

# 数据库个人理解

# 目录

- 1.数据库设计
- 2.事务处理
- 3.锁
- 4. InnoDB

# 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》

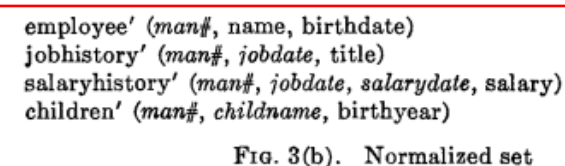
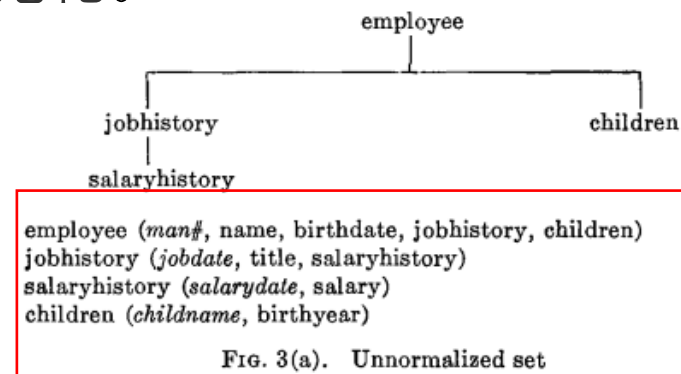
# 数据库设计

- 0.数据库设计三范式的来源（为什么要有范式）：

- Some more complicated data structure is 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



# 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	列2	列3	列4	列5	.....
----	----	----	----	----	-------

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) :

符合2NF ☒ (主键1, 主键2) 列1, 列2, 列3, 列4  
非主键列全部依赖于主键

不符合2NF ☐ (主键1, 主键2) 列1, 列2, 列3, 列4  
非主键列全部依赖于部分主键

不符合2NF ☐ (主键1, 主键2) 列1, 列2, 列3, 列4  
非主键类部分依赖于全部主键

不符合2NF ☐ (主键1, 主键2) 列1, 列2, 列3, 列4  
非主键列部分依赖于部分主键

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。

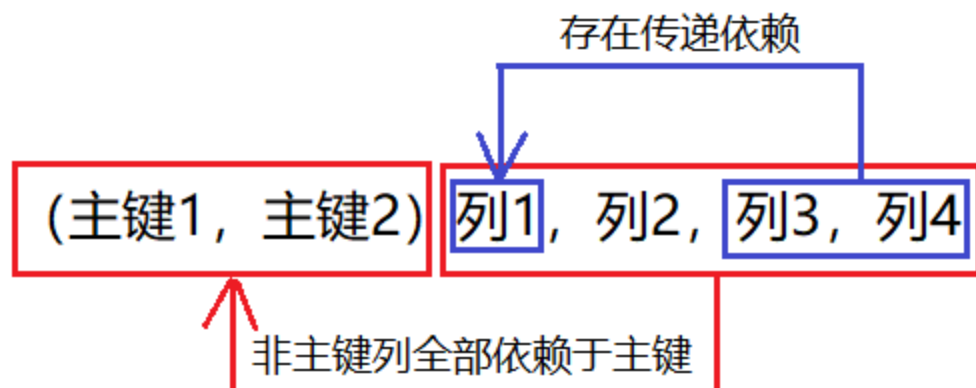
学号（主键）	成绩	绩点
0	100	5
1	100	4
2	90	3



### 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。



[https://blog.csdn.net/weixin\\_43971764](https://blog.csdn.net/weixin_43971764)

