数据库个人理解

目录

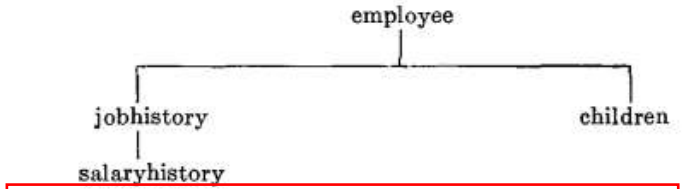
books

- 《深入浅出 MySQL》
  《MySQL 技术内幕：InnoDB存储引擎》
  《高性能 MySQL》
- 《数据库原理及应用(mysql) 》
- 《数据库系统概念》

数据库设计

- 0.数据库设计三范式的来源（为什么要有范式）：
- Some more complicated data structure in necessary for a relation with one or more *nonsimple domains*.For this reason the possibility of eliminating *nonsimple domains* appears worth investigating.
  - 与一个或多个非单域的关系需要一些更复杂的数据结构。出于这个原因，消除非简单域的可能性似乎值得研究。
- There is in fact, a very simple elimination procedure, which we shall call normalization.
  - 事实上，有一个非常简单的消除步骤，我们称之为规范化。

- 右图有个规范化的例子，可以看一下 →
- Ps：斜体的是主键



employee (*man#*, name, birthdate, jobhistory, children)
jobhistory (*jobdate*, title, salaryhistory)
salaryhistory (*salarydate*, salary)
children (*childname*, birthyear)

FIG. 3(a). Unnormalized set

employee′ (*man#*, name, birthdate)
jobhistory′ (*man#*, *jobdate*, title)
salaryhistory′ (*man#*, *jobdate*, *salarydate*, salary)
children′ (*man#*, *childname*, birthyear)

FIG. 3(b). Normalized set

有关于本文献的启示—非关系型数据库（nosql）

- Nonsimple domains
- 一行中的值不一定是一个像数字或字符串一样的原始数据类型，也可以是一个嵌套的关系（表），因此可以把一个任意嵌套的树结构作为一个值，这很像30年后添加到SQL中的JSON或XML支持。

同时也是未来json存储非关系数据库的基石

- If the user's relational model is set up in normal form, names of items of data in the data bank can take a simpler form than would otherwise be the case. A general name would take a form such as R(g).r.d
- where R is a relational name; g is a generation identifier(optional); r is a role name(optional); d is a domain name.

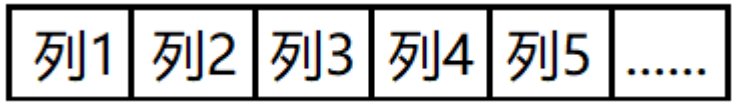
- E. F. Codd. 1970. A relational model of data for large shared data banks. Commun. ACM 13, 6 (June 1970), 377–387. https://doi.org/10.1145/362384.362685

数据库设计

- 1.数据库设计三范式
- .数据库表的设计依据，教我们怎么进行数据库表的设计。
- **设计数据库表的时候，按照以上的范式进行，可以避免表中数据的冗余，空间的浪费。**
- 翻译一下就是：怎么能够创建一张每个字段都不怎么浪费的表。

数据库设计三范式内容

第一范式（1NF）：



列1唯一确定列2, 列3, 列4, ...，即列2, 列3, 列4, ...不能再分裂出其它列。

例子:

