Week05｜课时 4｜语义层的现实主义：MetricFlow、Semantic Models 与本地 dbt Core 的真实边界

Table of contents

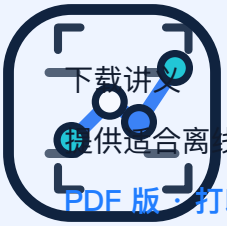
先理解语义层解决什么，再决定本地最小版本做什么

语义层不是平台名，也不是把 SQL 包一层新术语。

Week05 要先把语义层收紧成一句话：

让指标名、维度、实体、时间粒度、过滤边界和消费接口有唯一事实来源。

进入课时 5 回看课时 3 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

讲到 semantic layer，很多人会立刻跳到 dbt Cloud、MetricFlow 或某个平台产品。

但 Week05 需要先回答一个更基础的问题：

> **在本地 dbt Core 和当前课程项目里，哪些能力必须现在落地，哪些能力只是未来迁移选项？**

参考学习时间

建议按一节标准课安排：先理解语义层的内涵与外延，再把当前项目的 metric_registry_v1.yml 改写成你自己的指标清单。

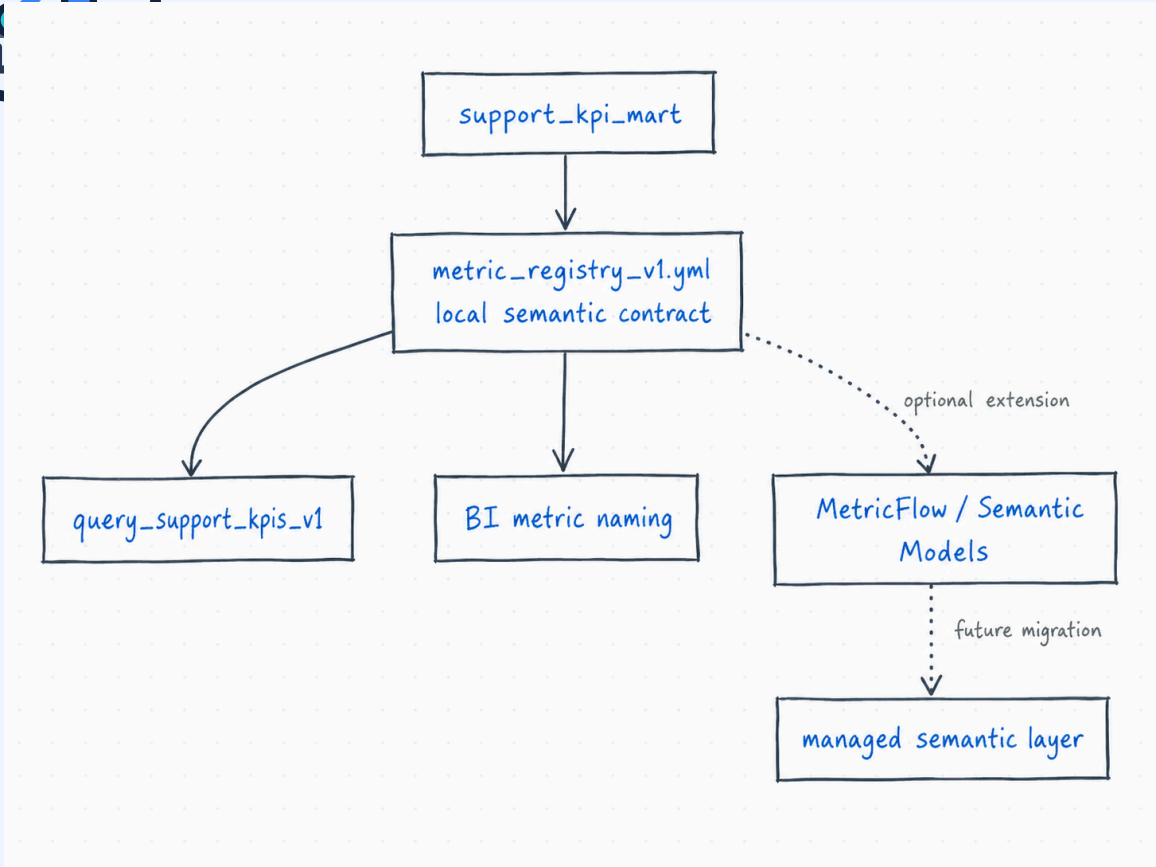
学完这一讲，你应该能做到什么

1. 解释 semantic model、entity、dimension、measure、metric、semantic graph。
2. 区分 MetricFlow / dbt Semantic Layer 的增强价值和本地 dbt Core 的边界。
3. 设计 metric_registry_v1.yml 的最小字段。
4. 避免把托管平台能力写成 Student Core 必备项。
5. 说明未来如何从本地 registry 迁移到更完整 semantic model。

本课产出

