**一：基础知识理解**

**1：简述一下什么是hadoop ？**

Hadoop的简单概述以及架构

hadoop概述：hadoop是一个适合海量数据分布式存储和分布式计算的平台。

hadoop是一个统称，hadoop主要包含三大组件：

（1）hdfs：是一个分布式存储框架，适合海量数据存储

（2）mapreduce：是一个分布式计算框架，适合海量数据计算

（3）yarn：是一个资源调度平台，负责给计算机框架分配计算资源

**2：简述一下 MapReduce工作机制?**

MapReduce是一种用于大规模数据处理的编程模型，其核心机制将任务分解为Map和Reduce两个阶段，并自动处理分布式计算细节。以下是其工作机制的简述：

输入分片（Input Splitting）‌

输入数据被划分为固定大小的‌分片（Split）‌，每个分片由一个Map任务处理。分片大小通常与HDFS块大小（如128MB）对齐，以优化数据本地性。

Map阶段‌

Map任务‌读取分片数据，通过InputFormat解析为键值对（如行号-文本），并交由用户编写的map()函数处理，生成中间键值对（如单词-计数1）。

内存缓冲区‌：Map输出先存入内存，达到阈值后‌溢写（Spill）‌到磁盘，期间进行‌分区（Partitioning）‌（按Reduce任务数哈希分区）、‌排序（Sorting）‌（按键排序）及可选‌合并（Combiner）‌（局部Reduce以减少数据量）。

Shuffle与排序（数据传输阶段）‌

Shuffle‌：各Reduce任务从所有Map节点拉取对应分区的数据（可能跨网络）。

归并排序‌：Reduce端对多Map输出的数据进行合并排序，确保相同键的数据连续排列。

Reduce阶段‌

用户定义的reduce()函数处理排序后的键及其值集合（如累加单词计数），生成最终结果。

输出通过OutputFormat写入HDFS，通常每个Reduce任务生成一个文件（如part-r-00000）。

容错与调度‌

主节点（如YARN的ResourceManager）‌协调任务分配与监控，‌工作节点（NodeManager）‌执行具体任务。

任务失败时自动重试，利用数据副本或重新执行确保可靠性。

关键组件‌：

Combiner‌：Map端的本地聚合优化，需满足结合律（如求和）。

Partitioner‌：决定中间数据的Reduce分配策略（默认哈希）。

RecordReader/Writer‌：解析输入数据与格式化输出。

MapReduce通过并行化与自动化的任务管理，高效处理海量数据，适用于日志分析、索引构建等场景，其设计核心在于简化分布式计算的复杂性。

**3：SPARK 中什么是宽依赖，什么是窄依赖？哪些算子是宽依赖，哪些是窄依赖？**

宽依赖：是指1个父RDD分区对应多个子RDD的分区

窄依赖：是指一个或多个父RDD分区对应一个子RDD分区

**4：简述数仓分层,每个层的所要建设的内容，数仓分层的意义是什么？**

解耦合，高复用，标准化

**5：星型模型和雪花模型区别**

雪花模型去除了冗余，设计复杂，可读性差，关联的维度表多，查询效率低，但是可扩展性好。  
星型模型冗余度高，设计简单，可读性高，关联的维度表少，查询效率高，可扩展性低。

星型模型：当所有的维度表都是和事实表直接相连的时候，整个图形看上去就像是一个星星，我们称之为星型模型。星型模型是一种非正规化的架构，因为多维数据集的每一个维度都和事实表直接相连，不存在渐变维度，所以有一定的数据冗余，因为有数据的冗余，很多的统计情况下，不需要和外表关联进行查询和数据分析，因此效率相对较高。

