AI 数据分析

从语义层到可核对的问数实践

技术原理 • AI与CLI实践 • 高级分析 • 持续运行

审阅稿 v1 | 2026-10-25 | 共40章

本版面向有一定编程或数据库基础的技术用户。各章注明真实验证版本和能力边界；当前教学环境为 Launcher 0.1.42 / CLI 0.1.25。

HTML保留可点击演示，Markdown可编辑，PDF适合离线阅读。样例与复跑源码随交付包提供。在线问数与FDP模型维护为第二阶段，Dashboard和报表为第三阶段。

本稿供作者审阅，尚未公开发表。

目录

第 1 章 从自然语言到 SQL 4

第 2 章 SQL 能执行结果为什么仍可能不对 8

第 3 章 语义层的作用与代价 12

第 4 章 自然语言到 DSL 再到 SQL 20

第 5 章 怎样选择和组织问数方案 23

第 6 章 认识 Foggy，准备第一个问数项目 25

第 7 章 让 AI 探查 Chinook 并确认业务口径 28

第 8 章 让 AI 生成语义模型并完成首次问数 32

第 9 章 让 AI 正确筛选数据：slice 与 having 40

第 10 章 聚合与计算指标：数清楚，再算平均和占比 44

第 11 章 按日期与月份汇总数据 49

第 12 章 从艺术家查到专辑：维度成员与多级关联 52

第 13 章 让 AI 只查询用户可见的数据 58

第 14 章 在 QM 中关联多个 TM：普通 JOIN 与聚合 JOIN 62

第 15 章 用字典连接业务名称与数据取值 66

第 16 章 模型维护与自动化回归 69

第 17 章 时间分析：比较、累计与滚动窗口 72

第 18 章 业务日期、日历与可比期间 78

第 19 章 存量与快照：月底还剩多少 83

第 20 章 业务耗时：已完成多久，未完成等了多久 86

第 21 章 同期群：同一批客户后来还买吗 90

第 22 章 父子维： 93

第 23 章 沿层级汇总与下钻：为什么总计不能再相加 98

第 24 章 Pivot：把多个维度放到一张交叉表 102

第 25 章 透视结果再计算：占父级多少，基准又是谁 107

第 26 章 把查询结果交给下一步：Compose 与 CTE 110

第 27 章 先汇总，再关联两个查询结果 114

第 28 章 先圈选对象，再继续分析 117

第 29 章 从一份 Excel 开始问数 120

第 30 章 不规范的 Excel 怎么整理 123

第 31 章 多份文件如何合并与更新 125

第 32 章 遗留数据库如何建立语义层 127

第 33 章 系统实际与 Excel 目标一起分析 129

第 34 章 让文件分析持续复跑 132

第 35 章 用 SQL 表达语义查询 135

第 36 章 SQL 转成 DSL 后，怎样核对查询含义 138

第 37 章 查询慢在哪里：先看执行解释 141

第 38 章 重复查询与数据更新：怎样判断缓存 143

第 39 章 预聚合：先汇总，再保持同一口径查询 145

第 40 章 分页与持续运行：把结果交付完整 147

第 1 章 从自然语言到 SQL

“帮我查一下，2025 年 6 月销售额最高的三个客户是谁。”

把问题、表结构和业务说明交给 AI，让它生成 SQL，再由查询工具执行，就是本章介绍的 Text-to-SQL 路线。

先明确销售额怎么算

本章使用 SQLite 和自编的虚构销售数据。销售额按有效发票总金额累计，月份按开票日期统计，只计算 ISSUED 状态。

金额以分保存，按人民币元展示时除以 100，因为 100 分等于 1 元；原本以元保存则无需换算。日期使用同一时区，格式为 YYYY-MM-DD HH:MM:SS。

实际项目中，应先从业务负责人、字段说明或已有报表确认这些口径。

把必要的表结构交给 AI

客户表 customers：

字段	含义
customer_id	客户 ID，主键
customer_name	客户名称

发票表 invoices：

字段	含义
invoice_id	发票 ID，主键
customer_id	对应客户表的客户 ID
invoice_date	开票日期，可能包含时分秒
total_amount_cents	发票总金额，单位为分
status	ISSUED 表示有效，CANCELLED 表示作废

一位客户可以有多张发票，两表通过 customer_id 关联。将上述结构和下面的要求一并交给 AI，也可以让它先通过工具读取表结构：

> 请用 SQLite 查询 2025 年 6 月有效发票金额最高的三个客户。按开票日期统计，只计算 ISSUED 状态，金额从分换算为元。返回客户 ID、名称和销售额，金额相同按客户 ID 升序。

看这句话怎样变成 SQL

对应的查询如下：

```
SELECT
    c.customer_id,
    c.customer_name,
    ROUND(SUM(i.total_amount_cents) / 100.0, 2) AS sales_amount
FROM invoices AS i
JOIN customers AS c
    ON c.customer_id = i.customer_id
WHERE i.status = 'ISSUED'
  AND i.invoice_date >= '2025-06-01'
  AND i.invoice_date < '2025-07-01'
GROUP BY c.customer_id, c.customer_name
ORDER BY sales_amount DESC, c.customer_id ASC
LIMIT 3;
```

i、c 是表别名。查询通过 JOIN 关联客户，用 WHERE 筛选有效发票，再按客户分组求和、排序取前三名。

时间范围写成“大于等于 6 月 1 日、小于 7 月 1 日”，可以包含六月最后一天带时分秒的记录。

/ 100.0 将分换算为元，例如 300000 分对应 3000 元；ROUND(..., 2) 将结果四舍五入到小数点后两位。

执行 SQL 后才能得到数字

生成 SQL 后，由数据库客户端或应用提供的查询工具执行，得到真实结果：



查询与建模流程

本例结果如下：

客户 ID	客户名称	销售额（元）
1	华东食品	5500.00
2	远山商贸	4200.00
3	青禾便利	3000.00

可以逐笔核对：华东食品的两张有效发票分别为 3000 元和 2500 元，合计 5500 元。北辰超市六月有一张 99999 元的作废发票，已被排除。

AI 整理回答时，应同时说明统计月份、使用开票日期和排除作废发票的口径。

先通过提示词解释数据结构

从这个例子可以看出，Text-to-SQL 首先要通过提示词向 AI 解释数据库的表、字段和关系，让它理解数据结构，再结合业务口径生成查询。

这些信息可以人工整理，也可以由程序读取元数据后补充到提示上下文中。表结构之外，还要说明状态编码、日期口径和金额单位，才能把“销售额”这样的业务说法对应到具体字段与计算方式。