- analytics/metric_registry_v1.yml
- docs/blueprints/week05/semantic-layer-boundary.md
- 一份 registry → semantic model 迁移草图

先看一张总图



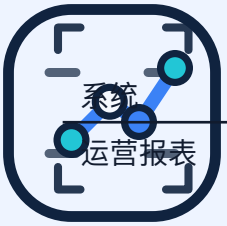
这张图表达的现实主义边界是：

- Student Core 必须能本地跑；
- 本地 registry 是当前唯一事实来源；
- MetricFlow / Semantic Models 可以理解和迁移，但不是本周必跑依赖。

如果没有语义层，会发生什么

同一个 `open_ticket_count`，三个系统可能各算各的：

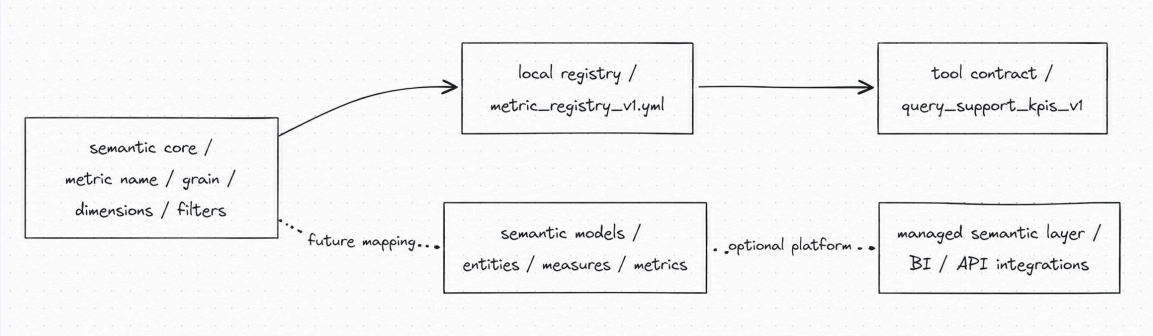
系统	它的定义	看起来合理在哪里	危险在哪里
BI 看板	`status != 'closed'` 的 ticket 数	能快速展示当前未关闭工单	没说明时间窗口和 release
Agent 工具	最近 7 天有活动且未关闭的 ticket 数	更像“当前活跃压力”	与 BI 同名但口径不同



系统	它的定义	看起来合理在哪里	危险在哪里
运营报表	SLA tier 内仍未解决的 ticket 数	更贴近运营复盘	过滤条件变成第三套

三者都叫 open_ticket_count，但 filters、time grain、role 和 source 都不同。语义层不是为了高级，而是为了 **同一个词在不同系统里不要变意思**。

语义层内核 vs 平台外延

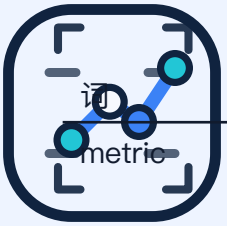


本周必须落的是内核：指标名、粒度、维度、过滤器、角色和窗口。平台外延可以迁移，但不能作为学生完成 Week05 的硬依赖。[1]

