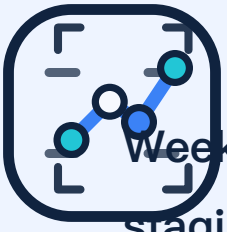


Week05 | 课时 2 | dbt 分层不是教条：从 sources / staging / intermediate / marts 到 AI 可消费数据产品

Table of contents

分层不是为了目录好看，而是为了让消费边界稳定

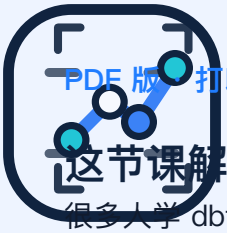
课时 1 讲清了为什么指标必须成为工程接口。这一讲回答：

这些接口从哪里长出来？SQL 文件怎样组织，才不会变成另一堆不可维护脚本？

进入课时 3 回看课时 1 返回 Week05 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

很多人学 dbt，会先记住一串目录名：sources、staging、intermediate、marts。这样学很容易变成教条。

Week05 要你建立的判断是：

> **分层不是为了“显得专业”，而是为了让上游变化、业务组合和下游消费不互相污染。**

参考学习时间

建议按一节标准课安排：先理解四层边界，再把当前项目的 dbt DAG 画出来。

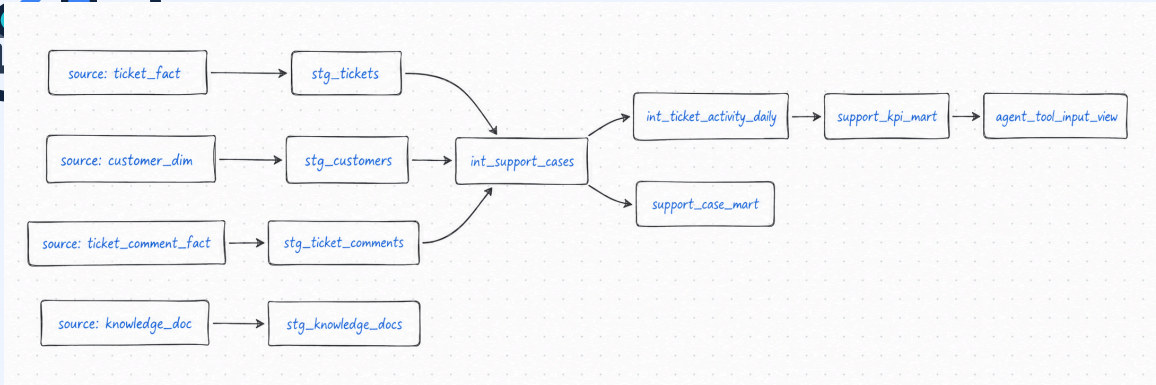
学完这一讲，你应该能做到什么

1. 解释 dbt 是什么、不是 什么。
2. 区分 sources、staging、intermediate、marts 的职责。
3. 说清 support_case_mart、support_kpi_mart、agent_tool_input_view 的粒度和消费边界。
4. 用 Docker-first 命令描述本地可复现运行路径。
5. 避免把还没开放的 source、维度或字段写进工具白名单。

本课产出

- analytics/dbt_project.yml
- analytics/models/sources.yml
- analytics/models/staging/stg_*.sql
- analytics/models/intermediate/int_*.sql
- analytics/models/marts/support_case_mart.sql
- analytics/models/marts/support_kpi_mart.sql
- analytics/models/marts/agent_tool_input_view.sql

先看一张总图



这张图里真正重要的是箭头：

- source 只承认真实上游；
- staging 只统一输入；
- intermediate 承接业务组合；
- marts 才成为消费产品；
- Agent 只读 safe view，不碰 raw source。