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

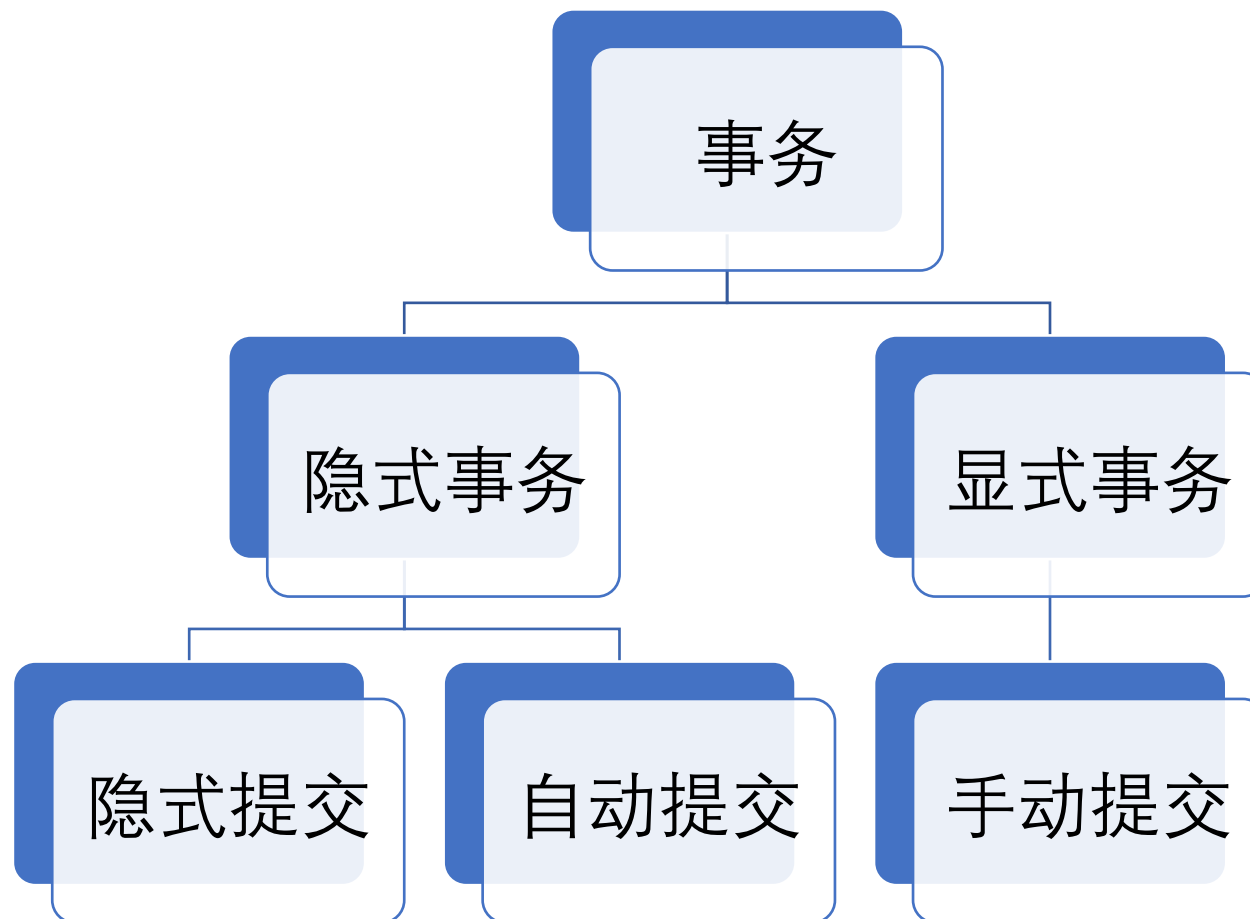
- 事务，是一批对数据库的操作；可以理解为批处理文件（windows的.bat/linux的.shell）
- 于此同时，事务由于mysql的单进程和多线程，也可以理解为在一个进程中的一个线程。
- 事务处理分为显式事务和隐式事务。