1. 先拆开语义层里的几个词

词	是什么	不是	OmniSupport 例子
semantic model	描述业务实体、维度、度量和关系的模型	不是某张单独的 dashboard SQL	support case metrics 的语义说明
entity	可以连接或识别业务对象的键	不是随便一个字段	ticket_id, org_id
dimension	用来分组、过滤、切片的业务属性	不是所有列都能当维度	product_line, priority, category
measure	可以被聚合的数值或表达式	不是完整业务指标	ticket_count, sla_breach_count

[1] dbt Semantic Layer 的完整下游集成通常依赖特定平台能力。本课程先保留理念和迁移路径，不把外部账户作为作业前提。



词	是什么	不是	OmniSupport 例子
metric	带业务定义、时间、过滤、权限边界的指标	不是裸 SQL 结果	p1_ticket_count
semantic graph	语义模型之间的实体关系图	不是好看的图，而是查询路径	case、 customer、metric、safe view 之间的关系
metric registry	本课程的本地最小语义契约	不是最终平台，也不是自由表单	analytics/metric_registry_v1.yml

MetricFlow 的价值是基于语义模型生成查询；dbt Semantic Layer 的价值是把这些能力托管化并连接下游工具。

但 Week05 的主线不是“先上平台”，而是先让指标定义在项目里可读、可测试、可迁移。

2. 语义层的内涵与外延

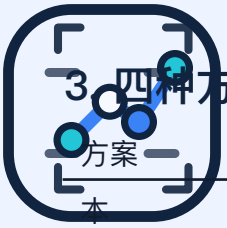
语义层的内涵

- 统一指标名；
- 统一维度名；
- 统一实体关系；
- 统一时间粒度；
- 统一过滤边界；
- 让消费系统不用各自发明 SQL。

语义层的外延

- BI 一致指标；
- Agent 受控查询；
- 指标 API；
- 权限与审计；
- 变更影响分析；
- 未来治理和 release 管理。

> ℹ 本周边界
>
> Week05 只把“语义层的内核”落到本地 registry 和受控查询工具里，不把平台托管能力变成作业必做项。



3. 四种方案的现实对比

方案	本周是否硬依赖	优点	边界
本地 metric_registry_v1.yml	是	本地可跑、可解释、可被工具读取	需要自己维护 runtime
MetricFlow local	否，可选了解	能用语义模型生成查询	版本和适配器管理更复杂
dbt Semantic Layer 托管	否	下游集成和权限能力更完整	需要平台账户，不适合作为 Student Core 硬依赖
NL2SQL	否	覆盖面灵活	口径、权限、审计不可控

> ⚠ 不要用 NL2SQL 替代语义层
>
> NL2SQL 解决的是“把自然语言变成 SQL”的生成问题，不会自动解决指标定义、角色权限、时间窗口、维度白名单和审计证据。Week05 先收紧 semantic contract，再谈更自由的查询入口。[2]

4. 当前项目的 registry 字段设计

当前 analytics/metric_registry_v1.yml 已经是一个最小 semantic contract：

```yaml
version: 1
registry_id: week05_support_metrics_v1
owner: analytics_engineering
source_model: support_kpi_mart
safe_view: agent_tool_input_view
time_dimension: metric_date
measure_column: metric_value
max_window_days: 31
allowed_dimensions:
  - product_line
  - priority
  - org_id
  - category
allowed_filters:
  - product_line
```

[2] NL2SQL 可以作为未来交互入口，但它不能替代指标白名单、角色过滤、时间窗口、审计和拒绝策略。

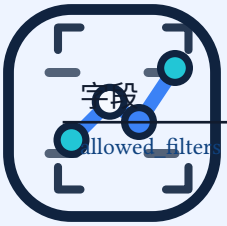
```
  - priority
  - org_id
  - category
  - data_release_id
allowed_roles:
  - support_ops
  - instructor
  - admin
metrics:
  - name: p1_ticket_count
    label: "P1 Ticket Count"
    description: "Number of P1 critical tickets created in the selected date window."
    aggregation: sum
    allowed_roles: ["support_ops", "instructor", "admin"]
```