让 AI 从源码整理提示词

已有 Java 源码时，可以让 AI 先整理提示词草稿：

- **JPA**：阅读实体类的表、列映射和关联注解，结合字段注释整理结构。例如，`@Column` 指定属性与数据库列的映射。
- **MyBatis**：阅读 Mapper XML、SQL 注解和 resultMap，提取字段映射、关联与筛选条件。

枚举和业务代码还能补充状态含义与计算规则。可以这样交代：

> 请阅读客户、发票相关的实体类、Mapper、SQL 和业务代码，整理表、字段、主键及关联，补充金额单位、状态、日期和计算规则。注明来源文件与位置，不确定的列为待核对项，输出供 Text-to-SQL 使用的提示词草稿。

再由人核对源码是否对应当前数据库，以及关联、单位和业务含义是否准确。这样可以减少逐表逐字段编写说明的工作。

自己改一次查询

把问题改成“2025 年 7 月销售额最高的两个客户”，调整时间上下界和条数限制，再核对结果。

将建表与数据、查询和复跑脚本放在同一目录，安装 Python 3 后进入该目录运行：

```
python -X utf8 verify.py
```

脚本使用 Python 自带的 SQLite，在内存中建库，核对六月查询和七月练习，输出结果并保存为 `chapter-01-results.json`。

本章小结

这次 Text-to-SQL 查询中，AI 承担了以下工作：

环节	AI 在本例中需要完成的工作
理解问题	确定月份、汇总对象和前三名要求
匹配表和字段	从结构说明中选择客户、发票及所需字段
确定关联与粒度	按客户 ID 关联，每张有效发票金额只累计一次

环节	AI 在本例中需要完成的工作
展开计算规则	确定有效状态、时间边界和金额求和方式
按需换算金额单位	以分保存、按元展示时，将金额合计除以 100.0；原本以元保存则无需换算
生成 SQL	按 SQLite 语法组织关联、筛选、分组、计算、排序和条数限制
根据结果回答	根据查询结果说明客户排名、金额和统计口径

查询工具提交 SQL，数据库执行数据读取与计算，AI 再根据结果组织回答。后面对比 DSL 路线时，可以逐项看哪些工作仍由 AI 完成，哪些定义交给语义模型和查询引擎。

下一章加入发票明细，看看 SQL 正常执行时，金额为什么仍可能算错。

第 2 章 SQL 能执行结果为什么仍可能不对

上一章，青禾便利六月的销售额是 3000 元。这次增加一个要求：“把商品明细行数也列出来。”

查询关联发票明细表后，SQL 正常执行，青禾便利却变成了 6600 元，排名超过了远山商贸。

加上明细表以后

统计口径沿用上一章：按开票日期统计六月有效发票的总金额。

新增的 invoice_lines 每行代表一条商品明细，通过 invoice_id 关联发票；line_id 是明细 ID，line_amount_cents 是明细金额，单位为分。本题统计明细行数，与商品销售件数不同。

如果直接关联明细，再按客户累计发票金额、统计行数，会得到下面的查询：

```
SELECT
    c.customer_id,
    c.customer_name,
    ROUND(SUM(i.total_amount_cents) / 100.0, 2) AS sales_amount,
    COUNT(d.line_id) AS detail_line_count
FROM invoices AS i
JOIN customers AS c
    ON c.customer_id = i.customer_id
LEFT JOIN invoice_lines AS d
    ON d.invoice_id = i.invoice_id
WHERE i.status = 'ISSUED'
  AND i.invoice_date >= '2025-06-01'
  AND i.invoice_date < '2025-07-01'
GROUP BY c.customer_id, c.customer_name
ORDER BY sales_amount DESC, c.customer_id ASC;
```

为方便核对，这次列出本期所有有有效发票的客户：

客户	销售额（元）	商品明细行数
华东食品	11000.00	4
青禾便利	6600.00	4
远山商贸	4200.00	1
南桥百货	1800.00	2

远山商贸的金额没变，其他客户却被多算了。

一张发票为什么被加了两次

华东食品的 1002 号发票共 3000 元，有两条明细。关联结果如下，金额已换算成元：

发票 ID	明细 ID	发票总金额（元）	明细金额（元）
1002	21	3000.00	1200.00
1002	22	3000.00	1800.00

一张发票关联成两行，两行都带着 3000 元的发票总金额，SUM(i.total_amount_cents) 就把它加成了 6000 元。华东食品的两张发票各有两条明细，因此 5500 元变成了 11000 元。

青禾便利的一张 1800 元发票有三条明细，另一张 1200 元发票有一条，错误结果是 1800 × 3 + 1200 = 6600 元。

这就是**粒度**问题：一行代表一张发票，还是一条商品明细。数据库按关联后的行求和，查询编写者需要保证发票金额每张只计一次。手写 SQL 同样可能犯这个错误。

先把明细汇总回发票这一层

先按发票统计明细条数，让中间结果每张发票只剩一行，再关联回来。将原来的明细 JOIN 替换为：

```
LEFT JOIN (
    SELECT invoice_id, COUNT(*) AS detail_line_count
    FROM invoice_lines
    GROUP BY invoice_id
) AS d
    ON d.invoice_id = i.invoice_id
```

