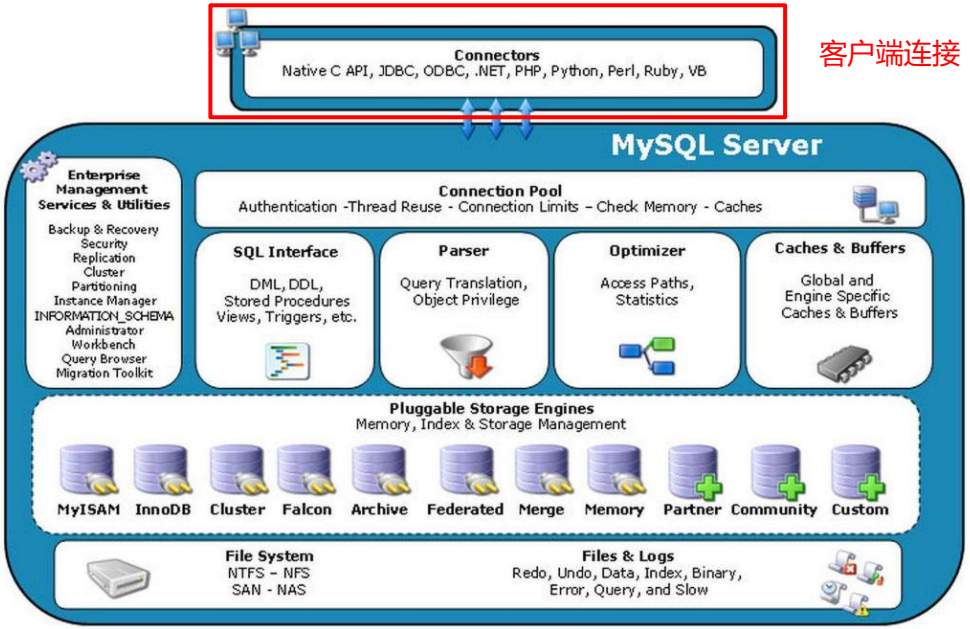
MySQL 高级

目录 Contents

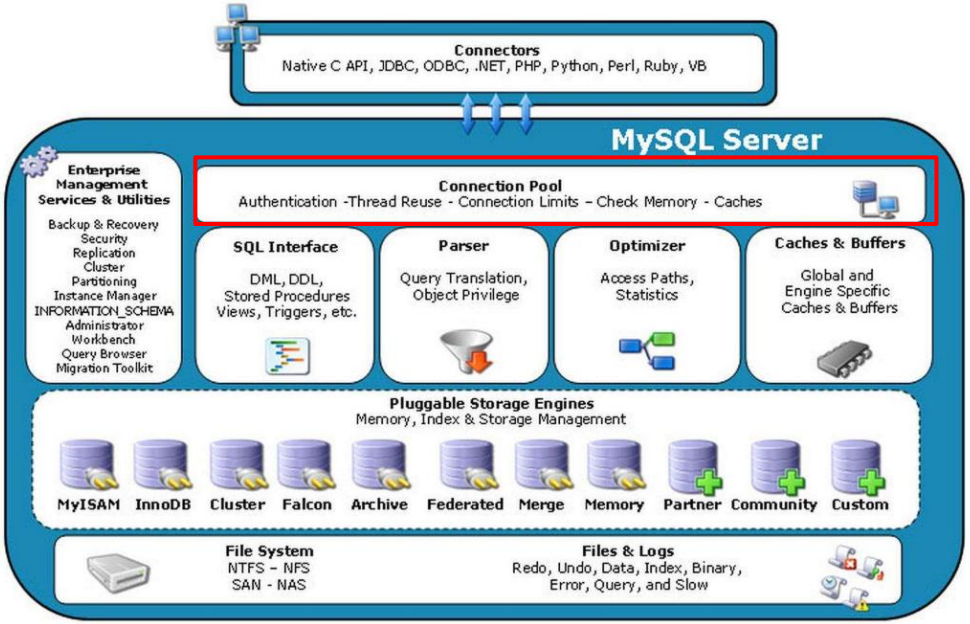
MySQL 存储引擎

MySQL 体系结构



MySQL 存储引擎

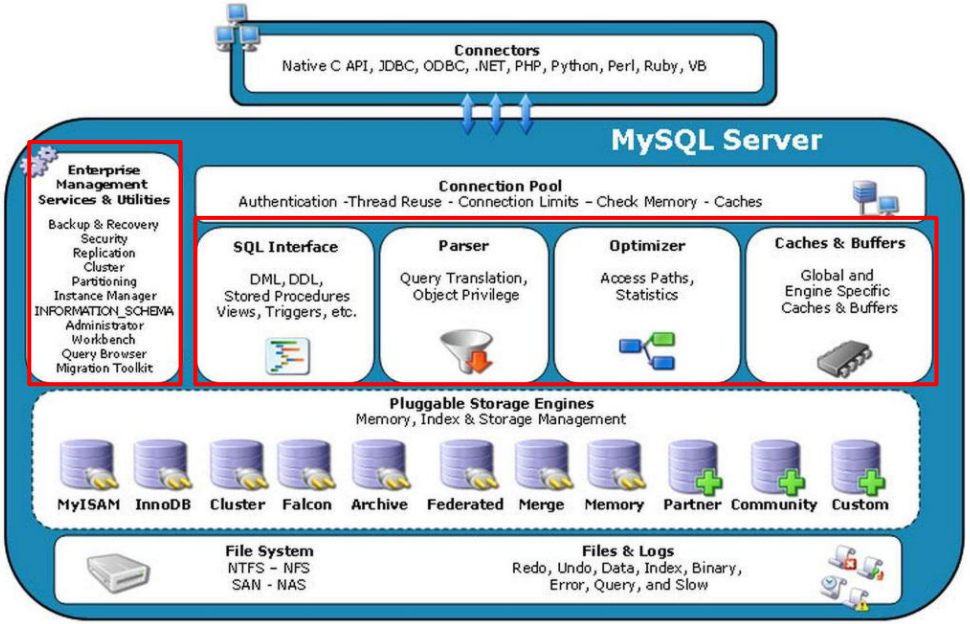
MySQL 体系结构



第一层：网络连接层

MySQL 存储引擎

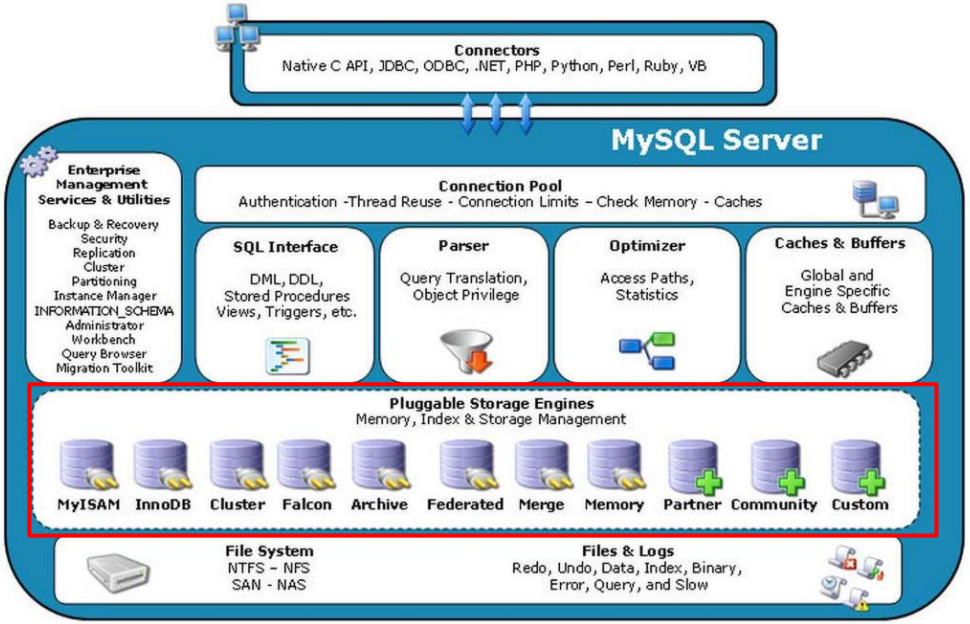
MySQL 体系结构



第二层：核心服务层

MySQL 存储引擎

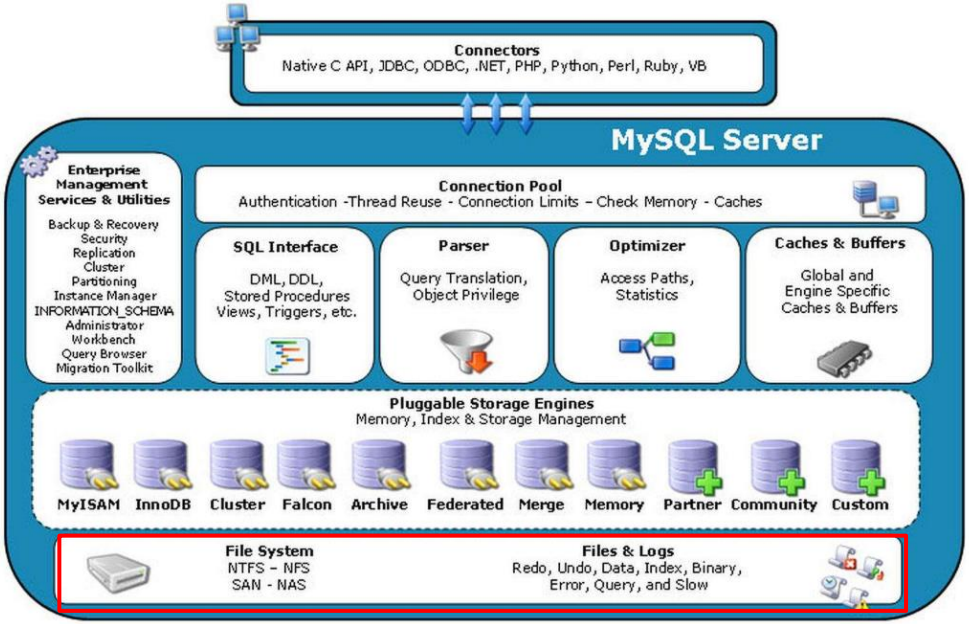
MySQL 体系结构



