从零实现数据库源码分析

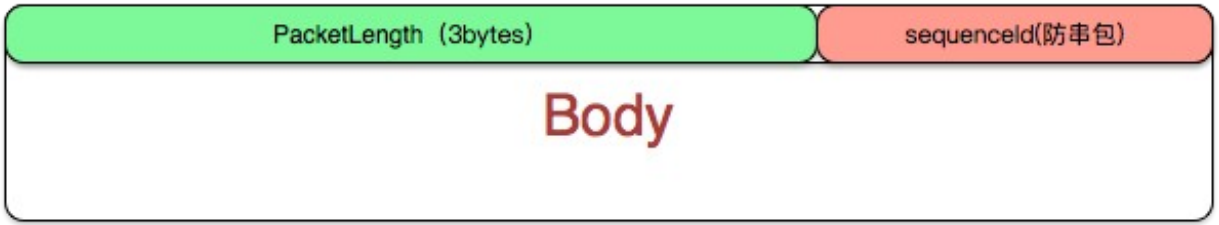
总览

- 网络协议层
  - MySQL 客户端登录认证
- sql 解析层
  - 逻辑执行计划
  - 物理执行计划
- 数据存储层
  - page cache 页缓存
  - 读写锁与事务
  - join 算法

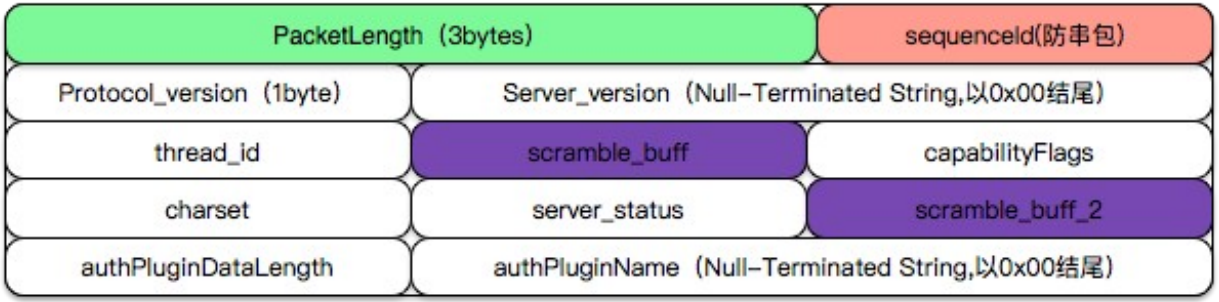
mysql 协议

- 报文分为消息头和消息体两部分，其中消息头占用固定的 4 个字节。
- 消息长度用来解决粘包与半包问题问题。
- 第四个字节为了防止串包用。机制是每收到一个报文都在其 sequenceId 上加 1，并随着需要返回的信息返回回去。如果 DB 检测到 sequenceId 连续，则表明没有串包。如果不连续，则串包， DB 会直接丢弃这个连接。
- 消息体则是最终传递信息的地方。