SELECT 中原来的 COUNT(d.line_id) 也要改为累加已汇总的明细条数：

```
SUM(COALESCE(d.detail_line_count, 0)) AS detail_line_count
```

日期、状态和金额求和保持原样，完整 SQL 见修正查询。每张发票只关联一行统计结果；没有明细时，LEFT JOIN 保留发票，COALESCE 将条数按零处理。

结果恢复为：

客户	销售额（元）	商品明细行数
华东食品	5500.00	4
远山商贸	4200.00	1
青禾便利	3000.00	4
南桥百货	900.00	2

本样例直接累计 line_amount_cents 也能对上，因为明细合计与发票总金额一致。这里保留发票口径，同时增加明细条数；实际项目中需先确认两种金额是否一致。

按金额去重还会遇到另一种错误

还有一种看似省事的改法：

```
SUM(DISTINCT i.total_amount_cents)
```

当前样例恰好能算对，但 DISTINCT 去重的是金额值。

给华东食品增加一张 ID 为 1099、金额为 3000 元的有效发票后，三张发票应合计 3000 + 2500 + 3000 = 8500 元。金额去重后却只剩 5500 元。

要判断是否同一张发票，应看发票 ID；两张发票金额相同，也需要分别累计。

粒度正确以后还要核对哪些口径

避开重复累计，还不等于选对了业务口径：用户问订单金额，却查了发票金额；同样叫“销售额”，也可能有含税、未税之分。

日期也会漏数：若结束条件写成 invoice_date <= '2025-06-30'，月底 23:59:59 的发票就会被漏掉，青禾便利只剩 1800 元。

空结果也不能直接解释为“没有销售”，还需检查筛选、数据覆盖范围和刷新时点。

熟悉 SQL 不等于熟悉你的数据库

教学库的 total_amount_cents 把单位写在了字段名里，真实业务库未必这么清楚。假设 order_amount 存的是分，service_fee 存的是元，同样保存 1000，分别表示 10 元和 1000 元。查询两个字段时，不能统一除以 100。

即使提示词逐项说明，LLM 仍可能漏掉分转元，或把已经是元的字段再除一次。说明能帮助选对字段，却不能代替实际执行换算的计算规则。

旧字段也是陷阱：两个金额字段名字相似，其中一个却早已停止更新。选错后 SQL 照样能执行，数值还可能看着挺像那么回事。字段名、注释和样例值都要结合业务核实。

直接基于物理表生成 SQL，AI 通常需要相关结构和字段说明；工具若开放 Schema 浏览或展示 SQL，使用者也会接触这些物理细节。

说点不那么规范的：有时候写代码写烦了，我也会给字段起个 xxx_*** 之类的名字，或者加点不方便在客户面前念出来的前缀。这些名字留在库里还好，出现在问数界面上，可能就得先解释一下当时的精神状态了。大家还是多遵守命名规范，出来混都是要还的，指不定哪天就社死了。

准备查询数据时，应只提供当前主题所需的表和字段，排除过时字段；把常用换算、计算和过滤写成可复用的 SQL 表达式或视图，再用简短说明补充业务歧义。表结构或口径变化后，这些定义也要维护。Databricks 的实践建议同样优先采用结构化定义和查询示例，避免把所有规则堆进文字说明。

复杂 SQL 的复核和权限处理成本

前面只加了一次关联，就需要展开记录才能看清错误。再增加多层子查询、CTE、UNION 和计算字段，复核就要逐层追查字段来源、过滤位置和聚合粒度。

权限条件也有位置之分：把右表的权限条件直接放进 WHERE，可能让 LEFT JOIN 丢掉本应保留的无匹配记录；先计算全公司的前三名再过滤，也不等于本部门的前三名。

Text-to-SQL 可以通过数据库行级策略、受限视图或查询服务落实权限。数据库策略需要身份映射与配置；查询服务自行解析、改写 SQL，则要处理表、别名、子查询和分支。复杂 SQL 增加的是这类改写与验证成本，不意味着数据库原生权限也必须逐条改写查询。数据库行级策略示例、查询引擎权限示例。

权限必须来自可信身份和授权配置，不能根据用户在对话里自报的部门判断，也不能只要求 AI 记得加条件。

跨表探索可能需要较宽的只读范围，但仍须按工作需要授权。只读限制写入，不等于限制能读取哪些人的数据。

结果仍需要独立复核

本例可以逐笔核对发票金额，按发票 ID 检查是否重复累计；实际项目也要对照原始记录或已确认的报表，核对口径、粒度和结果。AI 可以辅助审查，但让它再解释一遍自己生成的 SQL，还不能算完成了复核。

运行本章样例

按样例说明中的第 2 章，将所需的六个文件放在同一目录，运行：

```
python -X utf8 verify-chapter-02.py
```

脚本在独立内存库中核对基础样例和边界情况，结果写入 chapter-02-results.json，不修改第一章原始数据。

本章小结

讲到这里，Text-to-SQL 的实现与限制应该更清楚了：LLM 熟悉 SQL，不代表它熟悉你的数据库。字段含义、业务口径和权限范围，需要有人确认、由系统落实，并用真实数据核对。