学号（主键）	成绩	绩点
0	100	5
1	100	4
2	90	3

```
CREATE TABLE `test`.`XXX` ( `学号` INT NOT NULL auto_increment, `成绩` INT NOT NULL, `绩点` INT NOT NULL, PRIMARY KEY (`学号`))
ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

第二范式（2NF）：

满足2NF的前提是必须满足1NF。

此外，关系模式需要包含两部分内容。

一是必须有一个（及以上）主键；

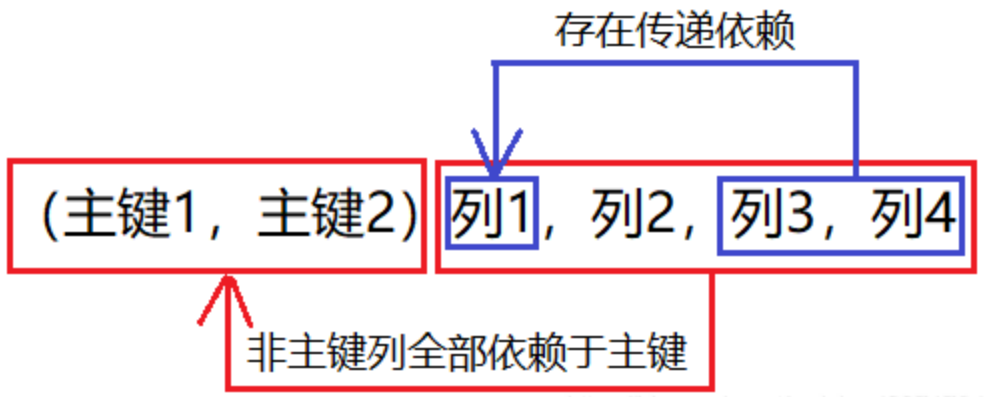
二是没有包含在主键中的列必须全部依赖于全部主键，而不能只依赖于主键的一部分而不依赖全部主键。

学号（主键）	成绩	绩点
0	100	5
1	100	4
2	90	3

第三范式（**3NF**）：

满足3NF的前提是必须满足2NF。

另外关系模式的非主键列必须直接依赖于主键，不能存在传递依赖。



