Week04｜课时 3｜从 Week3 入湖到 Week4 湖仓：Bronze / Silver 最小表设计与 hidden partitioning

Table of contents

先把最小 Bronze / Silver 表设计讲稳，再谈更复杂的 Gold 与语义层 1
这节课解决什么问题 2
参考学习时间 2
学完这一讲，你应该能做到什么 2
本课产出 2
先看一张总图 3
1. 为什么本周只做最小 4 表 3
2. 最小 4 表设计地图 3
3. Bronze 与 Silver 在本周的正确角色 4
4. Source-to-Iceberg mapping 为什么必须先写 5
5. Schema source of truth 6
6. Hidden partitioning 解决什么 7
7. Partition evolution 的边界 7
8. 本周明确不交付什么 8
9. 学员常见设计错误 8
10. 本课收口判断 9
11. 本课自检清单 9
12. 课后最小行动 9
延伸阅读 9

先把最小 Bronze / Silver 表设计讲稳，再谈更复杂的 Gold 与语义层

这一讲要解决的不是“大而全建模”，而是：

Week03 的 ingest baseline 进入 Lakehouse 之后，最小 4 表到底该怎样站住。

进入课时 4 回看课时 2 返回 Week04 总览

下载讲义

提供适合离线阅读的 PDF 版和适合批注整理的 Word 版。

PDF 版：打印 / 离线阅读 Word 版 · 批注 / 二次整理

这节课解决什么问题

状态模型讲清之后，Week04 下一步就必须回答：

> **当前项目里，到底哪些表要先真实存在，哪些对象现在还不该展开。**

按照 Week4 handoff 的边界，本周至少要让 4 张表真实站住：

- bronze.raw_ticket_event
- bronze.raw_doc_asset
- silver.ticket_fact
- silver.knowledge_doc

这节课的重点，是把这 4 张表为什么这样设计讲清楚。

参考学习时间

建议按一节标准课安排：读完正文后，把 source-to-Iceberg 字段映射和 Bronze / Silver 设计写进 blueprint。

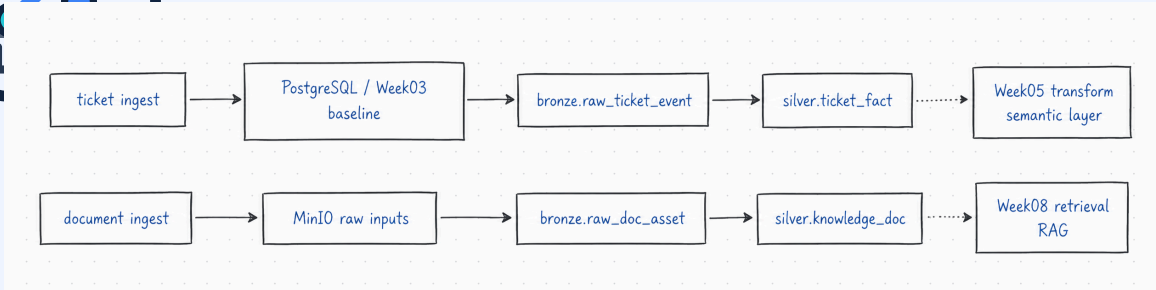
学完这一讲，你应该能做到什么

1. 理解为什么 Week04 先做最小 4 表，而不是一口气铺满 Gold。
2. 理解 Bronze 和 Silver 在当前项目里各自承载什么语义。
3. 知道 source-to-Iceberg 字段映射为什么必须先写清。
4. 解释 hidden partitioning 为什么比手工暴露分区目录更稳。
5. 明确哪些对象不是本周主交付。

本课产出

- docs/blueprints/week04/source_to_iceberg_mapping_v1.md
- docs/blueprints/week04/bronze_silver_table_design_v1.md

先看一张总图



实线是 Week04 要站住的最小流转：ticket / document 输入进入 Bronze，再形成两张可消费的 Silver。虚线是后续周次消费：Week05 做 transform / semantic layer，Week08 做 retrieval / RAG。虚线不是不重要，而是本周不抢跑。