下一章介绍怎样把可复用的业务定义组织为语义模型，让其中已经确认的规则由引擎重复执行。

第 3 章 语义层的作用与代价

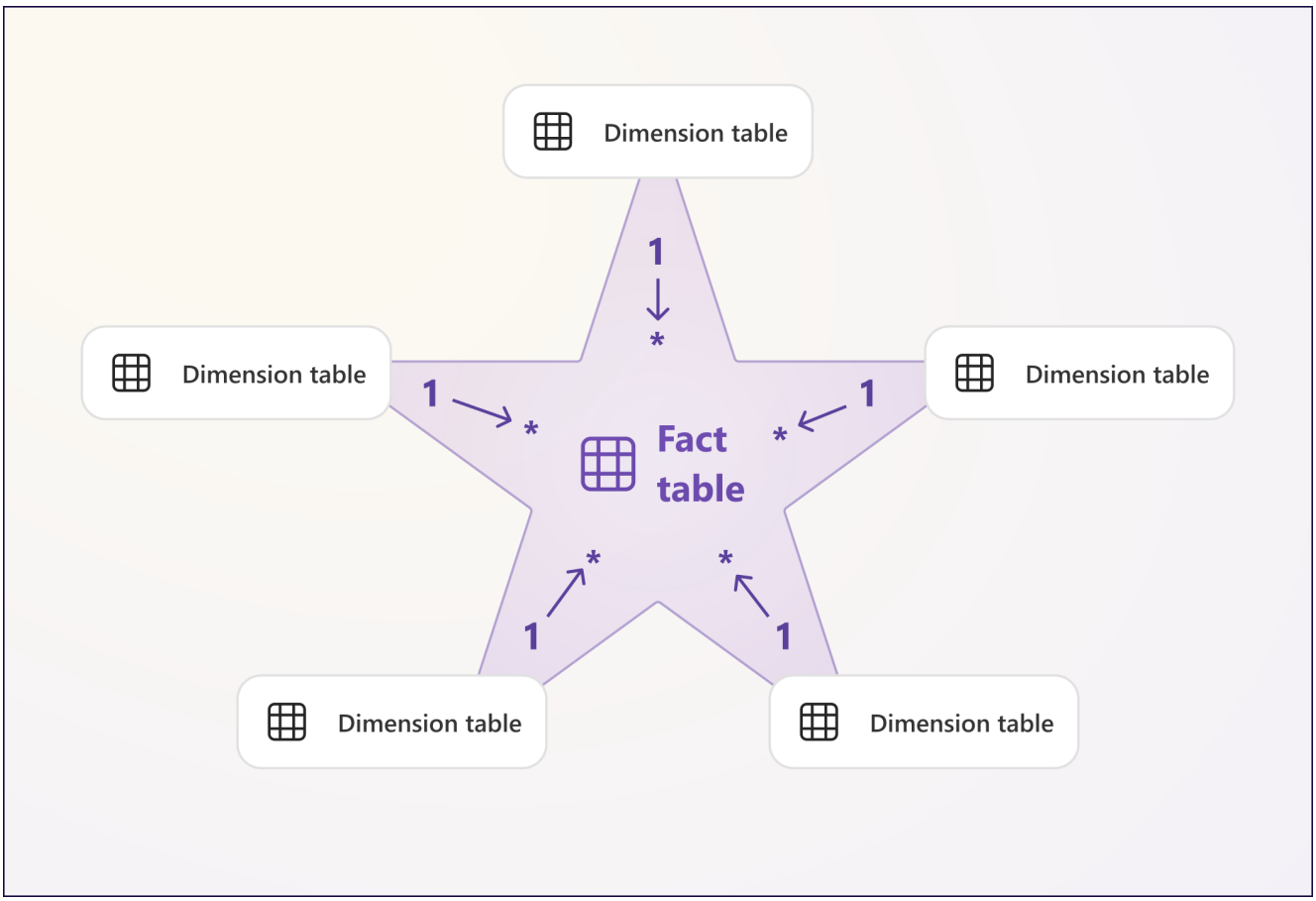
前两章为了查出六月客户销售额，已经确认了不少事情：金额来自发票，按开票日期筛选，只统计有效发票，分转元，每张发票只累计一次。

换成七月、按月份统计，这些规则大多仍然适用。我们希望建立的语义层，能让 AI 理解这些业务含义，正确调用查询，并依据实际结果如实回答用户。为此，系统需要提供可复用的定义、执行规则，以及 AI 能理解的查询接口和返回信息。

语义层是什么

面向分析的语义层，通常会将底层数据整理成**以事实为中心、由维度和指标组织起来的逻辑分析模型**，下文称“语义模型”。它沿用星型模型的核心思想，让 AI 面向业务概念查询，再由系统处理底层关联与计算。

先看一张典型的星型模型图：



典型星型模型：中心是事实表，周围是维度表

图片来源：Microsoft Learn：理解星型模型。

中间的 Fact table（事实表）记录业务发生了什么，例如一笔销售；周围的 Dimension table（维度表）提供观察角度，例如客户、商品、日期和地区。

这里的“向星型结构对齐”，发生在逻辑分析层。底层可以仍是现有业务表，日期也可以直接来自发票字段，无须另建日期表。语义模型负责整理来源与关联，并定义有效状态、分转元和汇总规则，为 AI 提供含义明确的销售分析入口。

整个语义层还可以包含销售、回款等多个分析主题，形成多颗相互关联的“星”；跨主题查询仍需协调各自粒度。

语义模型的基本构成

沿用客户和发票，几个常见概念可以这样理解：

要素	回答什么问题	本例
业务对象	分析围绕什么业务事物	发票、客户
粒度	一条事实代表什么	一张发票，而非一条商品明细
主键	怎样唯一识别对象	invoice_id 识别发票，customer_id 识别客户
维度	从什么角度筛选、分组	客户、开票日期
属性	怎样描述业务对象	客户名称；同名客户仍靠 ID 区分
指标	要计算什么、怎样汇总	有效开票销售额
关联	对象怎样连接、是一对几	多张发票对应一位客户

属性也可以公开为筛选或分组字段，具体用途由语义模型定义。指标则需要明确计算与聚合规则，不能只列一个字段名。

“六月哪些客户销售额最高”，就是按日期筛选、按客户汇总销售额。

先把销售额定义清楚

沿用小销售库，为本例的“有效开票销售额”写一张定义卡：

项目	本例定义
金额来源	invoices.total_amount_cents，存储单位为分
统计粒度	一张发票，以 invoice_id 识别；每张只累计一次
有效条件	status = 'ISSUED'
日期口径	使用 invoice_date，按统一时区解释
汇总与单位	累加有效发票金额，再除以 100.0，得到元
客户关系	通过 customer_id 关联客户，一位客户可以有多张发票