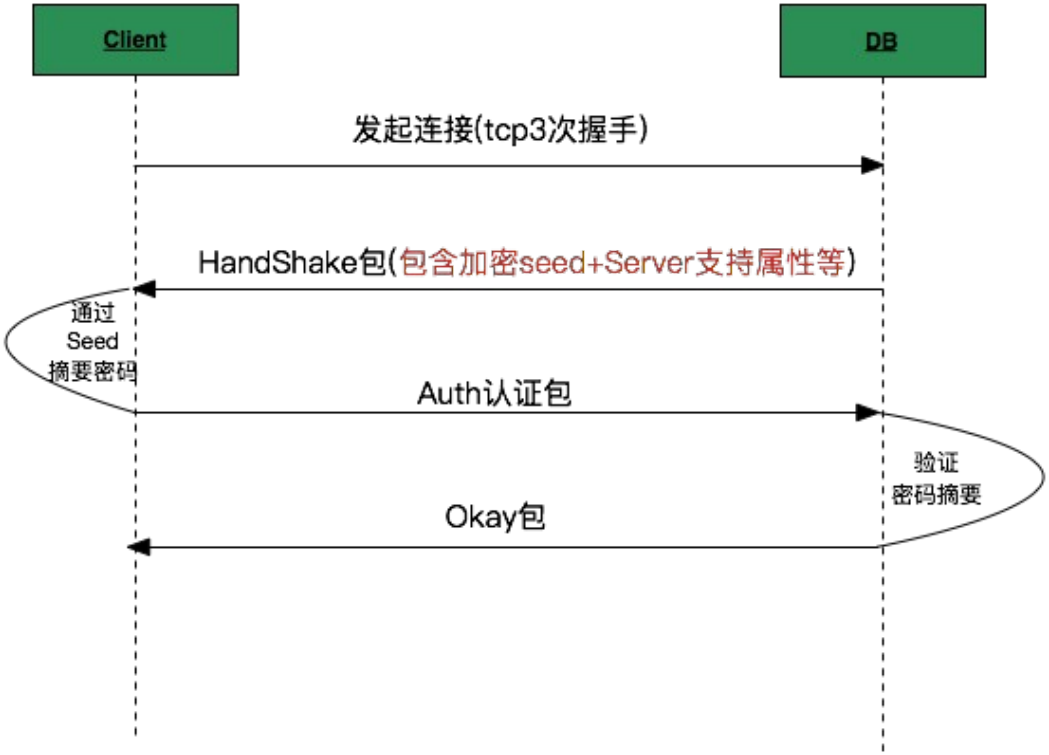
MySql包的外层结构如下:



以HandShake包为例，其具体结构如下:



登录过程

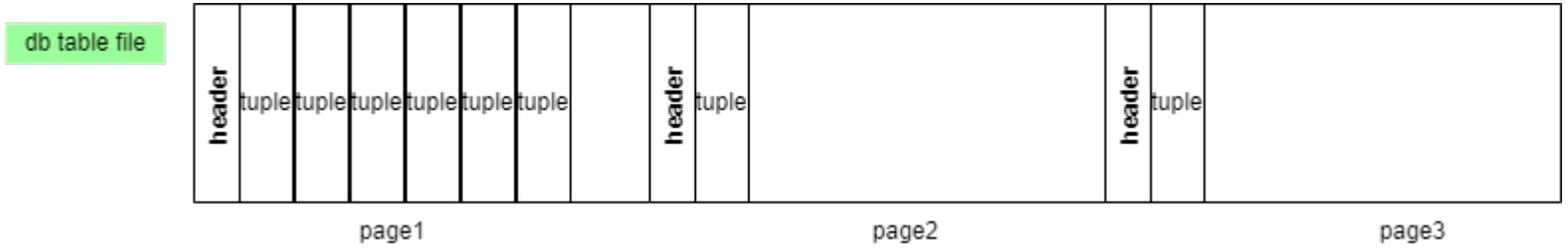

Client
DB
发起连接(tcp3次握手)
HandShake包(包含加密seed+Server支持属性等)
通过
Seed
摘要密码
Auth认证包
验证
密码摘要
Okay包

术语概述

- Database
  - 一个数据库实例
- Catalog
  - 类似 MySQL 中的库
- Table
  - 数据表
- Tuple
  - 一行数据
- TupleDesc
  - 对 tuple 的描述。可以理解为表结构，定义字段名和字段类型

insert 实现

- 文件存储
  - 数据文件由一系列线性 page 组成。每个 page 默认大小为 4KB，与磁盘页大小对齐。每一个 Page 中都有一个 header，是一个字节数组。 Page 是由一系列的 slot 组成的（slot 由 tuple 填充）。 header 中的每一位代表某一个 slot 是否有 tuple 。比如：如果 header 是 10010，代表第 1 个 slot 和第 3 个 slot 存储着 tuple，但是其他 slot 没有 tuple，只是一个空的 slot 。所以每一个 tuple 需要多余的一个 bit 的来存储。所以一个页能存储的 tuple 数量为： tupsPerPage = floor((BufferPool.PAGE_SIZE * 8) / (tuple size * 8 + 1)) 。计算出 tuplesPerPage 之后，我们就知道了需要用多少个字节来存储 header 。 headerBytes = ceiling(tupsPerPage/8) 。