学号（主键）	成绩	学习科目数量
0	100	5
1	100	4
2	90	3

但是…

- 请注意：这里所说的所有有关于“数据库范式”的内容均为一种“规范”，或者说是“通法”，而不是一板一眼的死板学识。
- 请设计数据库时遵循业务逻辑！不要为了遵循范式而强行遵循。
- 业务逻辑：老板让你建立一个表记录的内容类型（以老板的话为准）

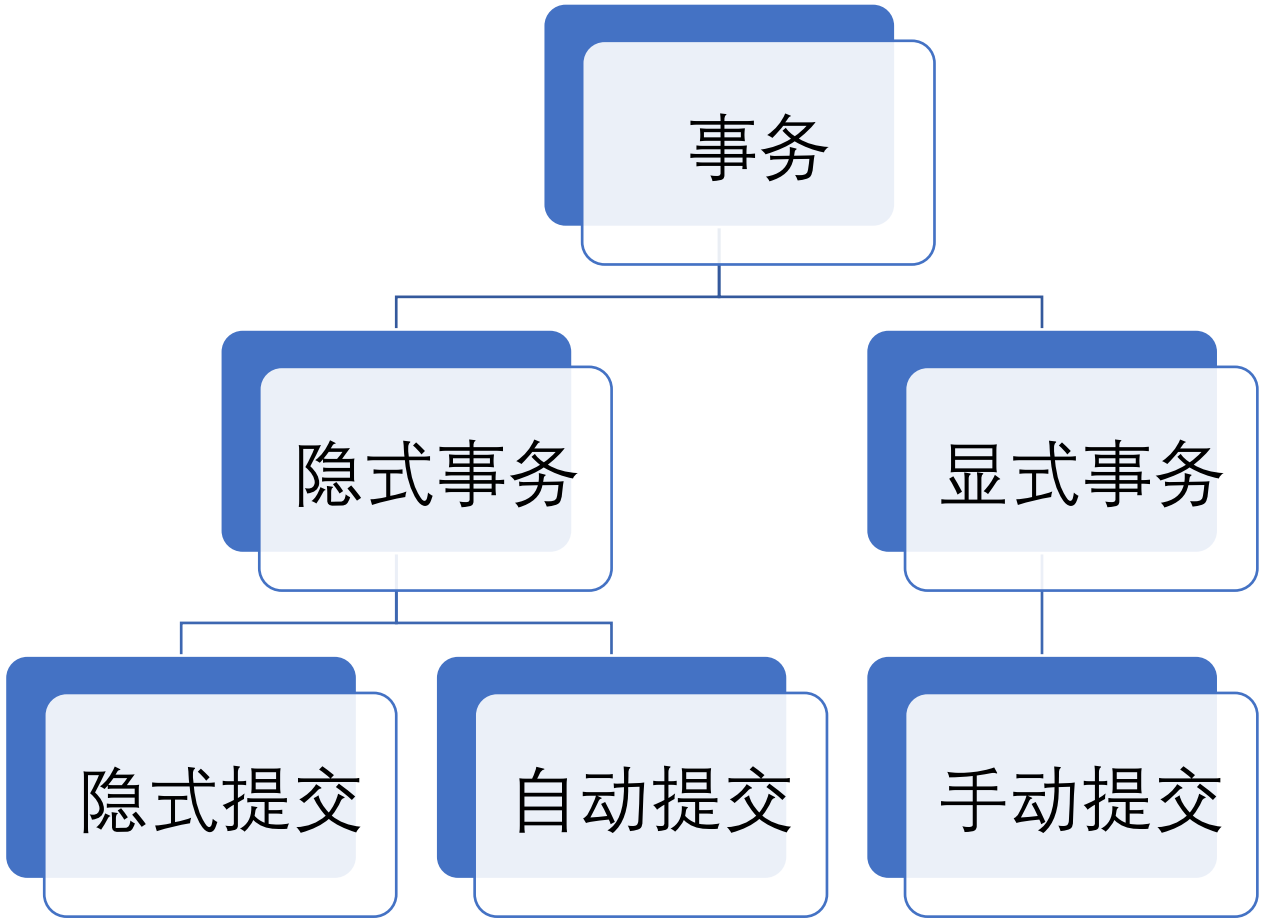
事务处理

- 事务，是一批对数据库的操作；可以理解为批处理文件（windows的.bat/linux的.shell）
- 于此同时，事务由于mysql的单进程和多线程，也可以理解为在一个进程中的一个线程。
- 事务处理分为显式事务和隐式事务。

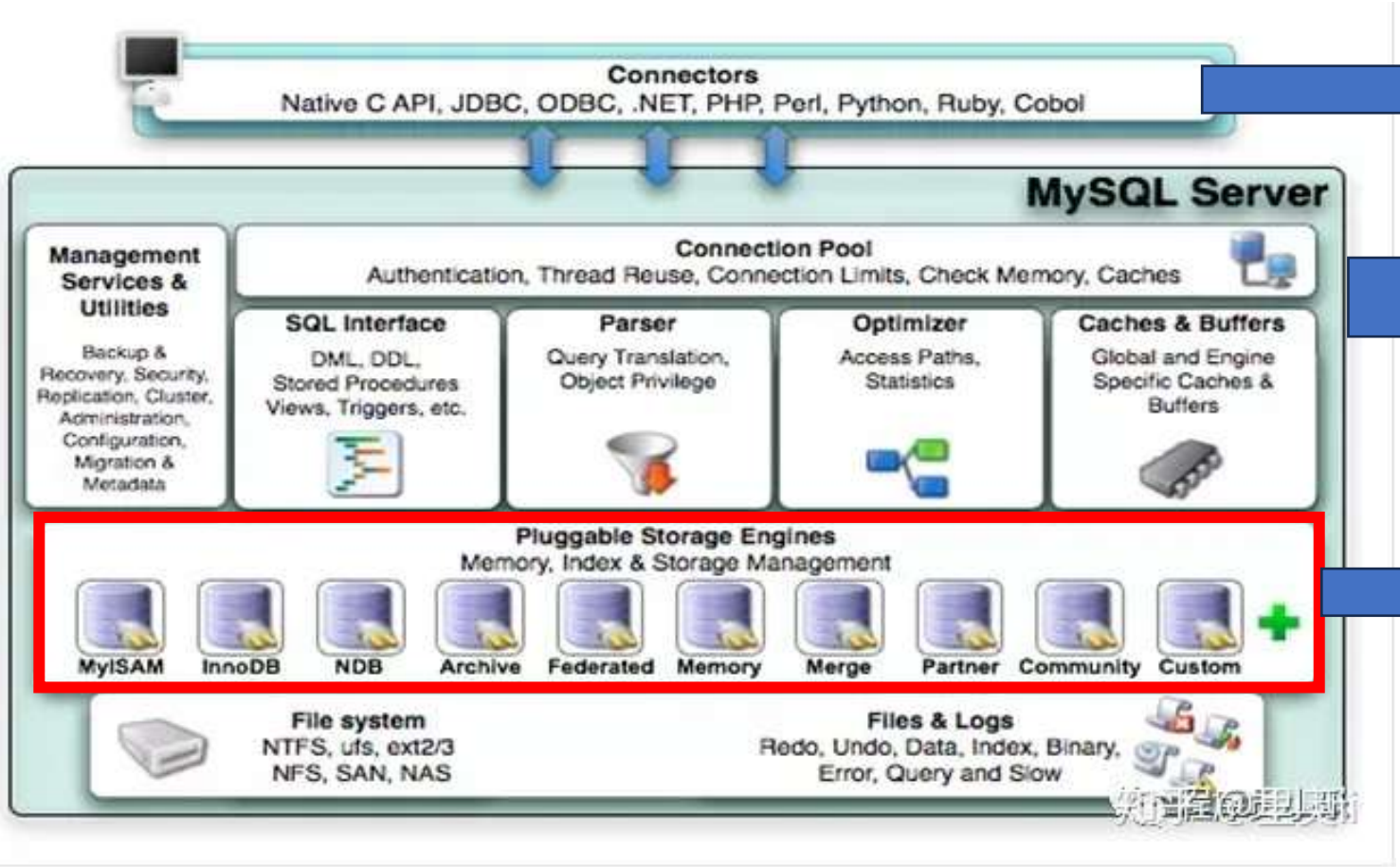
- 事务语法如下：

- BEGIN;
- 你要执行的sql语句;
- COMMIT;

显式事务:用commit/rollback的