第三层：存储引擎层

MySQL 存储引擎

MySQL 体系结构



第四层：系统文件层

MySQL 体系结构

- 客户端连接

  支持接口：支持的客户端连接，例如 C、Java、PHP 等语言来连接 MySQL 数据库。

- 第一层：网络连接层

  连接池：管理、缓冲用户的连接，线程处理等需要缓存的需求。

- 第二层：核心服务层

  管理服务和工具：系统的管理和控制工具，例如备份恢复、复制、集群等。

  SQL 接口：接受 SQL 命令，并且返回查询结果。

  查询解析器：验证和解析 SQL 命令，例如过滤条件、语法结构等。

  查询优化器：在执行查询之前，使用默认的一套优化机制进行优化sql语句。

  缓存：如果缓存当中有想查询的数据，则直接将缓存中的数据返回。没有的话再重新查询。

- 第三层：存储引擎层

  插件式存储引擎：管理和操作数据的一种机制，包括(存储数据、如何更新、查询数据等)

- 第四层：系统文件层

  文件系统：配置文件、数据文件、日志文件、错误文件、二进制文件等等的保存。

MySQL 存储引擎

存储引擎介绍

- MySQL 数据库使用不同的机制存取表文件, 包括存储方式、索引技巧、锁定水平等不同的功能。这些不同的技术以及配套的功能称为存储引擎。
- Oracle、SqlServer 等数据库只有一种存储引擎。而 MySQL 针对不同的需求, 配置不同的存储引擎, 就会让数据库采取不同处理数据的方式和扩展功能。
- MySQL 支持的存储引擎有很多，常用的有三种：InnoDB、MyISAM、MEMORY。
- 特性对比