- 事务语法如下：

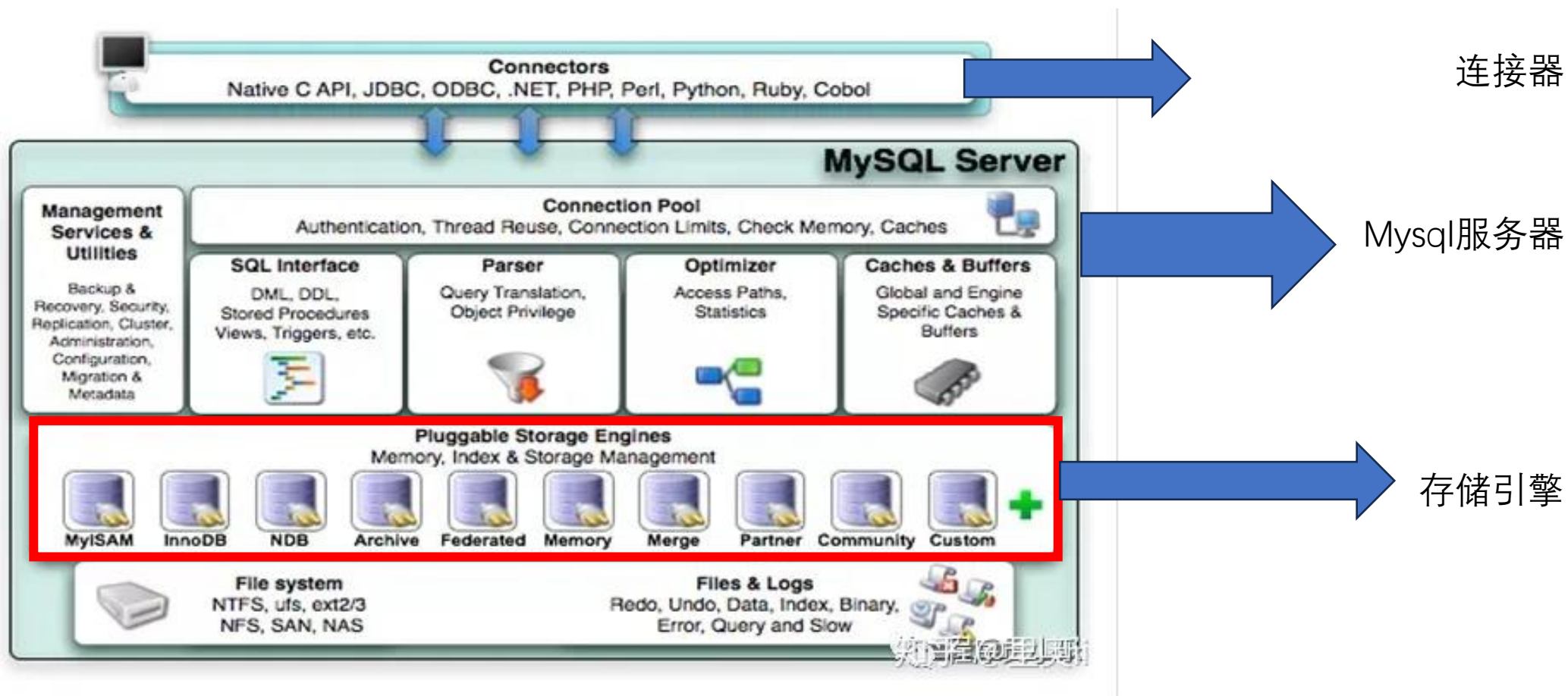
- BEGIN;
- 你要执行的sql语句;
- COMMIT;