1. 为什么本周只做最小 4 表

本周最重要的不是“表越多越厉害”，而是：

先让最关键的状态型表真实存在、真实可写、真实可回看。

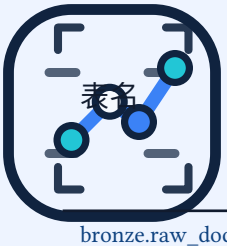
如果一上来就把 Gold、指标层、section/evidence 全铺开，会立刻出现 3 个问题：

- scope creep 到 Week05+；
- 本地教学环境难以稳定演示；
- 最小闭环没站稳，层数却先堆起来了。

真实企业里第一版 Lakehouse 失败，常见原因不是表太少，而是第一版模型过早追求完整：字段名还没对齐、source 语义还没确认、分区策略还没观察，就先铺了十几张表。Week04 的策略反过来：先让 2 张 Bronze 和 2 张 Silver 可写、可 inspect、可生成 baseline，再让后续周次继续扩展。

2. 最小 4 表设计地图

表名	层级	从哪里来	解决什么问题	关键字段类型	Week4 是否必须物化	后续会被谁消费
bronze.raw_ticket_event	Bronze	Week03 ticket ingest / source event	保留工单事件输入状态	业务键、事件键、ingest 时间、source manifest	是	silver.ticket_fact、Week06 backfill



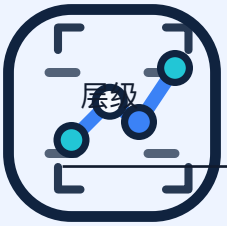
表名	层级	从哪里来	解决什么问题	关键字段类型	Week4 是否必须物化	后续会被谁消费
bronze.raw_doc_asset	Bronze	Week03 document ingest / raw object	保留文档资产输入状态	doc key、版本指纹、license / visibility、raw path	是	silver.knowledge_doc、Week07 parsing
silver.ticket_fact	Silver	Bronze ticket event 归一化	提供可消费的工单事实状态	ticket_id、状态、优先级、产品线、时间字段	是	Week05 transform、Week11 eval
silver.knowledge_doc	Silver	Bronze doc asset 归一化	提供可消费的文档资产状态	doc_id、doc_version、visibility、data_release 预埋	是	Week08 retrieval、Week14 release

这 4 张表足够支撑 Week05 继续 transform，也足够支撑 Week06/Week08 继续引用状态基线。

字段名最终以项目仓库的 pipelines/lakehouse/iceberg_schemas.py、实际 DDL 和 Week4 runbook 为准。课程页只解释字段类型与设计边界，不另起一套命名体系。

3. Bronze 与 Silver 在本周的正确角色

层级	小白理解	在本项目的角色	典型表	该做什么	不该做什么
Bronze	尽量保真的入湖记录	保留 ticket / doc 输入状态与 replay 入口	bronze.raw_ticket_event, bronze.raw_doc_asset	保留 source 线索、ingest 信息、去重锚点	不提前做最终业务解释
Silver	稳定可消费的业务对象	统一工单事实与文档资产状态	silver.ticket_fact, silver.knowledge_doc	统一业务键、状态、时	不等于最终 KPI 或 serving 表



层级	小白理解	在本项目的角色	典型表	该做什么	不该做什么
				间、权限/发布预埋	
Gold	面向指标和产品消费	后续语义层、指标层、服务层	Week05+ 再展开	做口径、指标、消费优化	不作为 Week4 主交付

Bronze / Silver 不是“新瓶装旧酒”。它们分别承担：

- 输入保真；
- 当前可消费状态；
- 后续 release / eval / retrieval 可以绑定的数据状态。

4. Source-to-Iceberg mapping 为什么必须先写

在任何真实 materialization 之前，先写清字段映射表，是本周的硬边界之一。

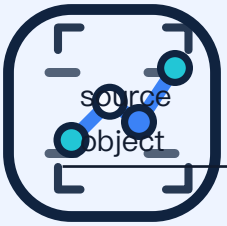
至少要明确：

- source 字段名如何映射到 Iceberg 列名；
- enum / code 是否转 string；
- json / jsonb 怎样进入表；
- list / array 怎样处理；
- timestamp 怎样保持时区语义；
- business key 与 technical key 分别是什么；
- 哪些字段来自 Week02 contract，哪些字段来自 Week03 ingest state。