6 数仓中拉链表，全量表，快照表分别是什么样的？优缺点是什么？分别用在什么场景？

### **（1）全量表**

‌定义‌：存储截至当前的最新全量数据，每次更新直接覆盖旧数据，不保留历史记录。

‌特点‌：

数据完整性：始终反映最新状态的所有数据。

简单高效：无分区设计，更新时直接替换旧数据。

‌缺点‌：无法追溯历史变化，仅适合数据量小或更新低频的场景

‌适用场景‌：用在数据量不大，并且仅需保存最新状态即可的场景。

（2）快照表

‌定义‌：按日分区存储，每个分区保存截止对应日期的全量数据（如 分区=2025-05-14 包含当日所有数据）。

‌特点‌：

历史可追溯：通过分区可回溯任意日期的数据状态。

数据冗余：每个分区均为全量副本，存储成本高。

‌适用场景‌：用在必须需高频查询历史状态的场景。

（3）拉链表

‌定义‌：记录数据从开始到当前状态的所有变化轨迹，通过字段标记有效期（如 start\_date 和 end\_date）14。

‌特点‌：

‌极限存储‌：仅存储变化数据，未变化的记录不重复保存14。

‌高效查询‌：通过时间范围筛选获取指定日期的全量数据（如 WHERE start\_date <= '2025-05-14' AND end\_date >= '2025-05-14'）6。

‌实现‌：通过历史表与增量表关联生成新版本记录，旧版本关链（end\_date 设为更新日期）。

‌适用场景‌：数据频繁变化但需长期追踪历史状态变更轨迹的场景。

二：实际问题处理

**1：分布式计算时，产生数据倾斜的根本原因是什么？有哪些不同的表现原因，分别怎么解决？**

**mr 和 Spark 的数据倾斜分别会发生在什么阶段？**

（1）数据倾斜的核心是‌数据分布不均引发负载失衡‌。在分布式计算中，由于分区策略（如哈希）导致大量相同Key的数据被分配至同一节点，该节点需处理远超其他节点的数据量，形成长尾效应，最终表现为任务执行缓慢或内存溢出（OOM）

（2）数据倾斜的根本原因‌

数据倾斜的核心是‌数据分布不均引发负载失衡‌。在分布式计算中，由于分区策略（如哈希）导致大量相同Key的数据被分配至同一节点，该节点需处理远超其他节点的数据量，形成长尾效应，最终表现为任务执行缓慢或内存溢出（OOM）。



（3）MR与Spark倾斜阶段的差异‌

MapReduce（MR）‌

主要阶段‌：Reduce阶段（Shuffle后相同Key集中处理）。

次要阶段‌：Map阶段（不可切分大文件被单任务处理）。

Spark‌

Shuffle阶段‌：GROUP BY、JOIN等操作触发数据重分布，产生倾斜（如Reduce任务负载不均）。

Map阶段‌：输入数据源倾斜（如HDFS文件分布不均或RDD分区策略缺陷）。

核心差异总结‌

MR的倾斜更集中于Reduce阶段，Spark因内存计算特性可能在更多环节（如RDD转换）暴露倾斜问题。

Spark可通过内存优化（如调整Executor堆内存）缓解OOM，而MR依赖磁盘I/O，容错成本更高。

**2：在离线数据同步（t+1）时经常会出现数据漂移，请解释什么是数据漂移，分别描述在不同场景下，如何分别解决数据漂移问题？**

数据漂移指在离线数据同步（T+1）过程中，数据因时间戳、分区策略或业务逻辑不一致，导致数据被错误划分到非目标时段或分区中，造成数据冗余、缺失或不一致的问题。常见于分布式数据库分区间数据迁移或跨时间分区的数据同步场景。

数据漂移的典型场景及解决方案‌

‌场景一：时间戳字段不一致‌

‌问题根源‌：数据库表时间戳（modified\_time、log\_time）、业务过程时间（proc\_time）与数据抽取时间（extract\_time）存在差异，可能导致数据被错误归入相邻时间段（如11日数据混入12日分区）。