  MyISAM 存储引擎：访问快，不支持事务和外键操作。

  InnoDB 存储引擎：支持事务和外键操作，支持并发控制，占用磁盘空间大。(MySQL 5.5版本后默认)

  MEMORY 存储引擎：内存存储，速度快，不安全。适合小量快速访问的数据。

MySQL 存储引擎

存储引擎介绍

特性	MyISAM	InnoDB	MEMORY
存储限制	有(平台对文件系统大小的限制)	64TB	有(平台的内存限制)
事务安全	**不支持**	**支持**	**不支持**
锁机制	**表锁**	**表锁/行锁**	**表锁**
B+Tree索引	支持	支持	支持
哈希索引	不支持	不支持	支持
全文索引	支持	支持	不支持
集群索引	**不支持**	**支持**	**不支持**
数据索引	不支持	支持	支持
数据缓存	不支持	支持	N/A
索引缓存	支持	支持	N/A
数据可压缩	支持	不支持	不支持
空间使用	低	高	N/A
内存使用	低	高	中等
批量插入速度	高	低	高
外键	**不支持**	**支持**	**不支持**

MySQL 存储引擎

存储引擎的操作

- 查询数据库支持的存储引擎

```
SHOW ENGINES;
```

- 查询某个数据库中所有数据表的存储引擎

```
SHOW TABLE STATUS FROM 数据库名称;
```

- 查询某个数据库中某个数据表的存储引擎

```
SHOW TABLE STATUS FROM 数据库名称 WHERE NAME = '数据表名称';
```

- 创建数据表，指定存储引擎

```
CREATE TABLE 表名(
    列名,数据类型,
    ...
)ENGINE = 引擎名称;
```

- 修改数据表的存储引擎

```
ALTER TABLE 表名 ENGINE = 引擎名称;
```

MySQL 存储引擎

存储引擎的选择

- MyISAM

  特点：不支持事务和外键操作。读取速度快，节约资源。

  使用场景：以查询操作为主，只有很少的更新和删除操作，并且对事务的完整性、并发性要求不是很高！

- InnoDB

  特点：MySQL 的默认存储引擎，支持事务和外键操作。

  使用场景：对事务的完整性有比较高的要求，在并发条件下要求数据的一致性，读写频繁的操作！

- MEMORY

  特点：将所有数据保存在内存中，在需要快速定位记录和其他类似数据环境下，可以提供更快的访问。

  使用场景：通常用于更新不太频繁的小表，用来快速得到访问的结果！

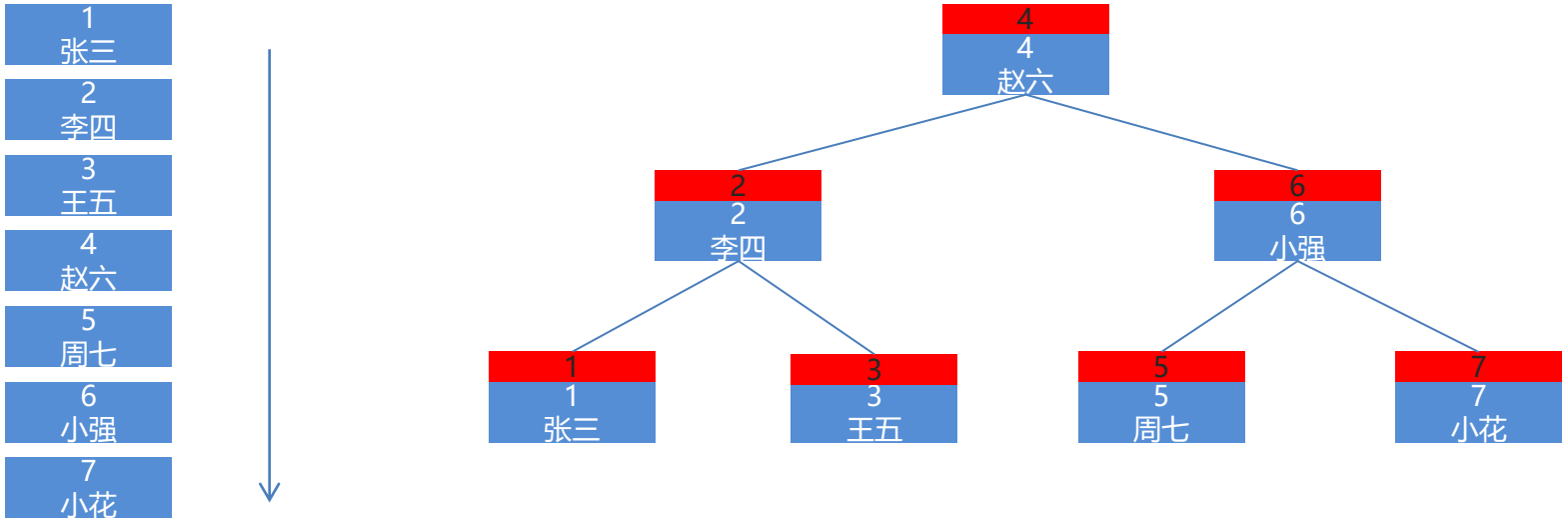
总结：针对不同的需求场景，来选择最适合的存储引擎即可！如果不确定、则使用数据库默认的存储引擎！