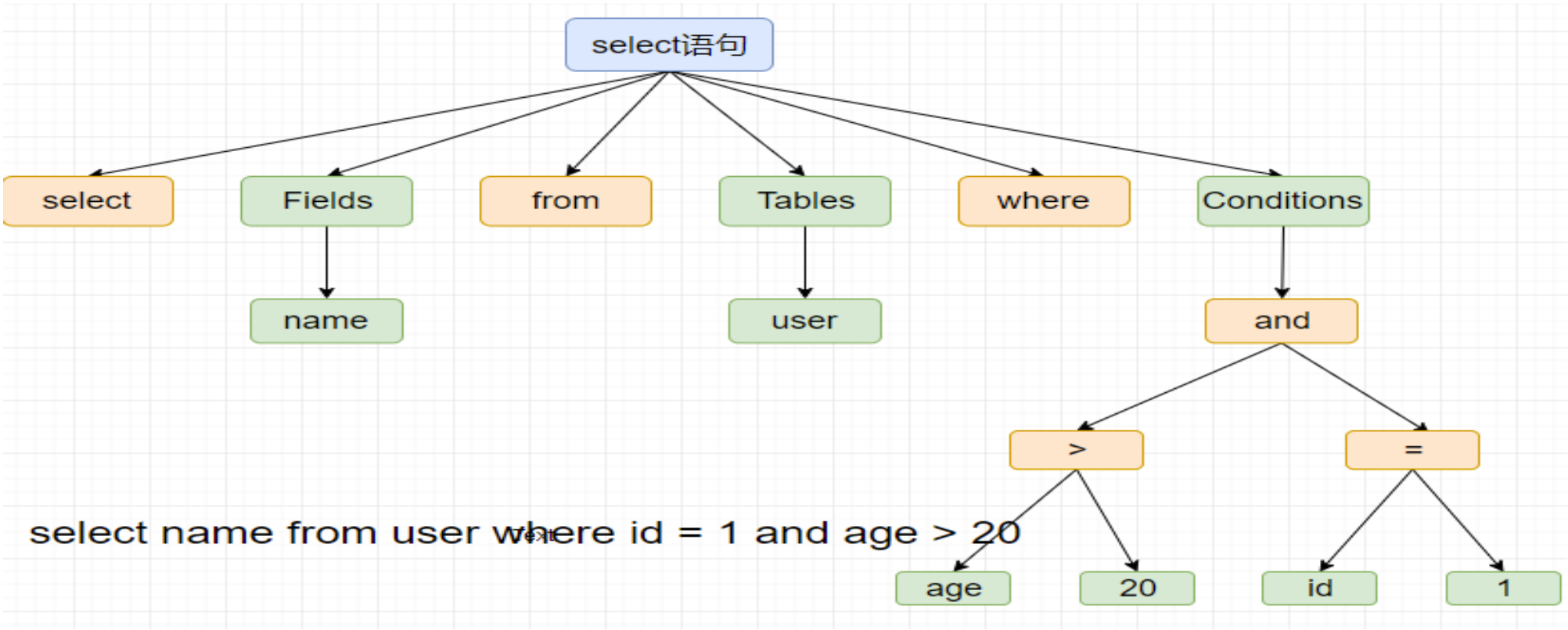


- page cache
  - page cache 对应数据文件中 page 的内容。读写数据时，先对 page cache 进行读写。如果没有命中 page cache，先去读磁盘中的数据缓存到 page cache，再进行后续操作。事务提交时，将脏页持久化到磁盘。
- 事务
  - 读操作时，对页加读锁。写操作时，对页加写锁。事务提交时释放锁。两个读操作不冲突，读写与两个写操作冲突。实现 read committed 隔离级别。

select 实现

- sql 解析为抽象语法树（ AST ）
- 通过 AST 生成逻辑执行计划（ logic plan ）
- 使用 CBO 优化器生成物理执行计划（ physical plan ）
- 获取数据，返回结果