‌解决方案‌：

‌冗余提取与精准筛选‌：扩大数据提取范围（如跨天冗余提取15分钟数据），结合实际的业务时间（如proc\_time）过滤目标数据。

‌记录首次变更‌：对跨天冗余数据按modified\_time升序排序，取每条记录首次变更的版本，避免多次更新导致的归属混乱。

‌场景二：分区键变更‌

‌问题根源‌：数据因分区键（如订单ID、用户ID）更新导致跨分区迁移（如原分区1的数据变更后归属分区2），引发数据归属异常。

‌解决方案‌：

‌分区策略优化‌：采用稳定的复合分区键（如主键+时间戳），避免单一字段变更引发漂移。

‌数据重分布校验‌：同步后对比源与目标分区的数据量及关键字段分布，发现异常后触发重分布流程。

‌场景三：文件同步延迟或丢失‌

‌问题根源‌：文件传输过程中因网络延迟或压缩格式不可分割（如GZIP），导致部分数据未被及时同步或完整加载。

‌解决方案‌：

‌可分割压缩格式‌：改用支持分片读取的压缩格式（如Snappy），确保文件可分块处理。

‌校验文件机制‌：附加校验文件（记录数据量、文件大小等），目标系统加载前验证完整性。

‌场景四：跨系统时间差‌

‌问题根源‌：源系统与目标系统时钟不同步，导致数据按错误的时间戳切分（如源系统时间滞后导致数据被归入未来分区）。

‌解决方案‌：

‌统一时钟源‌：部署NTP服务器强制系统时钟同步。

‌逻辑时间校正‌：在ETL流程中，基于业务时间（如proc\_time）而非系统时间划分数据。

**3：假设你现在进入一个新的数据组，给你分配了一个数仓开发的任务，你的工作思路是什么？**

1. 快速了解业务流程，找对应的业务老师，或者数据老师，了解业务背景，业务流程。
2. 了解每个业务流程产生的数据表，了解这些数据表结构，存储方式，数据量等等。
3. 了解项目上数据开发规范，以保持一致。
4. 按需求进行数据开发。