这里最重要的不是 YAML 语法，而是四个边界：

- source_model 指向统一 mart；
- safe_view 指向工具唯一可读对象；
- allowed_dimensions / allowed_filters 限制 Agent 可问什么；
- max_window_days 把查询成本和语义窗口收住。

5. 本地 registry 字段全解

字段	保护的问题	示例	如果缺失会怎样
registry_id	当前指标契约是哪一版	week05_support_metrics_v1	难以审计查询基于哪版口径
metric_name	指标机器名稳定	p1_ticket_count	BI、Agent、评测各叫各的
source_model	指标从哪个 mart 来	support_kpi_mart	runtime 可能绕到 raw source
grain	一行代表什么	metric + date + dimensions	count、rate、avg 容易错
measure / measure_column	聚合哪个数值	metric_value	工具不知道该取哪个列
allowed_dimensions	允许怎么切片	product_line, priority	Agent 可能请求 PII 或不可控字段



字段	保护的问题	示例	如果缺失会怎样
allowed_filters	允许怎么过滤	org_id, category, data_release_id	查询会变成无限制自由 SQL
allowed_roles	谁能查	support_ops, instructor, admin	工具层无法拒绝越权
max_window_days	时间窗口和成本边界	31	查询可能过大、过慢、语义不稳
freshness	结果是否还可信	可记录 release / updated_at 策略	下游不知道数据是否过期
examples	正负例和使用说明	query examples / denial examples	工具测试只能靠猜

下面是 p1_ticket_count 的可读示意。实际字段以 analytics/metric_registry_v1.yml 为准：

```
metrics:
  - name: p1_ticket_count
    label: "P1 Ticket Count"
    description: "Number of P1 critical tickets created in the selected date window."
    source_model: support_kpi_mart
    grain:
      - metric_date
      - product_line
      - priority
      - org_id
      - category
    measure: metric_value
    aggregation: sum
    allowed_dimensions:
      - product_line
      - priority
      - org_id
      - category
    allowed_filters:
      - product_line
      - priority
      - org_id
      - category
      - data_release_id
    allowed_roles:
      - support_ops
      - instructor
```

6. 为什么不把所有指标都立刻开放

当前 support_kpi_mart 中已经能看到 avg_first_response_minutes，但 registry 没把它开放给工具。

这是一个好设计，而不是遗漏。

如果一个指标的业务定义、测试、文档、权限、负例和审计还没有准备好，它可以先存在于 mart 中，供数据团队观察，但不要立即进入 Agent 工具白名单。

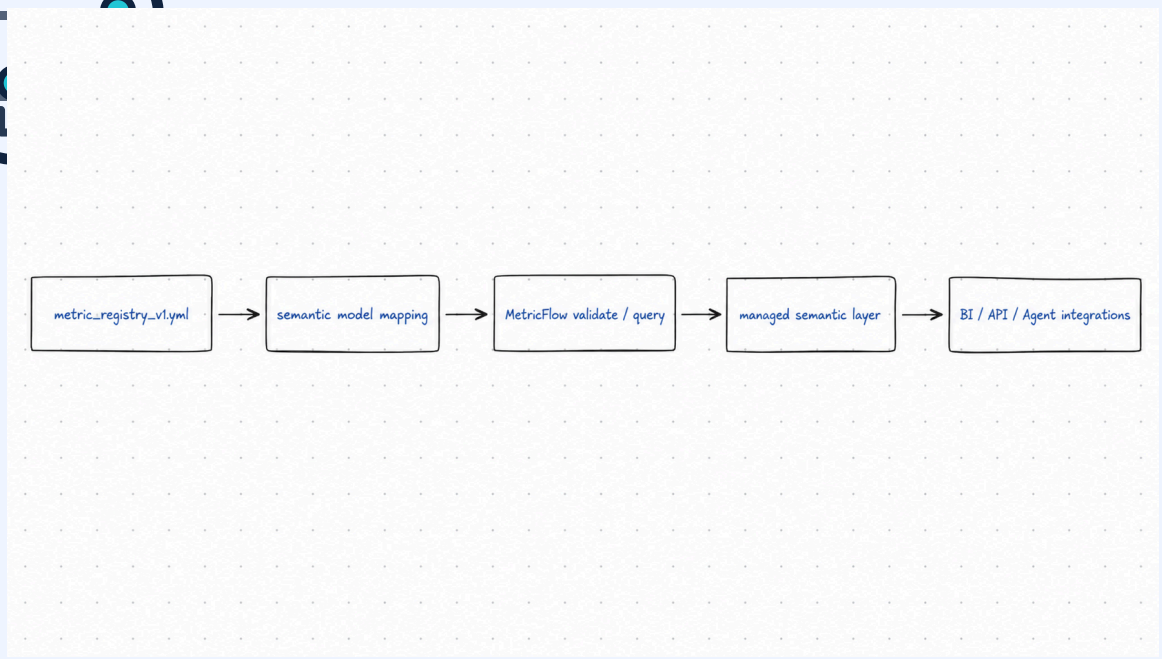
状态	可以做什么	不该做什么
mart 中已有字段	数据团队验证、写测试、补文档	直接开放给 Agent
registry 中已有指标	BI / Agent 可受控查询	绕过 tool contract
tool runtime 已支持	正例负例和审计可验证	接收 raw SQL