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

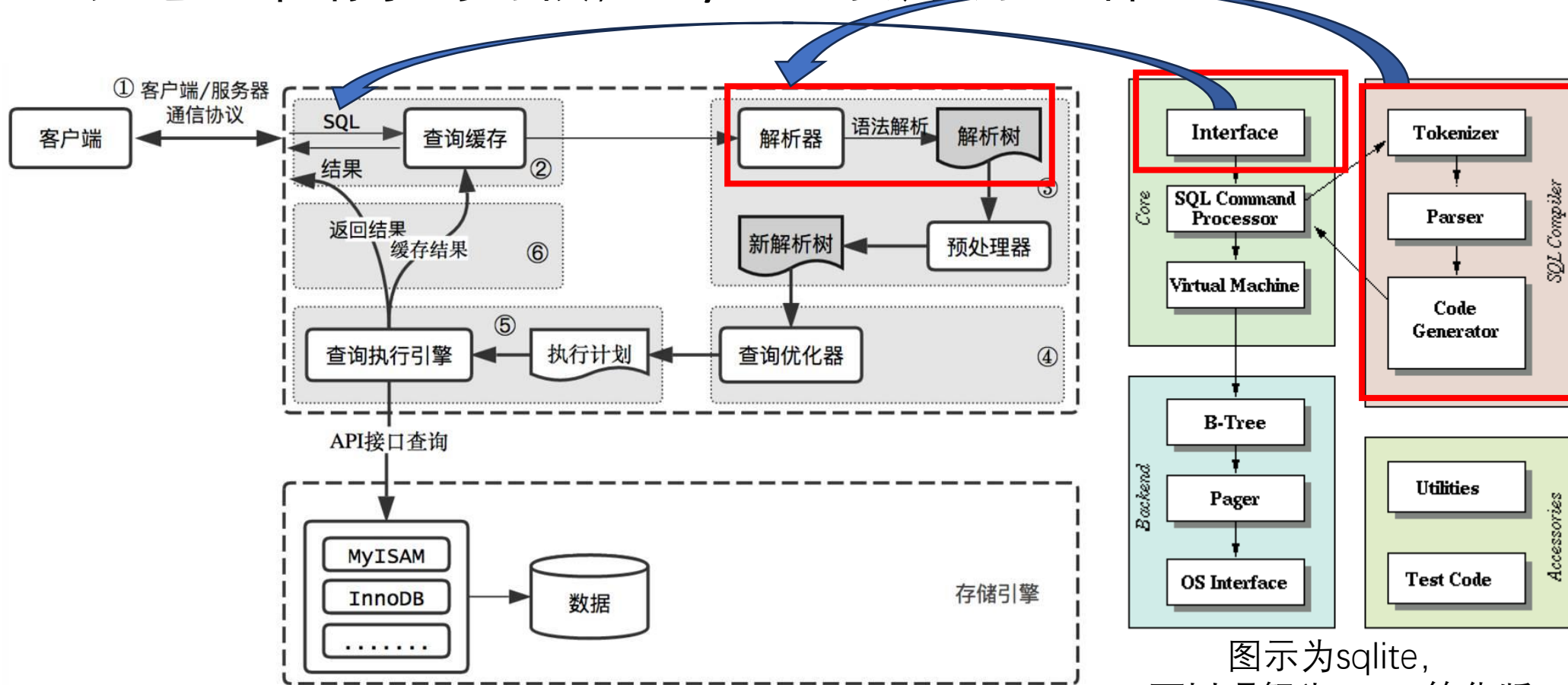


# 事务处理-知识之mysql运行原理



# 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

[illegible]