项目	本例定义
可见范围	按系统提供的当前用户身份和权限限制数据

“查六月”“按客户分组”“取前三名”属于本次查询条件。查七月时，指标定义不变，日期范围变了。

含税、未税、退款等业务差异也要明确，必要时分别命名指标，不能指望所有人说的“销售额”天然是同一个数。

从定义卡到可执行的语义模型

上面的卡片供人确认口径；要参与执行，还要转换为引擎认识的语义模型。下面是结构化定义示意，不是某个产品可直接加载的配置：

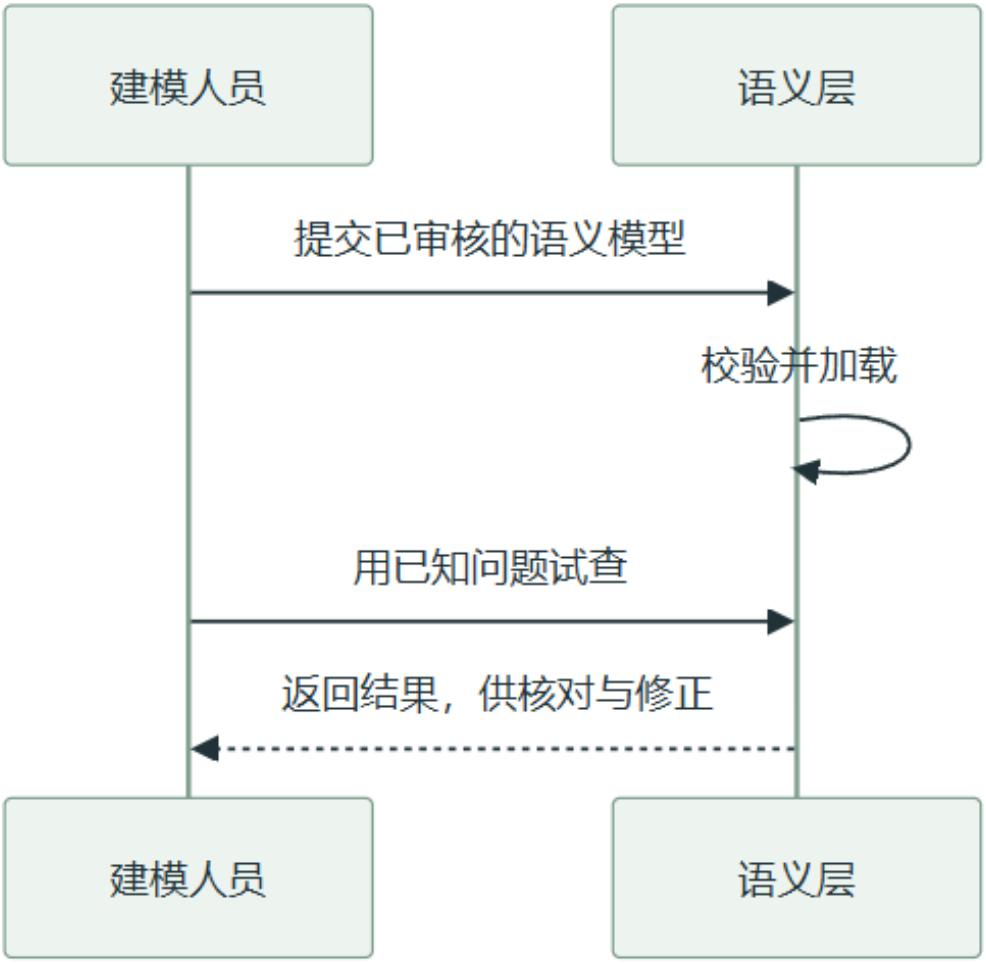
```
name: SalesInvoiceQueryModel
caption: 有效开票销售分析
object: invoice
source: invoices
primaryKey: invoice_id
fixedFilter: "status = 'ISSUED'"
dimensions:
  invoiceDate:
    column: invoice_date
    type: datetime
  customer:
    source: customers
    key: customer_id
    caption: customer_name
    join: "invoices.customer_id = customers.customer_id"
    relationship: many_to_one
metrics:
  salesAmount:
    expression: "invoices.total_amount_cents / 100.0"
    aggregation: sum
    type: decimal
    unit: 元
```

这里包含来源、主键、类型、过滤条件、关联关系和聚合表达式。加载时，系统将它们解析为语义模型对象，检查引用和表达式；查询时按这些定义组织计算。

写“单位为元”只是说明；`expression` 定义了单张发票的分转元，`aggregation: sum` 定义了汇总方式。系统依据这些规则生成计算，AI 不必每次重新决定这个字段是否除以 100。

建模人员：准备并验证语义模型

建模人员确认口径、编写语义模型，再用已知结果验证。这些工作可以由 AI 辅助；语法检查通过，仍不代表业务口径正确。



查询与建模流程

LLM 看到的是导出的查询元数据

语义模型加载后，系统按当前用户可见范围导出查询元数据，组织成 Markdown、JSON 或工具返回的描述，交给 LLM 使用。这里交付的是查询说明，而非完整的语义模型源文件；定义变更后，导出元数据也要同步更新。

导出的内容重点是：语义模型适合回答什么问题，字段怎样引用，单位和业务含义是什么，哪些字段能筛选、分组或聚合。物理表名、JOIN 表达式和内部权限配置，不必全部交给 LLM。

本例将客户关联的键和名称分别公开为 customerId、customerName；查询字段名可以与底层列名不同。

下面是该语义模型导出的 Markdown 简化示例。正式元数据还应包含字段的数据类型（如日期、数值、文本）等信息；示例字段名仅作说明，具体接口以实际公开名称为准：

```
# 有效开票销售分析
语义模型标识：SalesInvoiceQueryModel
语义模型名称：有效开票销售分析
适用：按开票日期分析有效发票销售额。

可查询字段：
```