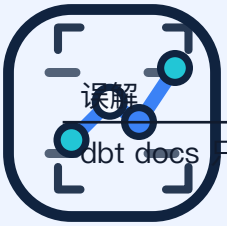
1. dbt 是什么、不是 什么

dbt 不是数据库，不负责存储数据；dbt 不是 BI 工具，不负责画图；dbt 也不是采集工具，不负责把外部系统的数据拉进来。

dbt 的核心作用是：

把 SQL transform 变成有目录、有依赖、有测试、有文档、有可复现命令的工程。

误解	更准确的说法	对 Week5 的意义
dbt 是数据库	dbt 不存数据，它组织 SQL transform	数据仍在 PostgreSQL / warehouse，Week5 只负责 transform 与口径层
dbt 是 BI	dbt 不负责画图，它负责把口径层做稳定	BI 应消费 mart / metric，而不是每张看板重写 SQL
dbt 是采集工具	dbt 不负责拉源数据	Week3 / Week4 是上游，Week5 从已进入系统的数据继续加工
dbt 是 SQL 文件夹	dbt 是依赖图、测试、文档和 artifacts 的工程	Week5 才能交付口径证据，而不是交付一堆散 SQL



误解	更准确的说法	对 Week5 的意义
dbt docs 只是好看的网页	docs / manifest / catalog 是变更影响分析和证据链的一部分	课时 3 会继续用这些 artifacts 证明口径能负责
有 mart 就能给 Agent 查	Agent 还需要 safe view、registry、tool contract 和 runtime guard	课时 5 才会把指标开放给受控工具

2. sources：先承认真实上游

sources 层的职责是承认真实上游，而不是美化上游。

当前项目已经在 analytics/models/sources.yml 中声明：

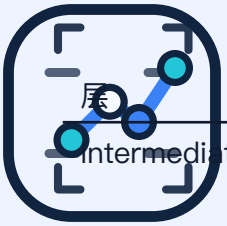
```
sources:
  - name: omni_postgres
    schema: public
    tables:
      - name: ticket_fact
      - name: customer_dim
      - name: ticket_comment_fact
      - name: knowledge_doc
```

这意味着课程页面不能再泛泛写 tickets / conversations / organizations 当作真实表名。那些表名如果出现，只能标注为示意；真实 Week05 主线以 omni_postgres source 和上述四张表为准。

sources 层要回答：

- 表来自哪里；
- schema 和 owner 是什么；
- 哪些关键字段必须存在；
- freshness 或数据释放版本如何解释；
- 这里不写复杂业务逻辑。

层	做什么	不做什么	小白判断方法	项目例子
sources	承认真实上游表	不改业务逻辑	“这张表真的从哪里来？”	omni_postgres.ticket_fact
staging	统一输入形状	不写最终 KPI	“字段名、类型、枚举是否稳定？”	stg_tickets



层	做什么	不做什么	小白判断方法	项目例子
intermediate	承接跨表组合	不直接给 Agent 消费	“复杂业务逻辑有没有集中入口？”	int_support_cases
marts	提供稳定消费边界	不塞进所有字段	“BI / registry 能不能长期复用？”	support_kpi_mart
safe view	给工具层安全消费	不暴露 PII、正文和任意 SQL	“Agent 能不能只看到被允许的结果？”	agent_tool_input_view

3. staging：统一字段，不塞 KPI

staging 层把真实 source 转成后续模型能稳定理解的输入。

当前项目里已经落了四类 staging：

model	来源	主要作用
stg_tickets	ticket_fact	统一 status / priority、派生 is_open、is_p1、sla_breached
stg_customers	customer_dim	提供 org / sla tier 维度，不暴露联系人 PII
stg_ticket_comments	ticket_comment_fact	提供 comment metadata 和首响计算输入
stg_knowledge_docs	knowledge_doc	保留文档元数据，给后续 Week08 留接口

staging 可以做：

- 字段重命名；
- 类型转换；
- 时间字段标准化；
- 枚举值规范化；
- 明确 PII 是否 redacted。

staging 不应该做：

- 最终 KPI 聚合；

- Agent 权限判断；
- dashboard 专用逻辑；
- 复杂跨表业务决策。

> ⚠ 不要把所有复杂逻辑塞进 staging
>
> staging 的目标是让输入可理解、可复用，不是提前做最终业务判断。把 KPI、权限和 dashboard 逻辑都塞进 staging，会让后面的 intermediate、marts、tests 和 docs 失去边界。