目录 Contents

MySQL 索引

索引介绍

- MySQL 索引：是帮助 MySQL 高效获取数据的一种数据结构。所以，索引的本质就是数据结构！
- 在表数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式指向数据， 这样就可以在这些数据结构上实现高级查找算法，这种数据结构就是索引。



索引的分类

- 按照功能分类

  普通索引：最基本的索引，没有任何限制。

  唯一索引：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值组合必须唯一。

  主键索引：一种特殊的唯一索引，不允许有空值。在建表时有主键列同时创建主键索引。

  联合索引：顾名思义，就是将单列索引进行组合。

  外键索引：只有 InnoDB 引擎支持外键索引，用来保证数据的一致性、完整性和实现级联操作。

  全文索引：快速匹配全部文档的方式。InnoDB 引擎 5.6 版本后才支持全文索引。MEMORY 引擎不支持。

- 按照结构分类

  BTree 索引：MySQL 使用最频繁的一个索引数据结构，是 InnoDB 和 MyISAM 存储引擎默认的索引类型，底层基于 B+Tree 数据结构。

  Hash 索引：MySQL 中 Memory 存储引擎默认支持的索引类型。

MySQL 索引

索引的操作

- 创建索引

```
CREATE [ UNIQUE | FULLTEXT ] INDEX 索引名称
[USING 索引类型]  -- 默认是BTREE
ON 表名(列名...);
```

- 查看索引

```
SHOW INDEX FROM 表名;
```

MySQL 索引

索引的操作

- 添加索引

  普通索引：ALTER TABLE 表名 ADD INDEX 索引名称(列名);

  组合索引：ALTER TABLE 表名 ADD INDEX 索引名称(列名1,列名2,...);

  主键索引：ALTER TABLE 表名 ADD PRIMARY KEY(主键列名);

  外键索引：ALTER TABLE 表名 ADD CONSTRAINT 外键名 FOREIGN KEY (本表外键列名) REFERENCES 主表名(主键列名);

  唯一索引：ALTER TABLE 表名 ADD UNIQUE 索引名称(列名);

  全文索引：ALTER TABLE 表名 ADD FULLTEXT 索引名称(列名);

- 删除索引

  DROP INDEX 索引名称 ON 表名;

索引的原理

- 索引是在存储引擎中实现的，不同的存储引擎所支持的索引也不一样，这里我们主要介绍 InnoDB 引擎的 BTree 索引。
- BTree 索引类型是基于 B+Tree 数据结构的，而 B+Tree 数据结构又是 BTree 数据结构的变种。通常使用在数据库和操作系统中的文件系统，特点是能够保持数据稳定有序。
- 需要理解的