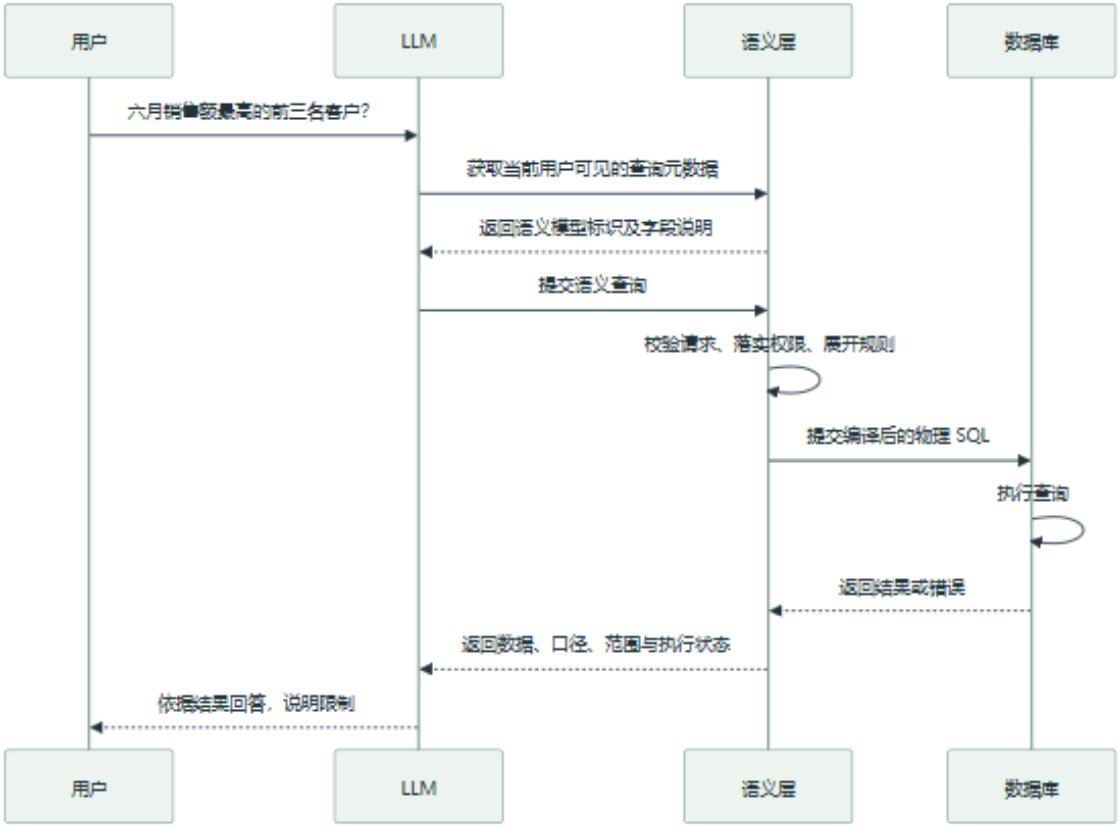
- customerId：客户 ID，可分组；用它区分同名客户。
- customerName：客户名称，展示时与客户 ID 一起使用。
- invoiceDate：开票日期，支持日期范围筛选。
- salesAmount：单张发票金额，单位元，可求和。

LLM 据此把“六月销售额最高的客户”对应到日期、客户分组和金额求和，并用 SalesInvoiceQueryModel 指定查询目标。语义模型元数据怎样用于查询生成。

一次查询怎样使用语义模型

本次查询只需表达：按客户统计六月的 salesAmount，金额降序，客户 ID 升序，取前三名。这里先用文字表示意图，下一章再写具体请求。

建模不必随每次提问重复。用户提问后，LLM 结合查询元数据与协议组织请求；问题有歧义时先澄清。下图展示问题已明确的查询过程：



查询与建模流程

其中，语义层的查询处理包括：

环节	本例怎样处理
字段绑定	将请求中的 customerId、salesAmount 绑定到语义字段及其物理来源
计算展开	按求和规则展开为 SUM(invoices.total_amount_cents / 100.0)

环节	本例怎样处理
过滤与关联	加入语义模型的 ISSUED 条件，按声明路径关联客户，并应用六月范围
聚合规划	按客户 ID、名称汇总，确保本次取数不展开为多条明细
校验与编译	拒绝不可用字段或不支持的组合，落实权限，生成目标数据库的查询

这些规则无须 LLM 在每次请求中重写：它没有填写有效状态条件，固定过滤仍应生效。报表或应用提交同样的语义请求，也会使用这些规则。

权限同样需要执行链路：可信身份提供用户范围，引擎或数据库施加约束。仅在描述里写“只能查本部门”，不会自动形成授权策略。

有些集成只用语义模型描述辅助 AI 生成物理 SQL。它能改善理解，却没有获得上述查询入口对业务规则的执行约束，两种用法需要分清。

指标也不只是字段别名

销售额可以求和，其他指标却未必。假设再定义“平均每张有效发票金额”：

平均每张有效发票金额 = 有效开票销售额 ÷ 有效发票张数

按客户查询时，分子和分母都在各客户范围内计算；看六月总计时，要用全期总金额除以总张数。不能把各客户的平均值直接相加，也不能不看张数就取平均。

因此，语义模型不仅要说明字段叫什么，还要表达指标依赖、聚合方式和适用维度；执行器要在当前分组、筛选范围和计算阶段展开这些定义。指标不是预先算好的一个数，也不是给某列换个中文名。指标在查询上下文中求值的解释。

字段公开了，不等于能任意组合

发票金额可以按客户汇总；扩展到商品时，一张发票却可能包含多种商品，不能把整张发票金额算给每一种商品。此时要明确按商品分摊的规则，或改用商品明细金额。

语义模型需要声明关联的基数（一对一、多对一或一对多）和指标适用的维度。引擎在支持范围内规划关联与汇总，无法可靠处理的组合应明确拒绝。只声明“发票关联明细”，不会自动解决重复累计。维度与指标粒度的约束示例。

