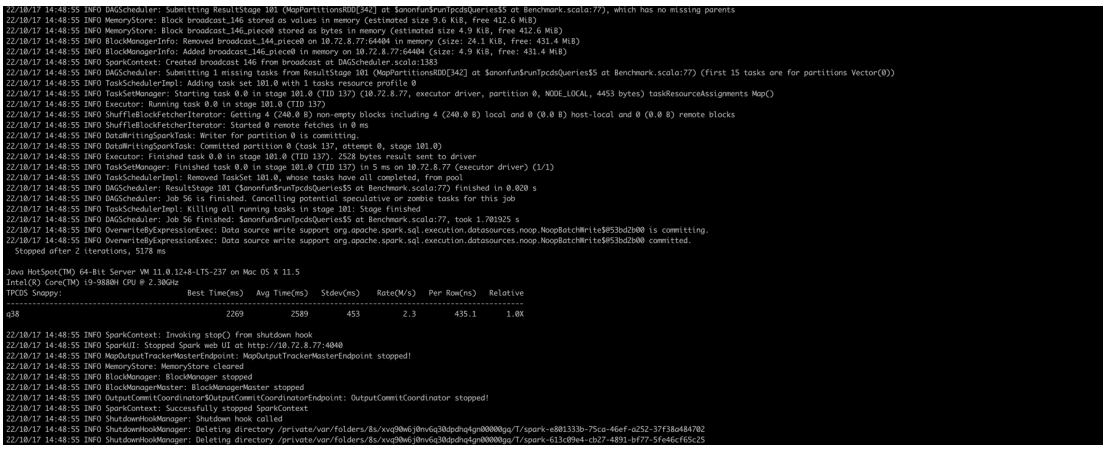
作业1:

- q38.sql

```
SELECT count(*)
FROM (
       SELECT DISTINCT
         c_last_name,
         c_first_name,
         d_date
       FROM store_sales, date_dim, customer
       WHERE store_sales.ss_sold_date_sk = date_dim.d_date_sk
         AND store_sales.ss_customer_sk = customer.c_customer_sk
         AND d_month_seq BETWEEN 1200 AND 1200 + 11
       INTERSECT
       SELECT DISTINCT
         c_last_name,
         c_first_name,
         d_date
       FROM catalog_sales, date_dim, customer
       WHERE catalog_sales.cs_sold_date_sk = date_dim.d_date_sk
         AND catalog_sales.cs_bill_customer_sk = customer.c_customer_s
         AND d_month_seq BETWEEN 1200 AND 1200 + 11
       INTERSECT
       SELECT DISTINCT
         c_last_name,
         c_first_name,
         d_date
       FROM web_sales, date_dim, customer
       WHERE web_sales.ws_sold_date_sk = date_dim.d_date_sk
         AND web_sales.ws_bill_customer_sk = customer.c_customer_sk
         AND d_month_seq BETWEEN 1200 AND 1200 + 11
     ) hot_cust
LIMIT 100
```

- 执行结果



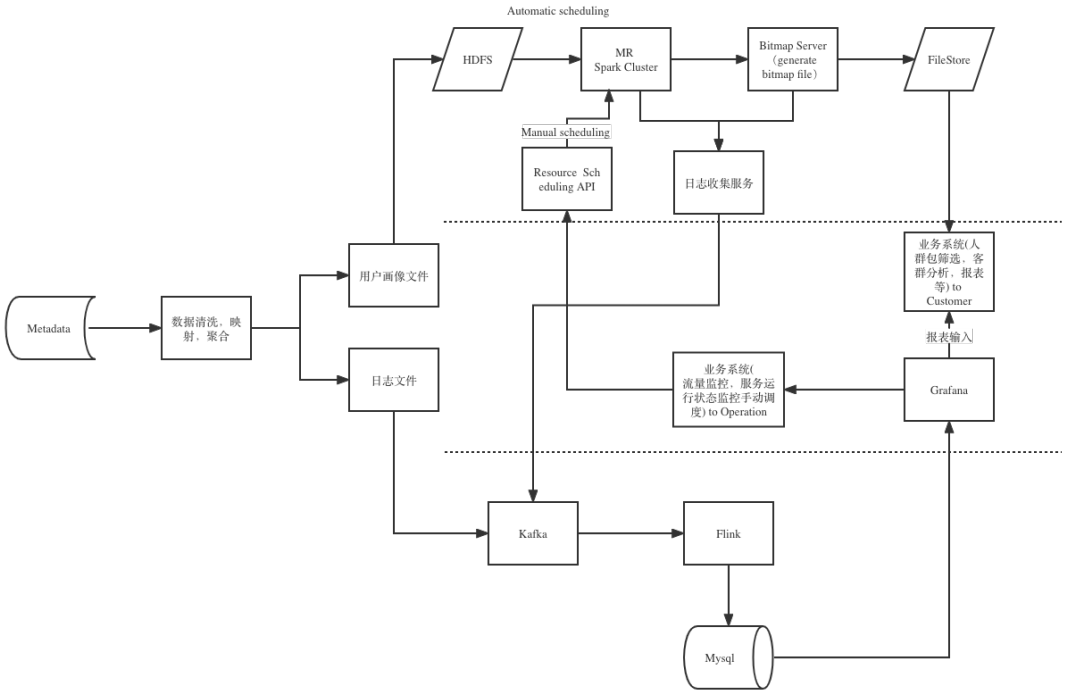
- 优化规则

```
CollapseProject
PushDownPredicates
ConstantFolding
EliminateLimits
ColumnPruning
InferFiltersFromConstraints
PushDownLeftSemiAntiJoin
RemoveNoopOperators
ReorderJoin
 ReplaceDistinctWithAggregate
 ReplaceIntersectWithSemiJoin
 RewritePredicateSubquery
```

