

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》

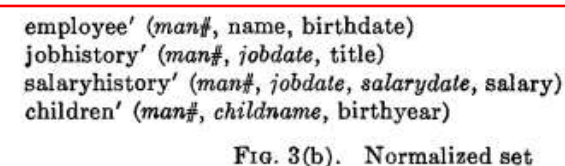
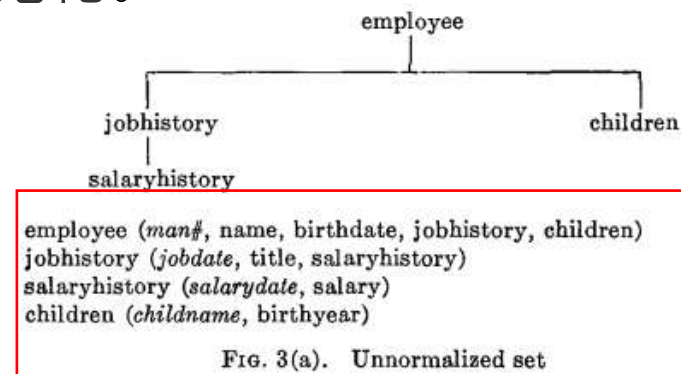
数据库设计

- 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



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。

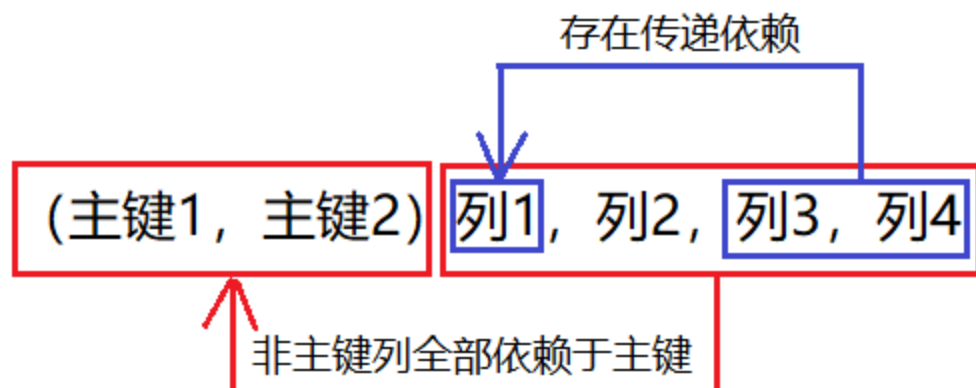
- 一是必须有一个 (及以上) 主键;
- 二是没有包含在主键中的列必须全部依赖于全部主键, 而不能只依赖于主键的一部分而不依赖全部主键。