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

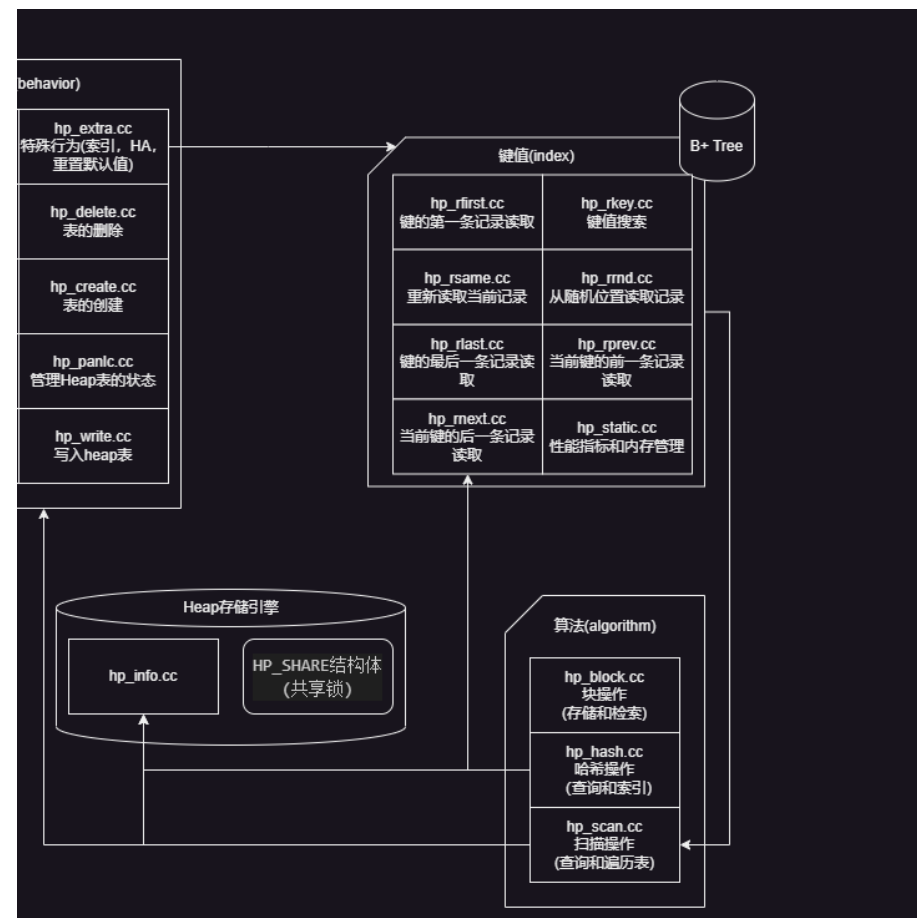
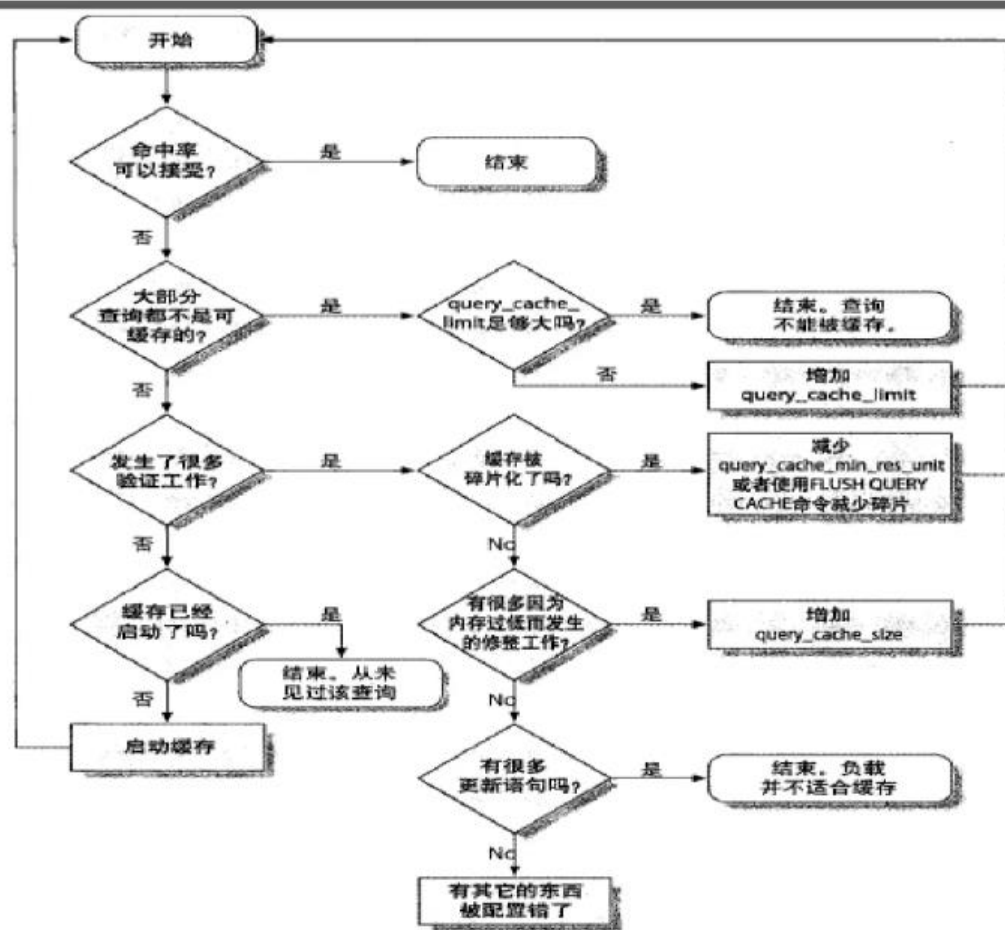
这个东西其实已经由各大 connectors([MySQL :: MySQL Connectors](#))实现了，这里只是复现一下原理。