- 优化规则描述
  - PushDownPredicates：指谓词下推逻辑优化器，谓词可以下推的前提：不影响查询结果，即要保证下推前和下推后两个sql执行得到的效果相同，包含三条规则：CombineFilters，PushPredicateThroughJoin，PushPredicateThroughNonJoin。CombineFilters就是把SQL语句中的where过滤条件，尽可能地放在执行计划靠前的地方，尽早地将不必要的数据过滤掉从而提高了整个执行计划的效率；PushPredicateThroughJoin在完成join后，如果对join的结果进行过滤操作，会把过滤条件尽量下推到数据源读取时，减少数据传输和join时的数据量，从而提升性能；PushPredicateThroughNonJoin是指非join情况的谓词下推的优化器，主要适用于Filter节点子节点为Project、Aggregate等情况。
  - ConstantFolding：即常量折叠，我们在select 语句中，如果掺杂了一些常量表达式，ConstantFolding 规则会自动地用等效文本值静态计算的结果来替换计算的表达式。例如上述语句中有三个纯常量运算表达式，即 d_month_seq BETWEEN 1200 AND 1200 + 11。一旦数据量很大的时候，每行都要计算一次该表达式的值，积少成多就会浪费了很多的时间。所以通过常量折叠可以将它预先这个表达式转化为d_month_seq BETWEEN 1200 AND 1211，这样就可以消除很多不必要的重复计算，从而提升sql语句的执行效率。

作业2:

1. 架构图



2.Lambda 的优缺点

Lambda 架构是一种数据处理架构，通过利用批处理和流处理方法来处理大量数据。这种架构方法通过使用离线批处理来提供批处理数据的全面和准确的视图，同时使用实时流处理来提供在线数据的视图来平衡延迟、吞吐量和容错性，两个视图输出可以在呈现之前合并。

Lambda 架构包含三层，分别是 Batch Processing Layer（批处理层）、Speed (Real-Time) Processing Layer（速度处理层）、Serving Layer（服务层）。

优点：

1. 架构比较简单，实时层计算量比较小，计算成本可控，整个计算系统稳定。
2. 在某些情况下，实时计算和离线计算可分开进行，避免计算高峰，能够缓解资源压力。
3. 对数据修正纠错也很友好，能够保证实时层处理出现问题之后整个计算数据不被破坏，可以通过重新运行离线任务，从而很快的将历史数据修正，保证最终一致性。

缺点：

1. 需要同时维护两套系统架构：批处理层和速度层，业务需要的不同需要分别编码维护，数据源的任何变化均涉及到两个部署的修改，任务量大，难以灵活应对。
2. 同时维护两个复杂的分布式系统，并且需要保证他们逻辑上产生相同的结果输出到服务层中，两系统可能同时需要占用资源，对资源需求大。
3. 部署复杂，需要部署离线及实时计算两套系统，给运维造成的负担比较重。
4. 随着数据量的急剧增加、批处理窗口时间内可能无法完成处理，并且对存储也会有巨大的挑战。