查对了，还要答得如实

本例查出的六月前三名客户销售额分别是 5500、4200、3000 元，合计 12700 元。AI 可以说“前三名客户合计 12700 元”，却不能说“六月总销售额为 12700 元”：查询只取了前三名。全体客户的六月合计是 13600 元，需要另一项汇总才能得到。

因此，查询工具返回数值时，还应保留指标、单位、筛选范围、是否截断和执行状态，供 AI 核对和说明。没有记录、金额为零和查询失败要分别处理；问题有歧义时先澄清，只有销售数据时，也不能凭空编出业绩变化的原因。语义层能提供依据与约束，最终回答仍需检查。

回看第二章：哪些工作交给了语义层

第二章已经用 SQL 修正了重复累计；视图、公共表达式和数据库策略也能复用规则。下面对比直接面向物理表生成 SQL 与通过语义层查询，看看工作怎样分配：

第二章的问题	直接生成物理 SQL	通过语义层查询
分、元混用，口径不一致	在 SQL 或视图中落实换算与过滤	在语义模型中集中定义，由引擎展开
关联后重复累计	按粒度组织 JOIN 与汇总，核对结果	声明粒度与关联，由引擎在支持范围内规划；无法可靠处理时拒绝
旧字段与内部命名	整理 Schema，或用视图提供业务字段	公开确认过的语义字段，物理映射留在后台
权限处理复杂	使用数据库策略、受限视图，或解析改写 SQL	对语义请求统一校验，结合执行引擎或数据库策略落实权限
查询复核困难	追查 SQL 中的字段来源、过滤与聚合	先核对语义定义与查询意图，再追溯实际 SQL

上一章那些不方便念出来的字段名，也终于有机会留在幕后。

可以看到，语义层承担了大部分确定性工作，LLM 不必直接处理复杂的物理表结构、关联与计算细节，减少了犯错的概率。

它又带来了哪些问题

规则有了统一的执行位置，LLM 还需要掌握查询接口。沿“自然语言 → DSL → SQL”这条路线，主要新增的门槛在 DSL（领域专用语言）的使用上：**LLM 即使熟悉 SQL，也不能默认它熟悉某个系统自定义的 DSL。**

查询元数据告诉它有哪些语义模型和指标，却不一定告诉它该怎样拼出合法请求。接入时需要提供两类上下文：

上下文	回答什么问题
查询元数据	查哪个语义模型、字段是什么意思、单位与可用分析范围是什么
查询协议	语义模型标识放在哪里，怎样选择字段、写过滤、分组、聚合、排序与分页

即使请求是合法 JSON，字段、操作符或计算阶段写错，仍然不能正确查询。第四章会用具体请求说明。

因此，要通过提示词、工具 Schema 和完整示例，让 LLM 知道这个接口的用法，包括支持什么、出错后怎样修正。复杂分析增加新操作后，协议说明与验证题库也要跟上；更换 LLM 时，也要重新验证适配效果。

放在一起看，直接查询物理表时，LLM 可以使用熟悉的 SQL，却要直面业务口径与表关系的复杂度；改用语义 DSL，业务含义清晰了，又需要教会 LLM 怎样使用查询协议。

语义层也可以提供 SQL 入口，让 LLM 面向虚拟业务表查询；本系列先用 DSL 学完整的问数流程，语义 SQL 留到高级篇“语义 SQL 与虚拟业务表”专题展开。

SQL 的表达能力很强，但部分复杂的分阶段或多维分析，用合适的 DSL 可以表达得更简洁。

我个人认为，随着 LLM 理解和推理能力增强，自定义 DSL 的适配门槛有望逐步降低。当 LLM 能够准确理解协议、拆解分析步骤时，就更能发挥 DSL 在复杂分析中表达简洁、边界清晰的优势。这些差异留到高级篇展开。

没有数仓，也可以先建语义层

ETL 负责抽取、转换和加载，例如统一日期格式、处理重复记录、汇集不同来源的数据；语义层负责业务含义和查询规则。源数据漏了发票，语义模型再完整也算不回来。

中小企业可以先对现有业务表或视图建模，遇到数据质量、跨来源对齐问题，再补充必要的数据准备。多来源处理放到高级篇展开。

一个小练习：客户数量怎么算

为“六月客户数量”补全三项定义：统计什么客户、按什么键去重、使用哪个日期。

在原始样例的全部数据范围内，“客户档案总数”是 5；“六月有有效发票的客户数”是 4。后者需要按开票日期和有效状态筛选，再按 `customer_id` 去重。即使一位客户有多张发票，也只能算一位。

两个数都可能算对，回答的是不同问题。语义层要把这种区别命名、定义清楚。

本章小结

语义模型定义业务含义和执行规则，元数据与查询协议帮助 LLM 表达意图，引擎负责校验与编译，数据库执行计算。AI 再依据结果及其范围回答；查询成功，仍需检查答案是否如实。

采用自定义 DSL 时，还要解决 LLM 对查询协议的适配。下一章用同一个销售问题，写出具体请求，逐项对照生成的 SQL。

延伸阅读：指标定义与分组维度、业务对象与语义映射。

第 4 章 自然语言到 DSL 再到 SQL

仍然查询“2025 年六月销售额最高的三位客户”。上一章已经定义了语义模型；现在看 LLM 怎样表达本次查询，以及引擎怎样把它转换为物理 SQL。

这条路线是：自然语言 → 语义 DSL → 物理 SQL → 数据库执行。

先选择语义模型，再读取字段

沿用上一章的 SalesInvoiceQueryModel：以发票为粒度，只统计有效发票，金额统一为元，并公开以下字段。

语义字段	含义及本例用法
customerId	客户 ID，分组时区分同名客户
customerName	客户名称，用于展示
invoiceDate	开票日期，用于筛选
salesAmount	单张有效发票金额，单位元，可以求和