隐式事务:又称自动提交事务，意思是在你写完一个事务后不commit/rollback，sql server会自动帮你提交


事务
隐式事务
显式事务
隐式提交
自动提交
手动提交

事务处理-知识之mysql运行原理



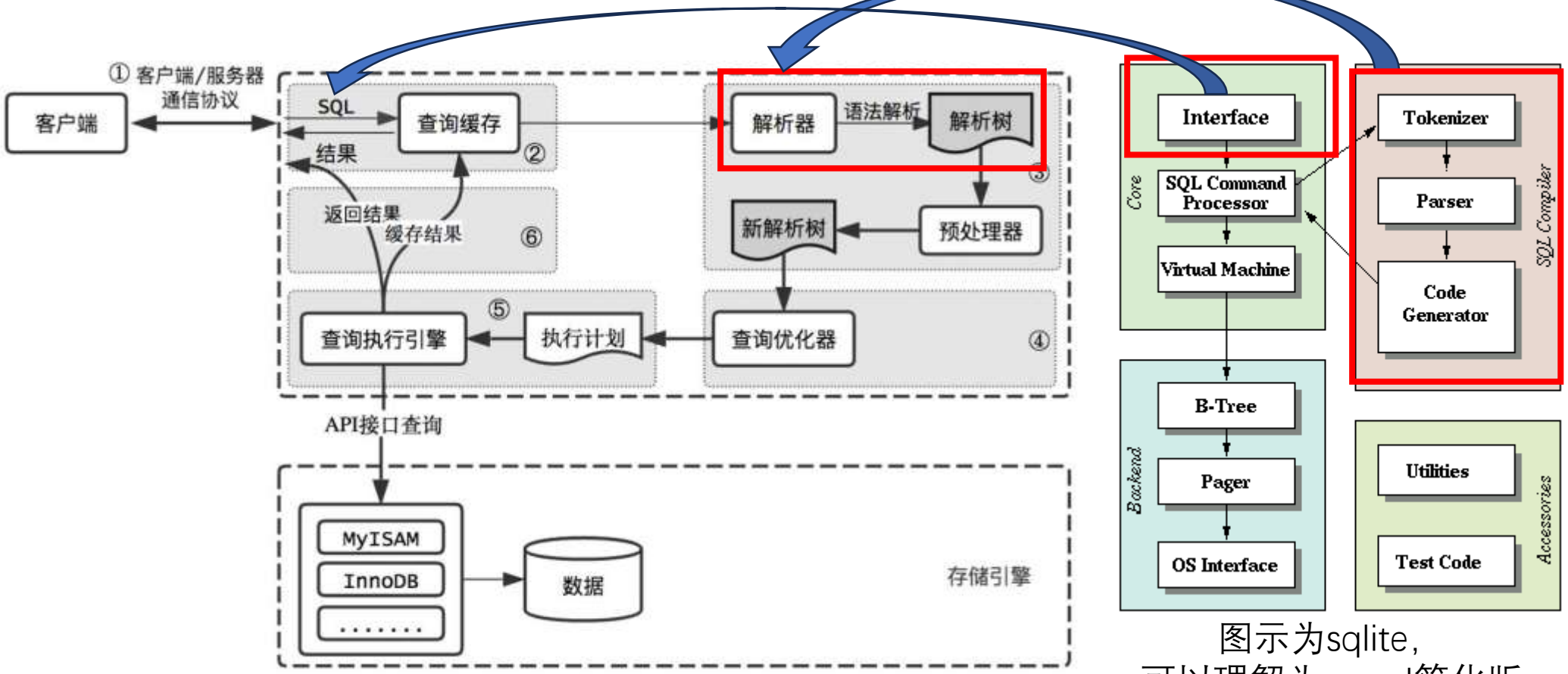
连接器

Mysql服务器

存储引擎

Mysql服务器---what happening?

- 当向MySQL发送一个请求的时候，MySQL到底做了些什么呢?



图示为sqlite，
可以理解为mysql简化版

TCP/IP

[zyb123nya/How-it-works-MySQL: about MySQL8.0 running. (github.com)]

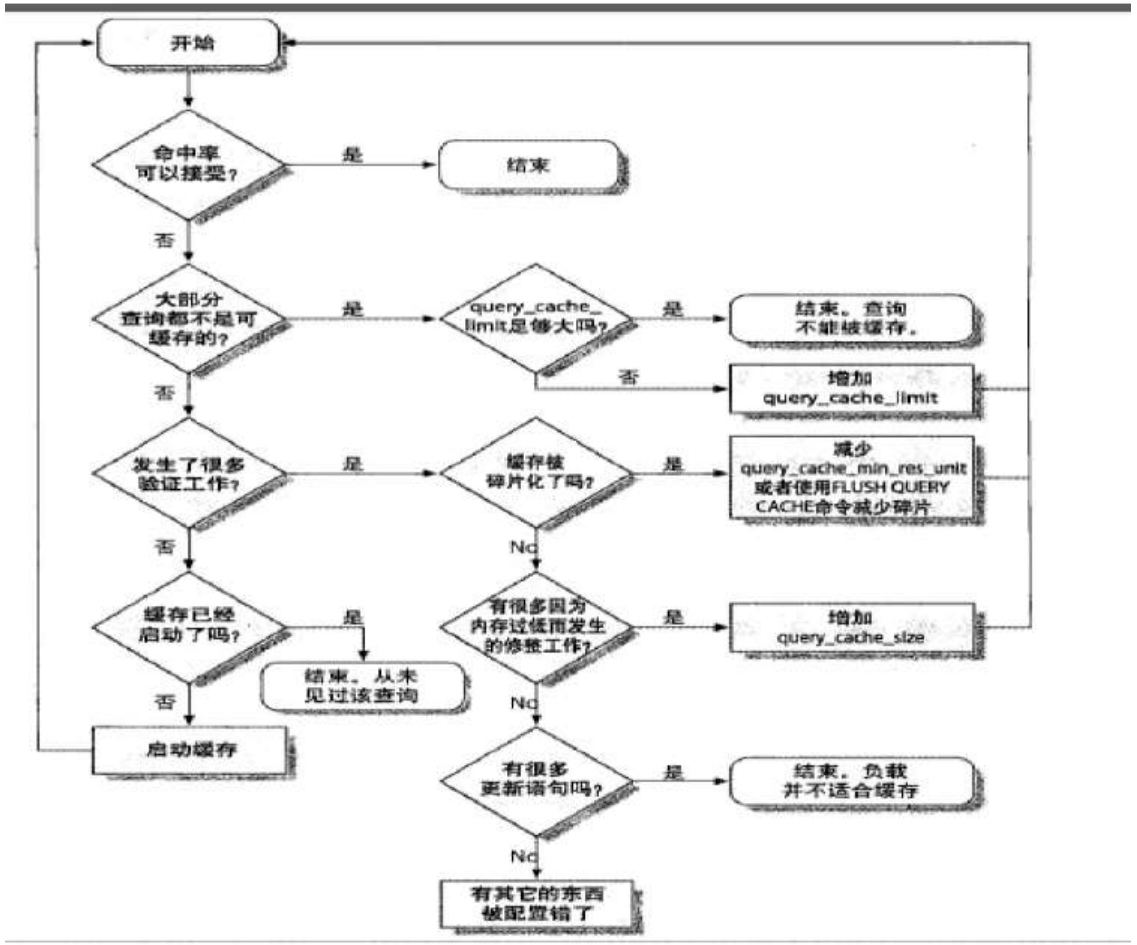
这个东西其实已经由各大connectors([MySQL :: MySQL Connectors])实现了，这里只是复现一下原理。