如果这张映射表不先写，后面 table create 和 write 逻辑很容易各写各的。

最小 mapping 模板建议从这里开始：

source object	source field	target table	target field	trans-form	required	note
ticket event	待项目 schema 确认	bronze.raw_ticket_events	待项目 schema 确认	保真映射 / 类型标准化	yes/no	以 Week02 con-tract 与 Week03



source object	source field	target table	target field	trans-form	required	note
						ingest 输出为准
ticket event	待项目 schema 确认	silver.ticket_fact	待项目 schema 确认	业务键归一化 / 去重	yes/no	明确 business key 与 technical key
docu-ment as-set	待项目 schema 确认	bronze.raw_doc_asset	待项目 schema 确认	保留 raw path / source finger-print	yes/no	不提前生成 ev-idence anchor
docu-ment as-set	待项目 schema 确认	silver.knowledge_doc	待项目 schema 确认	doc 版本归一化 / 权限预埋	yes/no	为 Week07/ Week08 留接口

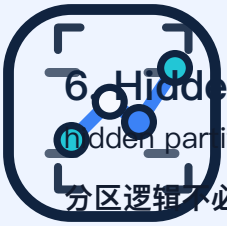
> 先写 mapping，再写物化脚本
>
> coding agent 最容易在字段名、nullable、时间语义和去重键上跑偏。mapping 是让人和 agent 对齐的契约：source column、target column、转换规则、是否必填、缺失处理和幂等策略都要先写清楚。

5. Schema source of truth

课程页面不能随意发明字段。真实字段以项目仓库为准：

- pipelines/lakehouse/iceberg_schemas.py
- 实际 DDL / source 表定义
- Week02 data contract
- Week03 manifest / ingest 输出

如果项目代码还没有落地真实 schema，本页只能写设计边界，不写死命令和字段全集。



6. Hidden partitioning 解决什么

hidden partitioning 的价值，不在于“多一个知识点”，而在于：

分区逻辑不必反复暴露给下游查询与上层语义。

传统手写目录分区容易把查询者绑死在物理布局上：

```
s3://warehouse/table/ingest_date=2026-04-27/source=ticket/...
```

Iceberg 的 hidden partitioning 则把分区 transform 交给表格式维护。查询者仍然写逻辑字段，表格式决定如何利用分区信息减少扫描。

这对课程很重要，因为 Week04 要先保证表状态可复现，再考虑下游怎么读。

7. Partition evolution 的边界

partition evolution 是后续随着查询模式变化调整布局的能力，但本周不做复杂演示。

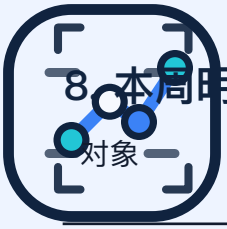
你只需要先建立判断：

- 当前分区策略服务于本地可跑 baseline；
- 不要为了“看起来高级”做过度分区；
- 分区字段必须服务查询模式和回放边界；
- 后续如果查询模式变化，可以演进 partition spec，但要配合 baseline 观察。

举个小白能迁移的例子：

- 第一版为了实验简单，可能先按 `month(ingest_ts)` 观察入湖批次；
- 后面排查发现客服团队更多按 `product_line` 分析问题；
- 正确做法不是马上重写所有历史目录，而是让分区策略随查询模式演进，并在 baseline 里记录变化影响。

> **i 不要把分区理解成手工建目录**
>
> hidden partitioning 的价值是让查询者继续按业务字段思考，表格式负责利用分区信息减少扫描。分区策略是工程判断，不是“字段越多越快”。