抽象语法树


select语句
select
Fields
from
Tables
where
Conditions
name
user
and
>
=
age
20
id
1
select name from user where id = 1 and age > 20

CBO （基于成本的优化器）

- Predicate
  - 谓词，查询条件。
- Cardinality
  - 基数， Selection cardinality: SC(P, R) 。表示当 predicate 是 P 的时候，对于表 R ，通过数据分布直方图，最后大约会有多少条 row 输出。
- Selectivity
  - 选择性， cardinality 除以总行数。 cardinality=NUM_ROWS*selectivity
- join 成本
  - joincost(t1 join t2) = scancost(t1) + ntups(t1) x scancost(t2) //IO cost + ntups(t1) x ntups(t2) //CPU cost

join order

join order 分为 left deep tree 和 bushy tree 。 left deep tree 处理起来更简单。

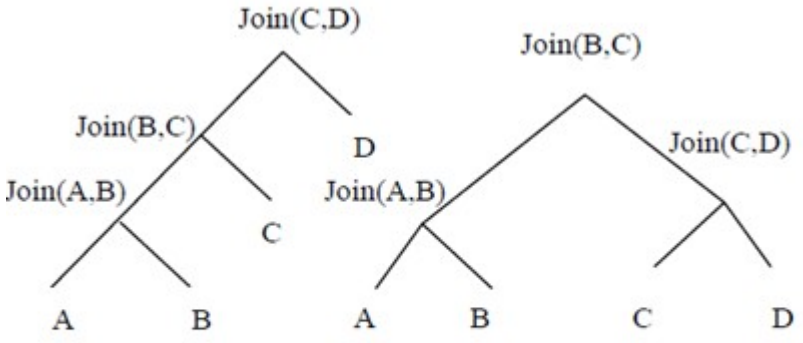


Figure 2. (a) Linear and (b) bushy join

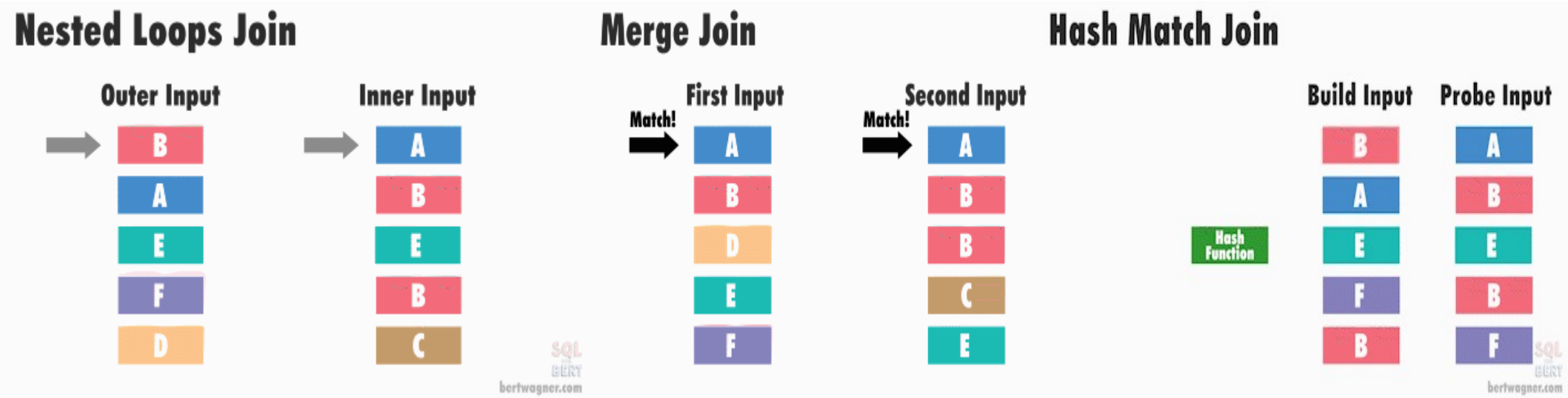
动态规划

- (a:b:c:d)
- (b:c:d):a (a:c:d):b (a:b:d):c (a:b:c):d
- (c:d):b (b:d):c (b:c):d
- c:d d:c