4. intermediate：承接业务组合

intermediate 层处理业务组合，但不急着暴露给消费方。

当前项目里有两个关键模型：

model	粒度	作用
int_support_cases	one row per support case	ticket + customer + first comment，派生首响、积压、SLA、P1、升级等字段
int_ticket_activity_daily	one row per date / product_line / priority / org_id / category	把 case 粒度聚合成 daily activity，供 KPI mart 展开成 metric rows

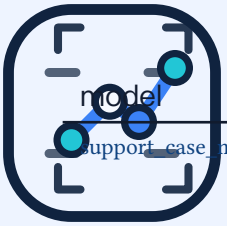
这里的好处是：

- support_case_mart 不需要重复 join 细节；
- support_kpi_mart 不需要理解原始 ticket/comment 表；
- 未来改首响逻辑时，有明确的影响入口。

5. marts：稳定消费边界

Week05 最小 mart 不是越多越好，而是边界必须稳定。

model	grain	谁消费	不该做什么
stg_ticket_events / stg_tickets	one row per source row	intermediate	不计算最终 KPI
int_support_cases	one row per case	marts	不直接给 Agent



model	grain	谁消费	不该做什么
support_case_mart	one row per support case	BI / KPI mart	不混入工具权限逻辑
support_kpi_mart	one row per metric / date / dimension	BI / Agent tool	不让 Agent 自由改 SQL
agent_tool_input_view	safe query shape	query_support_kpis_v1	不暴露敏感字段、正文、任意 SQL 入口

6. grain：比 SQL 写法更重要的问题

grain 是“一行代表什么”。如果 grain 不清，count、rate、avg 都可能看起来正确但业务含义错误。

对象	一行代表什么	为什么重要
support_case_mart	一个 support case / ticket	用来解释 ticket 状态、P1、SLA、首响、积压等 case 级事实
support_kpi_mart	一个 metric + date + 维度组合	用来让 BI、registry 和工具按统一粒度消费指标
agent_tool_input_view	一个工具允许返回的安全指标结果	用来隔离 PII、正文、raw event 和任意 SQL 入口
raw event / comment	一个事件或一条评论	不能直接当 ticket 或 KPI 计数，否则 reopen、comment 会放大数字

! 先说清一行代表什么，再写 SQL

如果一个 model 的 grain 说不清，所有 tests、docs、metric registry 和 tool contract 都会跟着变虚。

第一次写 mart，先写一句“一行代表什么”

在 SQL 前面先用自然语言写清 grain：例如 support_case_mart 是 one row per support case，support_kpi_mart 是 one row per metric / date / dimension combination。写不出来就先别急着写 SQL。

7. Docker-first 的运行方式

本课程默认走项目 runbook 的 Docker-first 路径，不要求宿主机直接安装 dbt。这样做是为了让学员环境尽量一致：dbt、profiles、数据库连接和依赖都在项目容器里对齐。[1]

先启动依赖：

```
cp infra/env/.env.example infra/env/.env.local

docker compose --env-file infra/env/.env.local -f infra/docker-compose.yml \
  up -d --build postgres minio minio_init
```

然后在 devbox 中验证 dbt 连接。DBT_PROFILES_DIR=. 的意思是让 dbt 使用 analytics/ 目录里的 profile 配置，而不是去读你电脑上的全局配置。dbt debug 主要验证 profile、依赖和数据库连接是否可用。[2]

```
docker compose --profile tools --env-file infra/env/.env.local \
  -f infra/docker-compose.yml run --rm devbox \
  bash -lc 'cd analytics && DBT_PROFILES_DIR=. dbt debug'
```