开放阶段	判断标准	avg_first_response_minutes 为什么可以暂不开放
mart 中已有	数据团队能计算和观察	可以先验证首响口径
docs/tests 完整	边界样本、字段解释、影响分析都齐	首响涉及人工、机器人、系统消息，需要更多测试
registry 开放	owner、roles、dimensions、filters 都明确	未明确前不进白名单
tool runtime 支持	正负例和 audit 都能追踪	未开放前不让 Agent 查询

7. 从本地 registry 迁移到正式 semantic model

迁移路线不要跳步：

1. 先保证 metrics 命名稳定；
2. 再把 registry 中的 measures / dimensions 映射成 semantic model；
3. 再本地尝试 MetricFlow validate / query；
4. 最后再考虑托管 Semantic Layer 和下游集成。



MetricFlow 可以帮助理解 semantic model 如何生成查询，但本地版本、适配器和命令链路会增加教学复杂度，所以本周不把它写成硬性作业。[3]

8. 本课最重要判断

> ! 核心判断
>
> 语义层的第一步不是买平台，而是先让指标名字、维度、过滤器和消费边界在工程里有唯一事实来源。

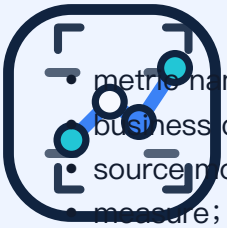
自检清单

- ☐ 我能说清 semantic model、entity、dimension、measure、metric 的区别。
- ☐ 我不会把 MetricFlow 写成本周必跑依赖。
- ☐ 我能解释本地 registry 为什么能作为最小语义层。
- ☐ 我知道 registry 和 tool contract 的关系。
- ☐ 我能说明未来如何迁移到更完整语义层。

课后最小行动

列出 3 个你最想放入 registry 的指标，并为每个指标写：

[3] MetricFlow 在 dbt Core 语境中可以帮助理解 semantic model 查询，但本地适配器、版本和 profile 管理会增加复杂度；Student Core 先用 registry 保证可复现。



- metric name；
- business definition；
- source model；
- measure；
- allowed dimensions；
- forbidden dimensions；
- owner；
- migration notes。

延伸阅读

主题	推荐资料	为什么读
dbt Semantic Layer	dbt Docs: Semantic Layer	理解托管语义层能提供哪些下游集成能力
dbt semantic models	dbt Docs: Semantic models	理解 entities、dimensions、measures 如何组织
dbt metrics	dbt Docs: Metrics	理解 metric 与 measure 的关系
MetricFlow commands	dbt Docs: MetricFlow commands	理解 validate / query 的概念边界
OpenAI Structured Outputs	OpenAI Docs: Structured Outputs	连接课时 5 的工具输出契约

Footnotes