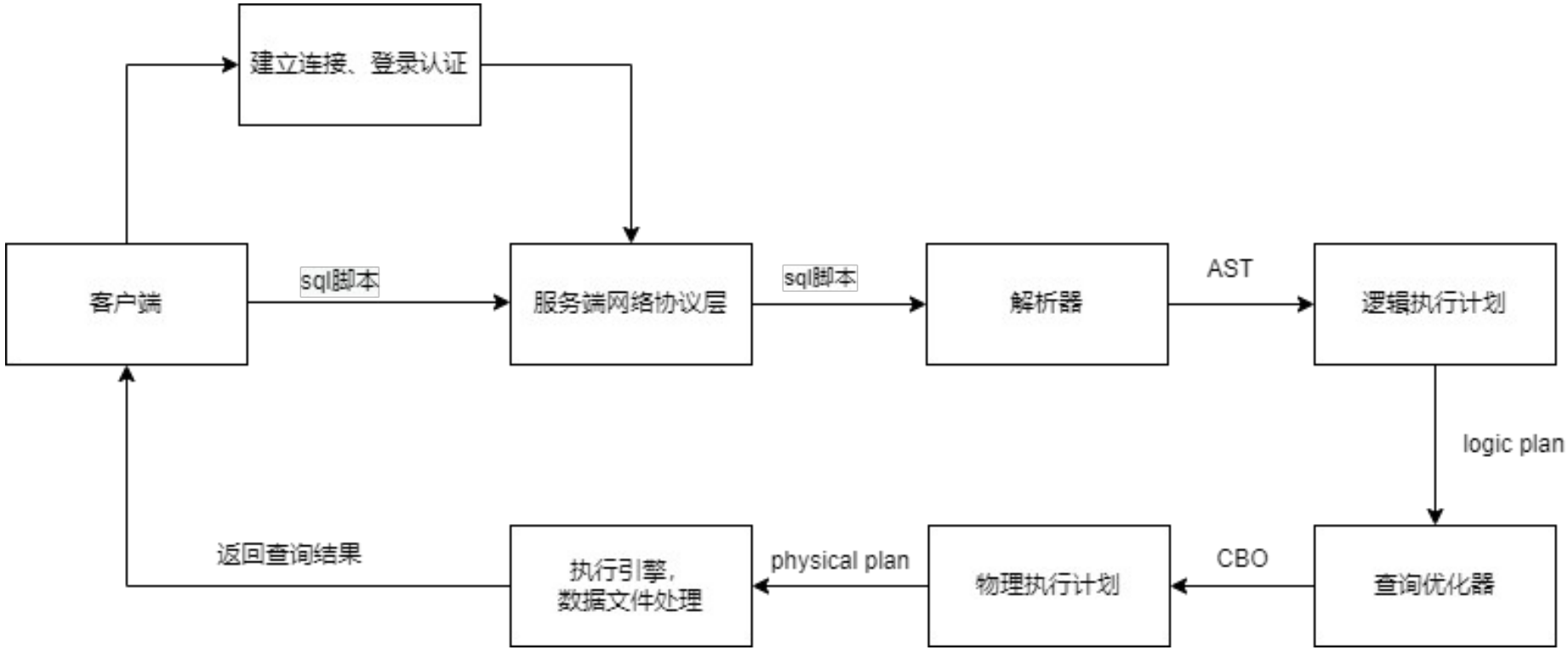
将一个问题拆成几个子问题，分别求解这些子问题，即可推断出大问题的解。

join 算法

- NestedLoopJoin
- BlockNestedLoopJoin
- SortMergeJoin
- HashJoin



总结


建立连接、登录认证
客户端
sql脚本
服务端网络协议层
sql脚本
解析器
AST
逻辑执行计划
logic plan
查询优化器
CBO
物理执行计划
physical plan
执行引擎，
数据文件处理
返回查询结果