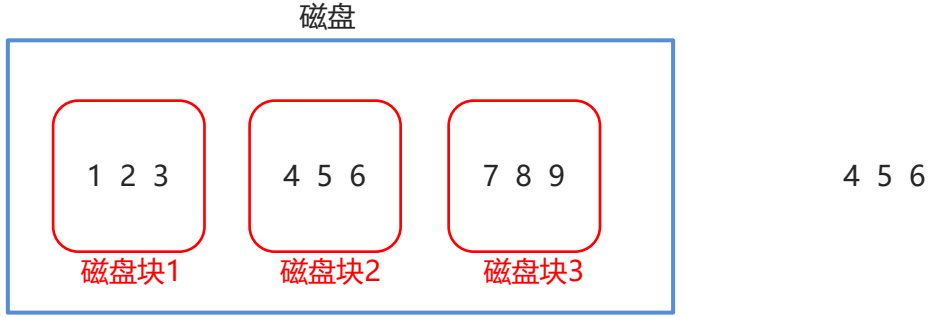
  磁盘存储。

  BTree。

  B+Tree。

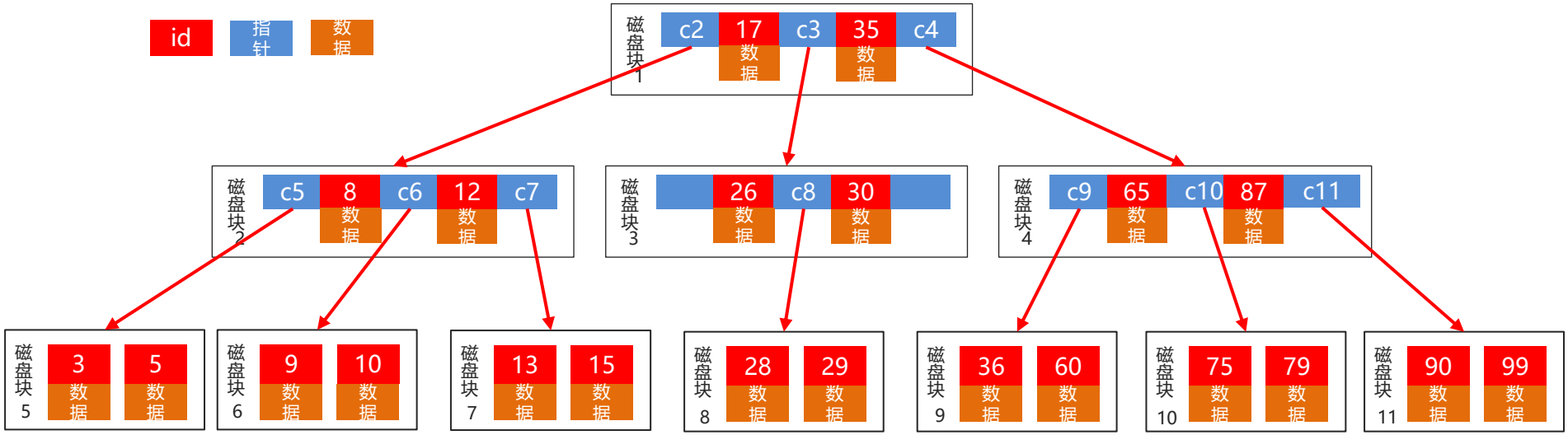
索引的原理 – 磁盘存储

- 系统从磁盘读取数据到内存时是以磁盘块（block）为基本单位的。
- 位于同一个磁盘块中的数据会被一次性读取出来，而不是需要什么取什么。
- InnoDB 存储引擎中有页（Page）的概念，页是其磁盘管理的最小单位。InnoDB 存储引擎中默认每个页的大小为 16KB。
- InnoDB 引擎将若干个地址连接磁盘块，以此来达到页的大小 16KB，在查询数据时如果一个页中的每条数据都能有助于定位数据记录的位置，这将会减少磁盘 I/O 次数，提高查询效率。