构建 Week05 模型。这里用 dbt build --select tag:week05，不是只跑 dbt run，因为 build 会把 models 和 tests 放在同一条验收路径里，更适合本周证明“模型能跑，口径也有检查”。[3]

```
docker compose --profile tools --env-file infra/env/.env.local \
  -f infra/docker-compose.yml run --rm devbox \
  bash -lc 'cd analytics && DBT_PROFILES_DIR=. dbt build --select tag:week05'
```

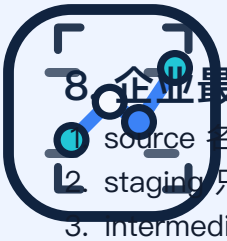
当前项目证据显示：dbt build --select tag:week05 已通过，support_case_mart=50、support_kpi_mart=300。

---

[1] Docker-first 是教学稳定性选择：降低本机 Python、dbt 版本、profiles 和数据库连接差异，便于学员复现 runbook。

[2] 当前 Student Core 从 PostgreSQL-backed source 起步，是为了让本地 transform 和指标工具链可复现；这不否定 Week04 的 Iceberg 学习，只是 Week05 的最小闭环先从项目已有 analytics 工程切入。

[3] dbt run 只构建模型；dbt build 会把模型、tests、seeds、snapshots 等资源按依赖关系执行，更适合课程验收。



8. 企业最佳实践：分层设计的 8 个判断

1. source 名称要稳定，不能让下游模型到处猜真实表。
2. staging 只做输入标准化，不写最终 KPI。
3. intermediate 用来收敛复杂业务组合，不直接开放给 Agent。
4. 每个 mart 都必须能用一句话说清 grain。
5. mart 不要把所有字段全塞进去，尤其不要把 PII、正文、raw body 带给工具层。
6. safe view 不是“又一个 mart”，而是工具层的安全出口。[4]
7. 业务逻辑进入 intermediate / marts 后，要配 tests、docs 和 lineage。
8. 分层不是银弹，真正要保护的是消费边界和变更影响。[5]

9. 本课最重要判断

> ! 核心判断
>
> dbt 分层不是为了目录命名统一，而是为了让 source 变化、业务组合、mart 消费和 Agent 工具边界能各自负责。

自检清单

- ☐ 我能说清 sources / staging / intermediate / marts 的边界。
- ☐ 我知道真实 source 名称是 omni_postgres.ticket_fact、customer_dim、ticket_comment_fact、knowledge_doc。
- ☐ 我能解释 support_case_mart 和 support_kpi_mart 的不同粒度。
- ☐ 我没有把复杂业务逻辑塞进 staging。
- ☐ 我的运行命令优先使用 Docker-first 路径。

课后最小行动

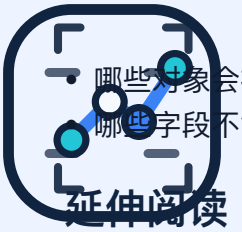
画出你自己的 Week05 dbt model map，至少包含：

- source 表；
- staging model；
- intermediate model；
- marts；
- 哪些对象会被 BI 消费；

---

[4] agent_tool_input_view 的职责是给工具层提供受控、安全、可审计的查询形状，不是让 Agent 绕过 registry 直接消费 mart。

[5] 分层本身不是目的。一个项目可以有不同目录结构，但必须能说清上游输入、业务组合、消费 mart 和工具安全出口分别由谁负责。



- 哪些对象会被 Agent 工具消费；
- 哪些字段不能进入 agent_tool_input_view。

延伸阅读

主题	推荐资料	为什么读
dbt sources	dbt Docs: Sources	理解 source 声明为什么是下游模型的输入契约
dbt project structure	dbt Docs: How we structure our dbt projects	对比 sources / staging / intermediate / marts 的常见组织方式
dbt build	dbt Docs: build command	理解为什么本周验收要把 run 和 tests 绑在一起
dbt docs generate	dbt Docs: docs generate	理解 docs、catalog 和后续 lineage 证据的来源
dbt artifacts	dbt Docs: Artifacts	理解 manifest.json、catalog.json、run_results.json 为什么会进入课时 3

Footnotes