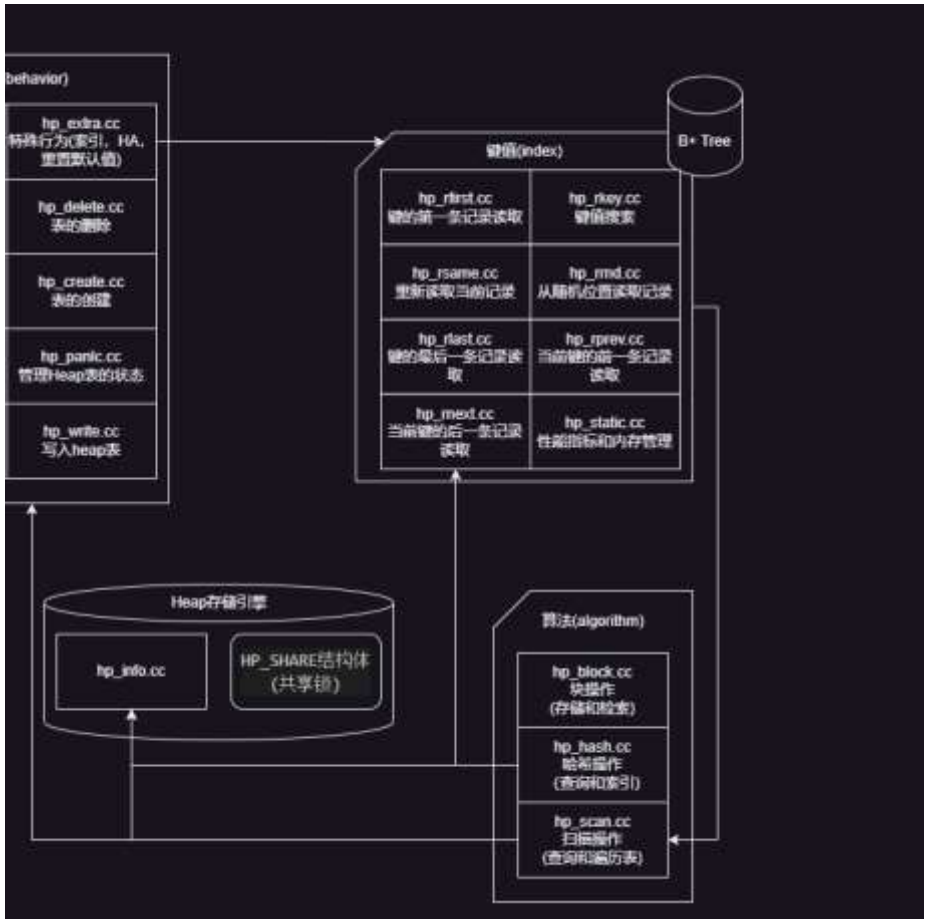
缓存查询

- Mysql8.0已经废除；但是我们可以分析其逻辑：
- 1.将你的select语句进行hash运算，得到hash值。
- 2.存入cache，等待mysql server响应
- 3.mysql server启动，打开cache，接受select语句，通过计算hash值进行匹配。

- 但我们可以搞一个简易版本(已经上传至github)

缓存查询与索引


开始
命中率可以接受？
是
结束
否
大部分查询都不是可缓存的？
是
query_cache_limit足够大吗？
是
结束。查询不能被缓存。
否
增加query_cache_limit
否
发生了很多验证工作？
是
缓存被碎片化了吗？
是
减少query_cache_min_res_unit或者使用FLUSH QUERY CACHE命令减少碎片
否
No
缓存已经启动了吗？
是
结束。从未见过该查询
否
有很多因为内存过低而发生的修整工作？
是
增加query_cache_size
No
启动缓存