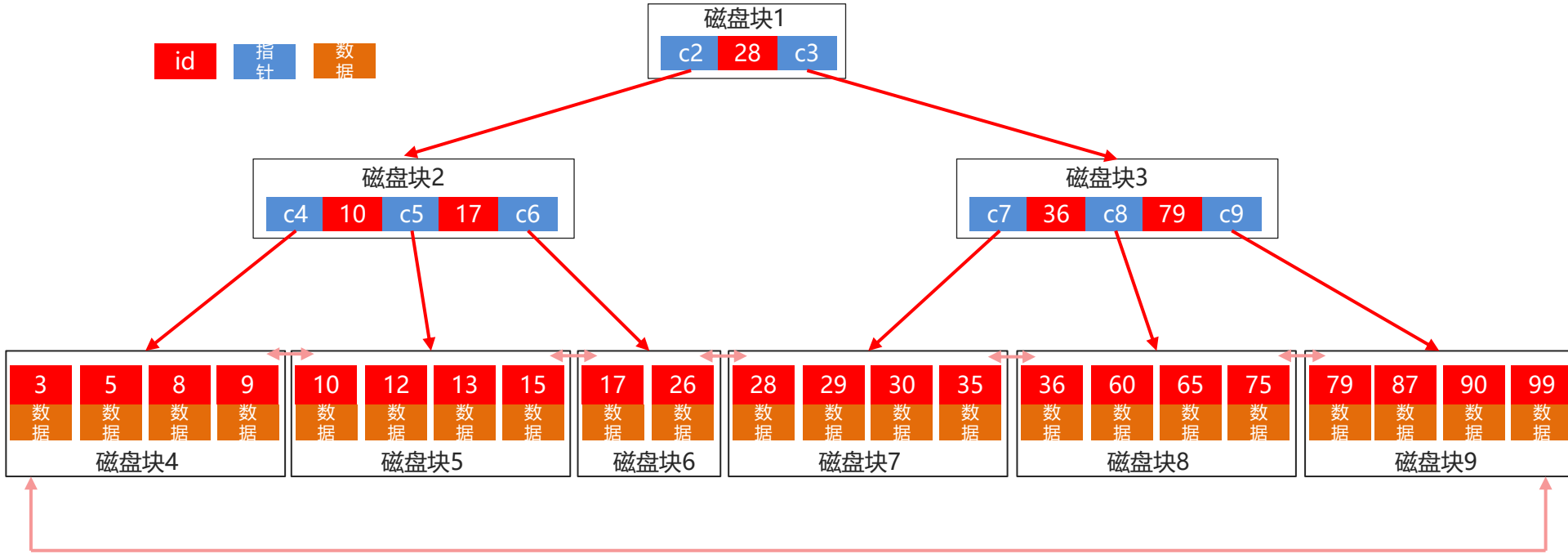
MySQL 索引

索引的原理 – BTree



MySQL 索引

索引的原理 – B+Tree



索引的原理 – B+Tree

- BTree 数据结构

  每个节点中不仅包含 key 值，还有数据。会增加查询数据时磁盘的 IO 次数。

- B+Tree 数据结构

  非叶子节点只存储 key 值。

  所有数据存储在叶子节点。

  所有叶子节点之间都有连接指针。

- B+Tree 好处

  提高查询速度。

  减少磁盘的 IO 次数。

  树型结构较小。

索引的设计原则

- 创建索引遵循的原则
  1. 对查询频次较高，且数据量比较大的表建立索引。
  2. 使用唯一索引，区分度越高，使用索引的效率越高。
  3. 索引字段的选择，最佳候选列应当从 where 子句的条件中提取。
  4. 索引虽然可以有效的提升查询数据的效率，但并不是多多益善。

索引的设计原则

- 最左匹配原则(适用组合索引)
- 例如：为 user 表中的 name、address、phone 列添加组合索引

  ALTER TABLE user ADD INDEX idx_three(name,address,phone);
- 此时，组合索引 idx_three 实际建立了 (name)、(name,address)、(name,address,phone) 三个索引
- 下面的三个 SQL 语句都可以命中索引

  SELECT * FROM user WHERE address = '北京' AND phone = '12345' AND name = '张三';

  SELECT * FROM user WHERE name = '张三' AND address = '北京';