# 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

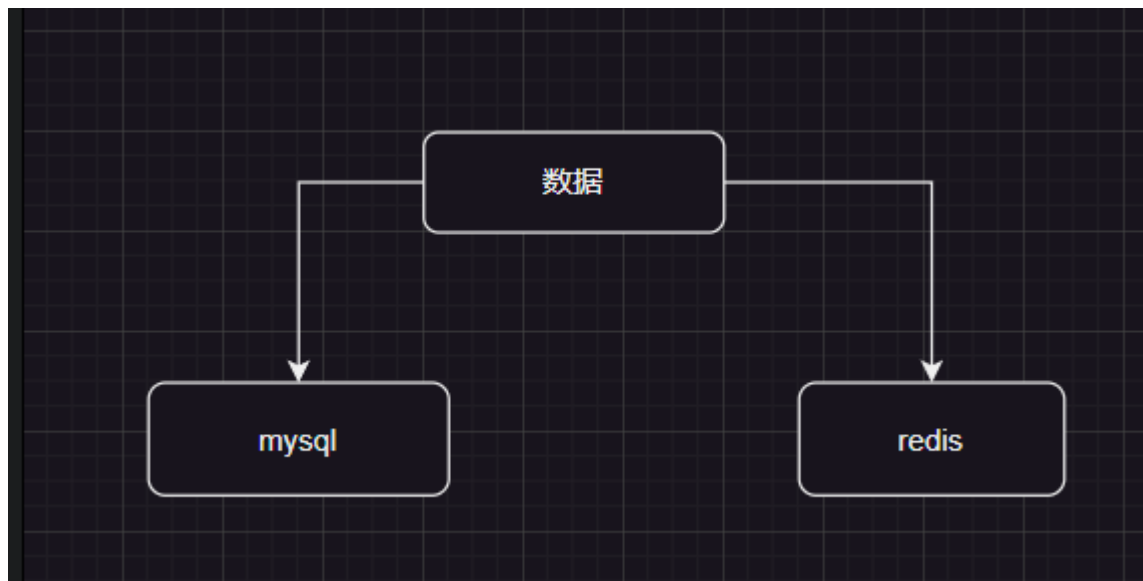


# 缓存查询与redis——mysql本地缓存(磁盘)

- 通过上图我们可以知道， Query Cache对数据库的读和写都会带来额外的消耗：
- 1) 读查询开始之前必须检查是否命中缓存。
- 2) 如果读查询可以缓存，那么执行完查询操作后，会查询结果和查询语句写入缓存。
- 3) 当向某个表写入数据的时候，必须将这个表所有的缓存设置为失效，如果缓存空间很大，则消耗也会很大，可能使系统僵死一段时间，因为这个操作是靠全局锁操作来保护的。
- 4) 对InnoDB表，当修改一个表时，设置了缓存失效，但是多版本特性会暂时将这修改对其他事务屏蔽，在这个事务提交之前，所有查询都无法使用缓存，直到这个事务被提交，所以长时间的事务，会大大降低查询缓存的命中

# Redis缓存(内存):

- redis是一个支持网络、可基于内存亦可持久化的日志型、Key-Value数据库。
- 所以，会有以下做法，即双写：



显然，其存在一个问题：  
**如何保证redis与mysql数据一致性？**

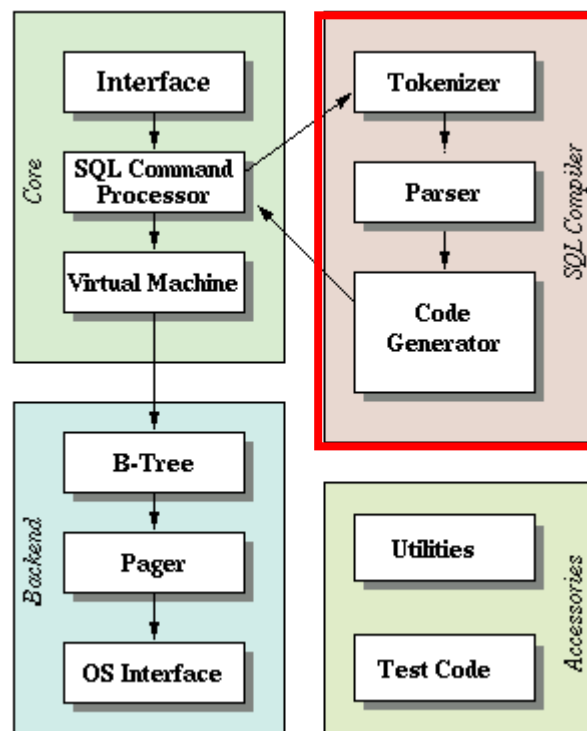
# dlc :关于redis缓存问题—本质上是I/O问题

- 缓存穿透---redis的key查不到。
- 解决方案：
- 无效的key也存进redis。
- 缓存雪崩---由于高并发导致数据库宕机，数据库被重启后继续被高并发流量进入。（可能redis宕机或双数据库采取了相同的过期时间）
- 解决方案：
- 1.失效时间+随机值
- 2.分库/表，读写分离
- 缓存击穿---同上，不过只是其中一个热点key，key失效了，并发到数据库上。

dlc2 :关于redis缓存问题-与数据库的一致性

# 解析器

- 将sql语句“读取”进去
- 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 (EXPLAIN TableName)