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。



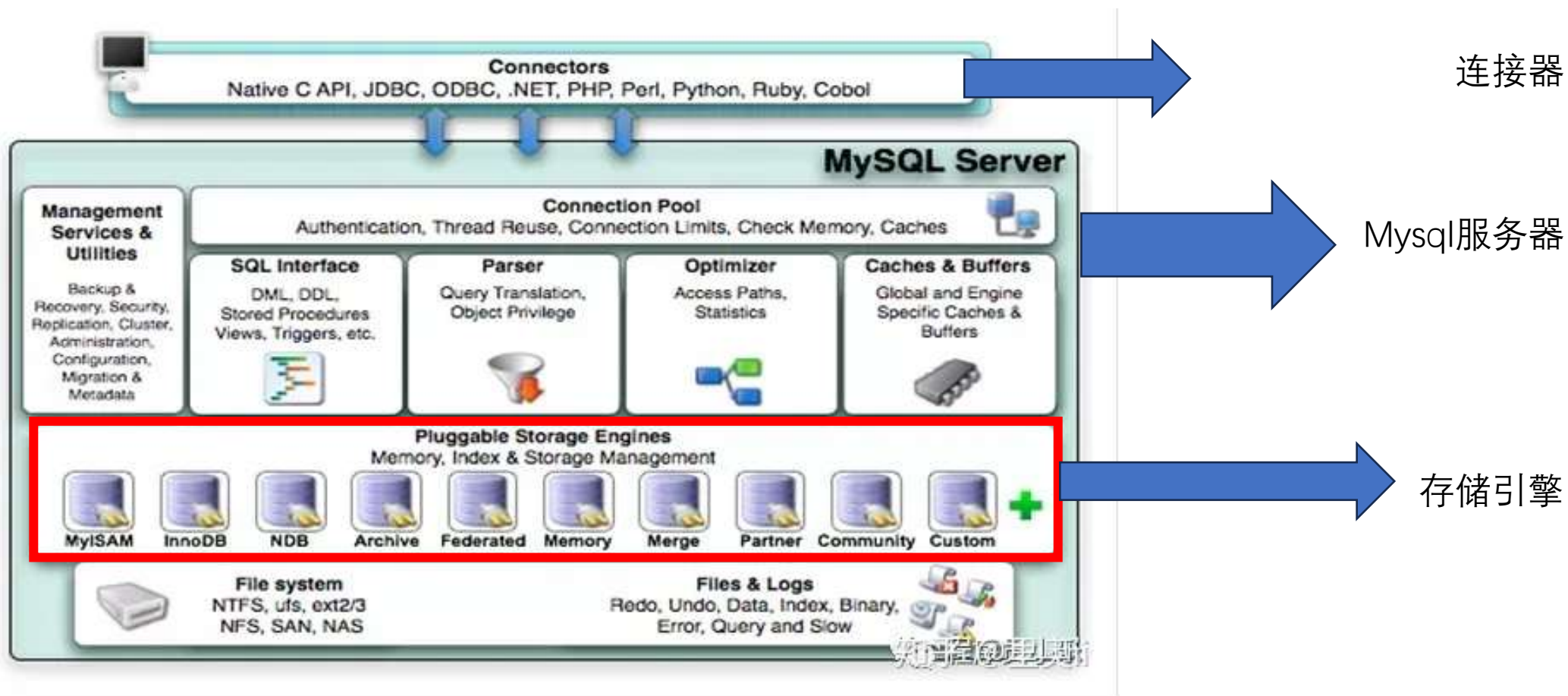
https://blog.csdn.net/weixin_43971764

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

但是…

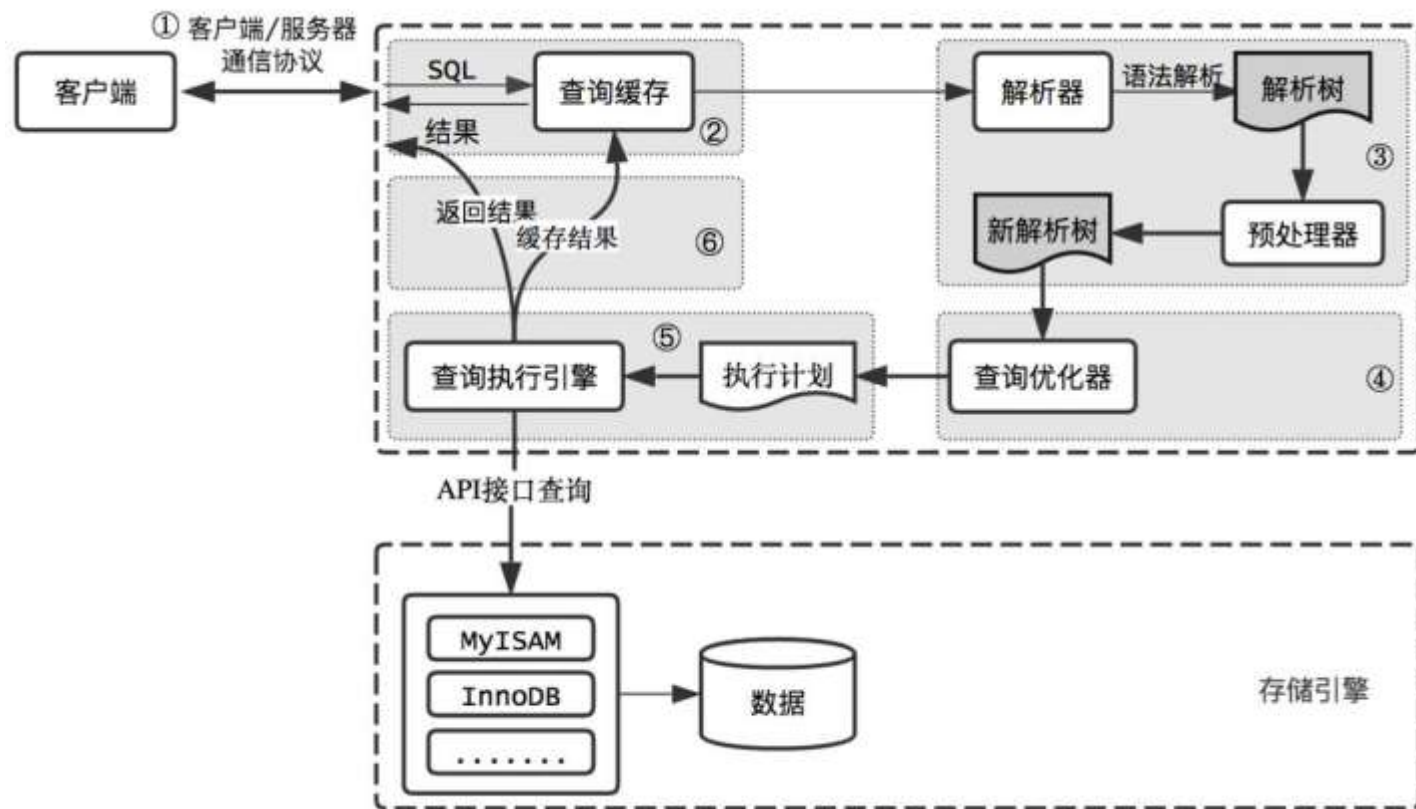
- 请注意：这里所说的所有有有关于“数据库范式”的内容均为一种“规范”，或者说是“通法”，而不是一板一眼的死板学识。
- 请设计数据库时遵循业务逻辑！不要为了遵循范式而强行遵循。
- 业务逻辑：老板让你建立一个表记录的内容类型（以老板的话为准）

事务处理-前置知识之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？



TCP/IP

```
#!/usr/bin/perl

import socket

# MySQL服务器地址和端口
mysql_host = '127.0.0.1'
mysql_port = 3306

# 创建一个TCP套接字
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    # 连接到MySQL服务器
    client_socket.connect((mysql_host, mysql_port))

    # MySQL登录信息
    username = 'root'
    password = 'password'
    database = 'test'

    # 构造MySQL登录请求
    login_request = {
        b'\x00': 0, # 协议版本
        username.encode('utf-8') + b'\x00': 1,
        b'\x1d': 0, # 数据库名称长度
        b'\x00': 21 + 0, # 填充
        password.encode('utf-8') + b'\x00': 3,
        database.encode('utf-8') + b'\x00': 4
    }

    # 发送登录请求
    client_socket.sendall(login_request)

    # 接收并打印服务器返回的响应
    response = client_socket.recv(1024)
    # 解析响应
    version_bytes = response[4:1]
    plugin_name_bytes = response[-len(b'caching_sha2_password'):1]
    version = version_bytes.decode('utf-8')
    plugin_name = plugin_name_bytes.decode('utf-8')

    print(f"版本号: {version}")
    print(f"加密插件名称: {plugin_name}")

except Exception as e:
    print(f"发生错误: {e}")

finally:
    # 关闭套接字
    client_socket.close()

# 输出结果示例
header = response[:4]
response_length = int.from_bytes(header, byteorder='little')

# 提取数据块
data_packet = response[4:]

# 打印解密后的数据内容
print(f"数据包长度: {response_length}")
print(f"数据包内容: {data_packet}")
```

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

缓存查询

- Mysql8.0已经废除;
- 但我们可以搞一个简易版本(已经上传至github)

事务处理(未写完)

- 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。
- 显式事务语法如下：
 - BEGIN;
 - 你要执行的sql语句;
 - COMMIT;