  SELECT * FROM user WHERE name = '张三';
- 这三条 SQL 语句在检索时分别会使用以下索引进行数据匹配

  (name,address,phone)

  (name,address)

  (name)
- 索引字段出现的顺序可以是任意的，MySQL 优化器会帮我们自动的调整 where 条件中的顺序
- 如果组合索引中最左边的列不在查询条件中，则不会命中索引

  SELECT * FROM user WHERE address = '北京';

目录 Contents

MySQL 锁机制

锁的介绍

- 锁机制：数据库为了保证数据的一致性，在共享的资源被并发访问时变得安全所设计的一种规则。
- 锁机制类似多线程中的同步，作用就是可以保证数据的一致性和安全性。

- 按操作分类
  - 共享锁：也叫读锁。针对同一份数据，多个事务读取操作可以同时加锁而不互相影响，但是不能修改数据。
  - 排他锁：也叫写锁。当前的操作没有完成前，会阻断其他操作的读取和写入。
- 按粒度分类
  - 表级锁：会锁定整个表。开销小，加锁快。锁定力度大，发生锁冲突概率高，并发度低。不会出现死锁情况。
  - 行级锁：会锁定当前行。开销大，加锁慢。锁定粒度小，发生锁冲突概率低，并发度高。会出现死锁情况。
- 按使用方式分类
  - 悲观锁：每次查询数据时都认为别人会修改，很悲观，所以查询时加锁。
  - 乐观锁：每次查询数据时都认为别人不会修改，很乐观，但是更新时会判断一下在此期间别人有没有去更新这个数据。

MySQL 锁机制

锁的介绍

- 不同存储引擎支持的锁

存储引擎	表锁	行锁
InnoDB	支持	支持
MyISAM	支持	不支持
MEMORY	支持	不支持

InnoDB 共享锁

- 共享锁特点

  数据可以被多个事务查询，但是不能修改。

- 创建共享锁格式

  SELECT语句 LOCK IN SHARE MODE;

InnoDB 排他锁

- 排他锁特点

  加锁的数据，不能被其他事务加锁查询或修改。

- 创建排他锁格式

  SELECT语句 FOR UPDATE;

MySQL 锁机制

MyISAM 读锁

- 读锁特点

  所有连接只能查询数据，不能修改。

- 读锁语法格式

  加锁

  ```
  LOCK TABLE 表名 READ;
  ```

  解锁

  ```
  UNLOCK TABLES;
  ```

MyISAM 写锁

- 写锁特点

  其他连接不能查询和修改数据。

- 写锁语法格式

  加锁

  ```
  LOCK TABLE 表名 WRITE;
  ```

  解锁

  ```
  UNLOCK TABLES;
  ```

悲观锁和乐观锁

- 悲观锁

  就是很悲观，它对于数据被外界修改的操作持保守态度，认为数据随时会修改。

  整个数据处理中需要将数据加锁。悲观锁一般都是依靠关系型数据库提供的锁机制。

  我们之前所学的锁机制都是悲观锁。

- 乐观锁

  就是很乐观，每次自己操作数据的时候认为没有人会来修改它，所以不去加锁。

  但是在更新的时候会去判断在此期间数据有没有被修改。

  需要用户自己去实现，不会发生并发抢占资源，只有在提交操作的时候检查是否违反数据完整性。

MySQL 锁机制

悲观锁和乐观锁

- 方式一

  给数据表中添加一个 version 列，每次更新后都将这个列的值加 1。

  读取数据时，将版本号读取出来，在执行更新的时候，比较版本号。

  如果相同则执行更新，如果不相同，说明此条数据已经发生了变化。

  用户自行根据这个通知来决定怎么处理，比如重新开始一遍，或者放弃本次更新。

- 方式二

  和版本号方式基本一样，给数据表中添加一个列，名称无所谓，数据类型需要是 timestamp。

  每次更新后都将最新时间插入到此列。

  读取数据时，将时间读取出来，在执行更新的时候，比较时间。

  如果相同则执行更新，如果不相同，说明此条数据已经发生了变化。

传智播客旗下高端IT教育品牌