**4：维度建模的流程**

（1） 概念建模：定义业务范围与核心元素‌

‌目标‌：明确业务需求，识别关键业务过程及数据实体。  
‌步骤‌：

‌业务过程梳理‌：确定分析的业务场景（如销售订单、用户行为）。

‌粒度定义‌：明确事实表中每条记录的最小业务单元（如订单中的单个商品项）。

‌事实与维度识别‌：

‌事实‌：可度量的数值型业务指标（如销售额、订单数量）。

‌维度‌：描述业务过程的上下文信息（如时间、地点、产品）。

（2）逻辑建模：设计维度模型结构‌

‌目标‌：基于概念模型构建模型模型，定义表结构与关联关系。  
‌步骤‌：

‌维度表设计‌：

包含主键（代理键或自然键）及描述性属性（如商品名称、分类层级）。

‌事实表设计‌：

包含外键（关联维度表）、度量字段（如金额、数量）及退化维度（如订单号）。

‌模型验证‌：检查业务指标是否可正确计算，维度是否覆盖所有分析场景。

（3）物理建模：实现技术适配与优化‌

‌目标‌：将逻辑模型转换为物理存储结构，平衡性能与存储成本。  
‌步骤‌：

‌表结构定义‌：

字段类型（如整数、字符串）、索引（如时间字段索引）及分区策略（按日期分区）。

‌性能优化‌：

‌预聚合‌：针对高频查询创建汇总表（如按日统计销售额）。

‌数据压缩‌：选择列式存储格式（如Parquet）减少存储占用。

‌同步策略‌：

根据数据变化频率选择全量、增量或新增及变化同步策略（如全量同步静态维度表）

**5：根据上述思路，请你简述一下电商订单主题表的设计思路和字段。**

第一步：根据对业务理解和学习复现电商下单流程**‌**

（1）‌商品浏览与选择‌

‌商品浏览‌：用户通过搜索、分类导航或推荐列表查看商品详情，包括价格、规格、评价等信息。

‌加入购物车‌：选定商品后加入购物车，支持调整数量、删除商品或接收平台推荐商品。

‌（2）下单与结算‌

‌结算页入口‌：用户从购物车进入结算页，确认最终购买商品清单。

‌填写订单信息‌：

‌收货信息‌：选择或新增收货地址、联系人及电话。

‌优惠选择‌：使用优惠券、积分抵扣或参与满减活动。

‌支付方式‌：选择支X宝、微信支付、银行卡等支付渠道。

‌生成订单‌：提交后生成唯一订单号，系统保存商品价格快照（防止价格波动影响订单）。

‌（3）支付流程‌

‌支付跳转‌：用户跳转至第三方支付平台完成支付操作。

‌状态变更‌：支付成功后订单状态从“待支付”变更为“已支付”；若超时未支付，订单自动关闭。

‌（4）订单处理与拆单‌

‌库存校验‌：系统检查商品库存，若缺货则通知用户取消或延迟发货。

‌拆单逻辑‌：根据商家归属、仓库分布或物流限制（如包裹重量）自动拆分订单。

‌状态流转‌：订单状态依次变更为“待发货”“已发货”等，并与WMS（仓库管理系统）、TMS（物流管理系统）联动。

‌（5）物流配送与签收‌

‌物流对接‌：订单信息同步至物流系统，生成物流单号并实时更新轨迹（如揽收、运输中、派送中）。

‌用户签收‌：用户收货后确认订单状态变更为“已签收”，超时未确认则自动完成。

‌（6） 售后与交易闭环‌

‌退货退款‌：用户发起退换货申请，商家审核通过后触发退款流程，订单状态变更为“退款中”“已退款”。

‌评价闭环‌：用户对商品和服务进行评价，订单最终状态标记为“已完成”。

**第二步：根据对上述业务过程的分析，进行建模设计**

（1）‌业务过程整合‌

以‌订单全生命周期‌为核心，整合下单、支付、拆单、物流、售后等环节的关键数据，形成宽表以支持多维度分析（如时效分析、用户行为追踪）。

合并高频关联维度属性（如商品类目、用户等级、物流公司）到宽表中，减少联表查询开销。

（2）‌缓慢变化维处理‌

记录关键维度属性的‌历史快照‌（如下单时的商品价格、用户地址），避免因业务数据变更导致分析失真。

（3）使用‌代理键‌关联维度表（如用户维度代理键、商品维度代理键），支持维度历史版本回溯。

（4）‌粒度与时效优化‌

定义宽表粒度为‌订单明细层级‌（如一个订单中多个商品对应多条记录），支持按商品SKU分析。

增加时间分区字段（如 order\_date），按日/月分区提升查询效率

（5）核心字段包含

事实字段











**三：编程思路：根据表结构编写SQL**

**1：假设 hive 表结构如下：cust\_pv 表**

**客户编号 当日浏览次数 当日点击次数 当日收藏次数 当日加购次数**

**cust\_id , vi\_pv , cl\_pv , co\_pv , pur\_pv**

**逻辑：**

**当日浏览次数大于10次，点击次数大于20次，收藏次数大于5次，当日加购次数大于2次**

**满足其中任意一种为'低意向',满足其中任意二种为'中意向',满足其中任意三种为 '中高意向',全部满足为'高意向'。**

**需求：撰写SQL计算每个人意向度。**

sql

-- 创建意向等级计算逻辑的CTE临时数据集

WITH condition\_calculation AS (

SELECT

cust\_id,

-- 统计满足各阈值条件的数量

(CASE WHEN vi\_pv > 10 THEN 1 ELSE 0 END) +

(CASE WHEN cl\_pv > 20 THEN 1 ELSE 0 END) +

(CASE WHEN co\_pv > 5 THEN 1 ELSE 0 END) +

(CASE WHEN pur\_pv > 2 THEN 1 ELSE 0 END) AS cnt

FROM

user\_behavior\_table

)

-- 根据条件数量映射意向等级

SELECT

cust\_id,

CASE

WHEN cnt >= 4 THEN '高意向' -- 全部满足

WHEN cnt = 3 THEN '中高意向' -- 满足任意三种

WHEN cnt = 2 THEN '中意向' -- 满足任意两种

WHEN cnt = 1 THEN '低意向' -- 满足任意一种

ELSE '未达标准' -- 未满足任何条件（可选逻辑）

END AS intention\_level

FROM

condition\_calculation;

逻辑说明

条件统计‌：通过CASE WHEN将每个字段的阈值条件转换为布尔值（满足为1，不满足为0），求和得到满足条件的总数 cnt。

等级映射‌：

高意向‌：所有条件均满足（cnt=4）。

中高意向‌：满足任意三个条件（cnt=3）。

中意向‌：满足任意两个条件（cnt=2）。

低意向‌：满足任意一个条件（cnt=1）。

未达标准‌：未满足任何条件（可选，若需过滤可移除ELSE分支）。