- 执行计划引擎接受一个查询执行计划，执行该计划并把返回结果给查询。
- 本质上是一种扫描sql语句的工具。
- 换言之，sql在进入存储引擎之前，需要先扫描一下运行时间
- 也就是这个东西

```
6 rows in set (0.00 sec)
```

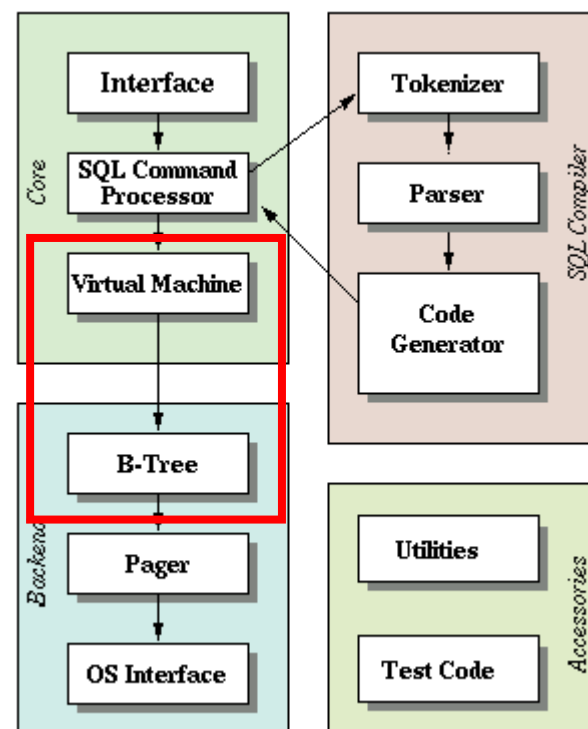


- 1、system: 系统表, 少量数据, 往往不需要进行磁盘IO;
- 2、const: 常量连接;
- 3、eq\_ref: 主键索引(primary key)或者非空唯一索引(unique not null)等值扫描;
- 4、ref: 非主键非唯一索引等值扫描;
- 5、range: 范围扫描;
- 6、index: 索引树扫描;
- 7、ALL: 全表扫描(full table scan);

# 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)



# innodb与myisam的区别

# 并发处理小能手-锁

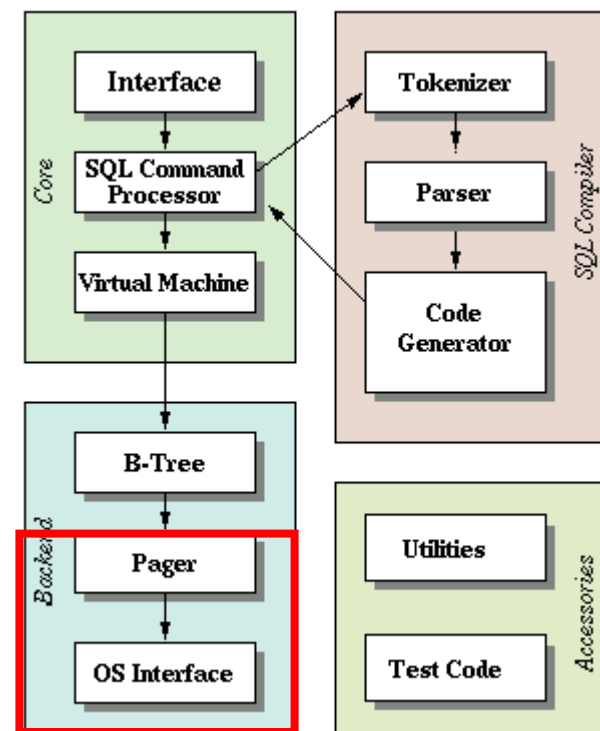
- 以下，我们讲讨论mysql在并发情况下的问题。

# 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。

# 锁介绍

- 在mysql中， 主要存在以下锁：
- **锁级别分类**
- **共享锁**
- 如果一个事务Ti获得了数据项Q上的共享模式锁，  
• 则Ti可以读Q， 但不能写Q。（事务可读不可写）
- **排他锁**
- 如果一个事务Ti获得了数据项Q上的排他模式锁，  
• 则Ti可以读Q也可以写Q。（事务可读也可写）



# 锁(操作系统级别)