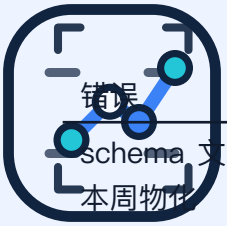
8. 本周明确不交付什么

对象	Week4 做到什么程度	为什么	后续周次
raw_ticket_event	进入 Bronze 最小表	保留工单事件状态	Week06 backfill / Week11 eval
raw_doc_asset	进入 Bronze 最小表	保留文档资产状态	Week07 parse / Week08 retrieval
ticket_fact	进入 Silver 最小表	提供稳定工单事实	Week05 transform
knowledge_doc	进入 Silver 最小表	提供稳定文档资产	Week08 indexing
knowledge_section	只预留边界，不主做	section 属于解析与证据锚点主线	Week07
evidence_anchor	只说明不抢跑	需要解析、chunk、引用策略	Week07/Week08
support_kpi_mart	不做	属于指标口径与 semantic layer	Week05
kb_serving_asset	不做	属于检索服务层	Week08

这些会在 Week05—Week08 逐步进入。

9. 学员常见设计错误

错误	后果	更稳的做法
Bronze 里提前业务解释	原始输入语义丢失，replay 难复盘	Bronze 保真，Silver 再统一
Silver blind append 当前状态表	当前事实重复、状态难解释	明确 business key、validity、dedupe 策略
分区字段和查询模式无关	文件布局看似复杂但没有收益	先从最小可解释分区开始
用技术字段替代业务键	下游无法解释同一业务对象的演进	technical key 与 business key 分开
课程页面随意发明字段	学员按讲义做，repo 无法对齐	以项目 schema 和 contract 为准



错误	后果	更稳的做法
schema 文件里出现的表都本周物化	scope creep，最小闭环不稳	先按最小 4 表验收

10. 本课收口判断

> ! 核心判断
>
> **Week04 先站住最小 4 表和字段映射，再谈更复杂的 Gold、指标和消费层。**

11. 本课自检清单

- 我能解释为什么本周只做 2 张 Bronze、2 张 Silver。
- 我能说清 Bronze 保真与 Silver 统一的区别。
- 我知道 mapping 必须先于 table create 和 materialization。
- 我没有把 Week07/Week08 的 evidence anchor / RAG serving 提前塞进 Week04。

再做一个小练习：

> 用 6 行以内为 `silver.ticket_fact` 写一段表设计说明。

必须包含：来源、业务键、去重策略、关键时间字段、权限/发布预埋字段、为什么它不是 Gold mart。

12. 课后最小行动

新建并补全：

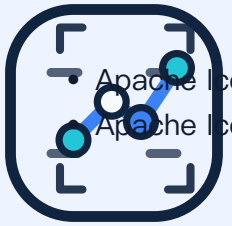
```
docs/blueprints/week04/bronze_silver_table_design_v1.md
```

建议至少包含：

1. 4 张表的角色说明；
2. source-to-Iceberg mapping 草案；
3. business key / technical key 区分；
4. partition 初始选择与边界；
5. 本周明确不做的对象清单。

延伸阅读

- Apache Iceberg / Evolution – schema 与 partition 演进[1]



- Apache Iceberg / Performance – metadata filtering 与文件剪枝[2]
- Apache Iceberg / Reliability – snapshot 与 atomic commit[3]

[1] Iceberg Evolution 文档说明，schema evolution 可以在不重写数据文件的情况下调整表结构；hidden partitioning 会让 SQL 不必暴露具体分区转换，partition evolution 也允许后续改变分区策略。本课的 Bronze / Silver 设计要服务这些长期演进能力。Apache Iceberg | Evolution

[2] Iceberg Performance 文档强调 table metadata、manifest 和文件级统计信息可以帮助规划器剪枝不需要扫描的数据文件；这也是 hidden partitioning 和合理表设计的工程价值之一。Apache Iceberg | Performance

[3] Iceberg Reliability 文档把 atomic metadata swap 和 snapshot history 作为表状态可靠性的核心机制。最小 4 表的设计不是为了多建表，而是为了让关键业务对象进入可提交、可回看的状态层。Apache Iceberg | Reliability