有很多更新语句吗？
是
结束。负载并不适合缓存
No
有其它的东西被配置错了


behavior)
hp_extra.cc
特殊行为(索引，HA，重置默认值)
hp_delete.cc
表的删除
hp_create.cc
表的创建
hp_panic.cc
管理Heap表的状态
hp_write.cc
写入heap表
键值(index)
B+ Tree
hp_rfirst.cc
键的第一条记录读取
hp_rkey.cc
键值搜索
hp_rsame.cc
重新读取当前记录
hp_rnd.cc
从随机位置读取记录
hp_rlast.cc
键的最后一条记录读取
hp_rprev.cc
当前键的前一条记录读取
hp_rnext.cc
当前键的后一条记录读取
hp_static.cc
性能指标和内存管理
Heap存储引擎
hp_info.cc
HP_SHARE结构体(共享锁)
算法(algorithm)
hp_block.cc
块操作(存储和检索)
hp_hash.cc
哈希操作(查询和索引)
hp_scan.cc
扫描操作(查询和遍历表)

缓存查询与redis mysql本地缓存与k-v缓存

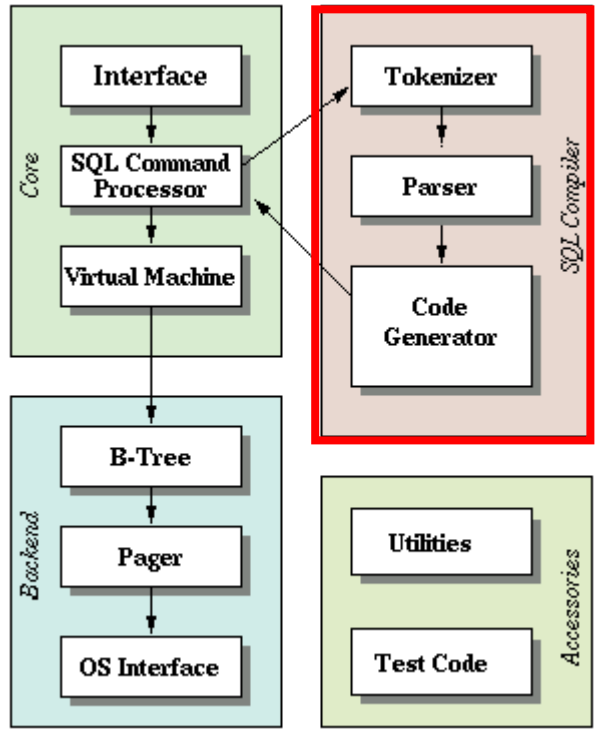
dlc :关于redis缓存问题—本质上是I/O问题

- 缓存穿透
- 缓存雪崩
- 缓存击穿

dlc2 :关于redis缓存问题-与数据库的一致性

解析器

- 有一个自己实现的小型解析器，可以参考github（



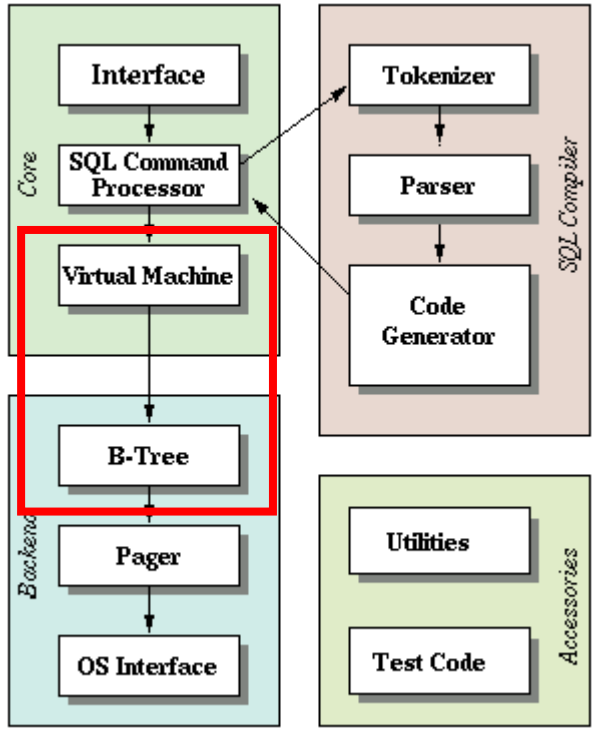
查询优化器

执行计划

执行计划与事务的区别

- 执行计划：描述sql语句在数据库的执行过程
- 事务处理：一组sql操作
- 好比一个是史官，一个是工人。

存储引擎（innodb为例）



并发处理小能手-锁

- 以下，我们讲讨论mysql在并发情况下的问题。

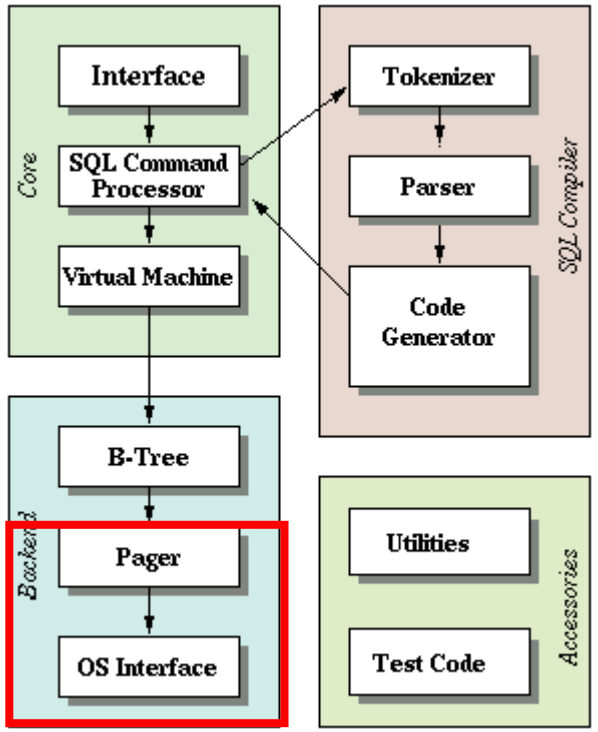
锁(数据库系统)

- 概念：字面意思，对数据加一个锁。
