需page这一层间接引用，减少cpu cache miss。

## Kv式存储

## Lsm索引块存储

## 矩阵式存储

## **八．图结构存储**

图通常用来表示和存储具有“多对多”关系的数据，是数据结构中非常重要的一种结构。

## **六．对象与块存储**

本章节描述，以 Ceph 分布式存储系统为参考。

1. 对象存储结构

在文件系统一级提供服务，只是优化了目前的文件系统，采用扁平化方式，弃用了目录树结构，便于共享，高速访问。

对象存储体系结构定义了一个新的、更加智能化的磁盘接口 OSD。OSD 是与网络连接的设备，包含存储介质，如磁盘或磁带，并具有足够智能可管理本地存储的数据。计算结点直接与 OSD 通信，访问它存储的数据，不需要文件服务器的介入。对象存储结构提供的性能是目前其它存储结构难以达到的，如 ActiveScale 对象存储文件系统的带宽可以达到 10GB/s。

# Atitit 数据库核心技术index索引技术 btree hash lsm fulltxt

目录

1.1. HASH索引、 1

1.2. LSM索引 LSM算法(Log Structured-Merge Tree) 2

1.3. BTREE 2

1.4. Fulltxt比较 lsm适合高并发的数据写入，btree更适合大量查询 3

# 读写更新api

Atitit.自定义存储引擎的接口设计 api 标准化 attilax 总结 mysql

1. 图16.1：MySQL体系结构 1

2. 16.7. 创建表create()虚拟函数： 2

3. 16.8. 打开表 open（） 2

4. --------------------------------------------------------------------------------------------------------------------- 2

5. 16.9. 实施基本的表扫描功能 2

5.1. 目录 3

5.1.1. 16.9.1. 实施store\_lock()函数 3

5.1.2. 16.9.2. 实施external\_lock()函数 3

5.1.3. 16.9.3. 实施rnd\_init()函数 3

5.1.4. 16.9.4. 实施info()函数 3

5.1.5. 16.9.5. 实施extra()函数 3

5.1.6. 16.9.6. 实施rnd\_next()函数 3

5.2. CSV引擎的9行表扫描过程中进行的方法调用： 3

5.3. 16.9.1. 实施store\_lock()函数 4

5.4. 实施rnd\_init()函数 5

5.5. 16.9.4. 实施info()函数 5

5.6. 16.9.5. 实施extra()函数 5

5.7. 16.9.6. 实施rnd\_next()函数 6

6. ------------------------------------------------------------------------------------------------------------------- 6

7. 关闭表close(void) 6

8. 6

9. 16.11. 为存储引擎添加对INSERT的支持write\_row() 6

10. 16.12. 为存储引擎添加对UPDATE的支持update\_row() 7

11. 16.13. 为存储引擎添加对DELETE的支持delete\_row() 7

12. 16.14. API引用 与详细说明 7

13. 8

14. 参考 8

## 简单的实现kv模型

仅仅从功能上来说，要实现一个存储引擎似乎并不困难，如今也有很多Key-Value Store摇身一变就成为了数据库存储引擎，无非是加上一套事务处理机制罢了。但是作为数据库的底盘，一个成熟的存储引擎必须要考虑效率，如何高效(性能/成本最大化)的实现数据存取则成了在设计上做出种种权衡的主要考量。可以从存储引擎的几个主要组件来讨论：

# Ref

## Atitit.数据库存储引擎的原理与attilax 总结

1. 存储引擎是什么 1

2. 其它数据库系统(包括大多数商业选择)仅支持一种类型的数据存储 2

3. 表的存储有三个文件：结构+数据+索引 2

4. 页和字段 2

5. 数据存取的选择：行存储还是列存储？ 3

6. 常见的存储引擎 3

6.1. 简单类型MyISAM、 3

6.2. 复杂类型，支持事务与外键 MySQL存储引擎【InnoDB、 3

6.3. InnoDB数据存储结构 3

6.4. Memory】(Heap) 存储引擎 5

6.5. NDBCluster分布式存储引擎 6

7. other 6

7.1. CSV： 逻辑上由逗号分割数据的存储引擎。它会在数据库子目录里为每个数据表创建一个.CSV文件。这是一种普通文本文件，每个数据行占用一个文本行。CSV存储引擎不支持索引。 6

7.2. Merge ：将一定数量的MyISAM表联合而成一个整体，在超大规模数据存储时很有用 7

7.3. Archive ：非常适合存储大量的独立的，作为历史记录的数据。因为它们不经常被读取。Archive拥有高效的插入速度，但其对查询的支持相对较差 7

7.4. Federated存储引擎 7

8. 参考 7

关于数据存储引擎结构，没有比这篇更详细的 - InfoQ 写作平台