- 如果把表/库/页当成一道门，那么锁就是限制人员（事务提交）进出的一个工具。
- 为什么要有锁?
- 锁是一种用来调度事务顺序的一个协调工具，让各个事务与数据在既保持一致性写入的同时还有序排队。
- 类似测核酸时候你排队的样子.jpg
- 你也不想你家门被一堆人挤着进去吧.jpg

锁介绍

- 在mysql中，主要存在以下锁：
- **按锁粒度分类**
- **行级锁&表级锁&页级锁**

- **锁级别分类**
- **共享锁 & 排他锁 & 意向锁**



行级锁

表级锁

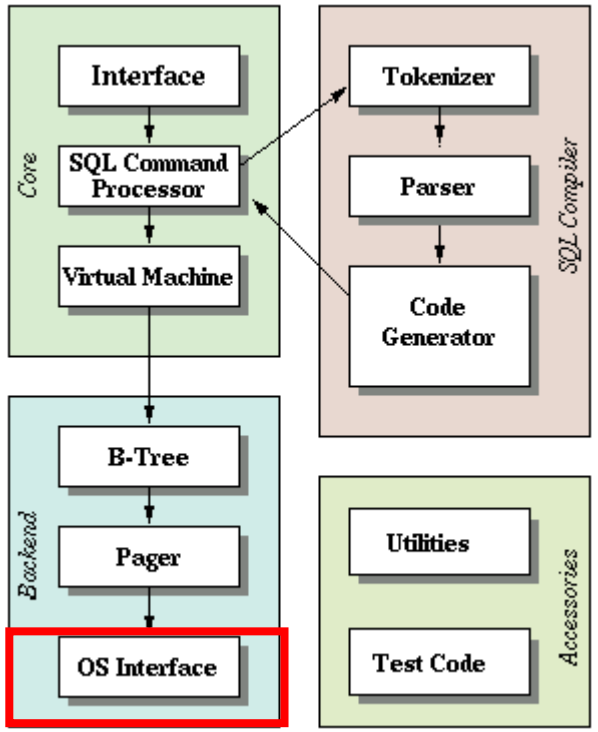
页级锁

共享锁

排他锁

意向锁

锁(操作系统级别)


Core
Interface
SQL Command Processor
Virtual Machine
SQL Compiler
Tokenizer
Parser
Code Generator
Backend
B-Tree
Pager
OS Interface
Accessories
Utilities
Test Code