LLM 先从当前用户可见的语义模型中选择，再读取查询元数据，确认字段、类型、单位与可用操作。

本次问题可以拆成：筛选六月、按客户分组、累计销售额、按金额降序取三位。金额并列时按客户 ID 升序，保证结果顺序稳定。

把业务用语对应到字段和值

字段选对了，筛选值也要有依据：

- **业务别名。** 用户说“销售额”，应对应已确认的指标；同时存在开票额、回款额时，先按上下文判断，无法确定就澄清。
- **枚举与维度值。** 扩展到地区等维度时，“华东”可能对应一个编码。应从已定义的值映射或工具返回的当前用户可见维度成员中确认，不能让 LLM 凭名称猜编码。
- **相对日期。** “上个月”需要结合当前日期、业务时区和统计日期，转换为明确区间。本例已指定 2025 年六月，无须猜测年份。

查询元数据提供业务含义与已定义的映射；具体维度值可以按需查询，不必一次全部塞给 LLM。

用 DSL 表达本次查询

元数据告诉 LLM“能查什么”，查询协议告诉它“怎样写请求”。为说明二者怎样配合，这里约定一套简化 DSL，用 JSON 承载查询意图，不作为某个产品的可执行接口：

```
{
  "model": "SalesInvoiceQueryModel",
```

```
  "columns": [
    "customerId",
    "customerName",
    "SUM(salesAmount) AS totalSales"
  ],
  "slice": [
    {
      "field": "invoiceDate",
      "op": "[)",
      "value": ["2025-06-01", "2025-07-01"]
    }
  ],
  "groupBy": ["customerId", "customerName"],
  "orderBy": ["-totalSales", "customerId"],
  "limit": 3
}
```

即使之前没用过这套协议，看到字段、分组、排序和条数，大概也能读出它想查什么。DSL 不必从零发明一套陌生语法，可以借助 JSON、YAML 等熟悉的格式表达领域含义：上一章用 YAML 定义语义模型，这里用 JSON 描述查询。需要学习的是具体的键名、操作符和执行含义。

在这套约定里，`model` 指定语义模型；`columns` 选择返回字段，支持求和及别名；`slice` 筛选聚合前的记录；`groupBy` 指定分组；`orderBy` 中的负号表示降序；`limit` 限制返回条数。

这里的 `SUM` 适用于可加总的单张发票金额。平均金额、比例等指标，应遵循语义模型公开的计算方式，不能一律求和。

`[)` 表示左闭右开区间，包含六月末尾，不包含七月一日。LLM 需要通过协议说明、工具 Schema 和示例获知这些写法。

筛选发生在聚合前还是聚合后

同样是“超过三千元”，筛选对象不同，结果含义也不同：

用户问题	处理含义
只统计单张金额超过三千元的有效发票	先筛选发票，再汇总金额
找累计有效开票销售额超过三千元的客户	先按客户汇总，再筛选客户的汇总金额

在 SQL 中，两者通常分别用 `WHERE` 和 `HAVING` 表达。本章的简化 DSL 只示范了 `slice` 聚合前筛选；聚合后筛选需要查询协议另行支持，不能把汇总条件直接塞进 `slice`。

语义模型的有效状态和权限约束仍要生效；请求按需表达行筛选、分组、汇总后筛选、排序和条数。LLM 要分清这些阶段，引擎也要校验相应操作是否支持。

引擎怎样落实查询

引擎先检查语义模型和字段是否可用、类型、字段组合与聚合是否合法，再根据可信身份落实权限。随后展开金额公式、加入固定过滤、组织客户关联，编译为目标数据库的 SQL。

下面用 SQLite 写出本例的等价物理 SQL，帮助读者核对规则落在了哪里。它是教学示意，不是某个引擎的真实编译输出。

```
SELECT
    c.customer_id,
    c.customer_name,
    SUM(i.total_amount_cents / 100.0) AS total_sales
FROM invoices AS i
JOIN customers AS c ON c.customer_id = i.customer_id
WHERE i.status = 'ISSUED'
  AND i.invoice_date >= '2025-06-01'
  AND i.invoice_date < '2025-07-01'
GROUP BY c.customer_id, c.customer_name
ORDER BY total_sales DESC, c.customer_id ASC
LIMIT 3;
```

其中，客户 JOIN、ISSUED 条件和 / 100.0 来自语义模型；六月范围、客户分组、排序与前三名来自本次请求。

在小销售库中执行这段 SQL，结果仍为：

客户	销售额（元）
华东食品	5500.00
远山商贸	4200.00
青禾便利	3000.00

本例使用全部样例数据，实际结果还取决于当前用户的数据范围。LLM 回答时，应保留“2025 年六月”“有效开票销售额”“前三名”等信息。

字段写错时怎样修正

如果 LLM 把 customerName 写成 clientName，当前语义模型没有这个字段，请求应在校验阶段被拒绝。

LLM 根据诊断重新读取查询元数据，确认是拼写错误、字段未公开，还是选错了语义模型，再修正请求。不能仅凭名字相近就替换，也不能为绕过权限退回物理表查询。

诊断可以帮助修正无效字段，却未必识别业务选择错误。比如用户问回款，LLM 却选择了有效开票销售额，字段和语法都可能合法，仍需核对查询意图。

本章小结

LLM 理解问题、选择语义模型并表达本次查询；引擎依据语义定义校验与编译，数据库完成计算，LLM 再按实际结果回答。业务规则由系统落实，查询意图和回答仍需要核对。

下一章把直接 Text-to-SQL 与语义层查询放在一起，讨论不同场景怎样选择，以及怎样组织完整的问数系统。

本章样例与核对方法提供请求示意、等价 SQL 与核对范围。

第 5 章 怎样选择和组织问数方案

前四章介绍了两条路线：直接生成物理 SQL，或通过语义 DSL 查询。本章先根据使用者和问题选择起点，再看怎样组合工具、语义能力和多步分析。

什么场景适合直接 SQL，什么场景适合语义层

如果服务的是日常与 SQL 打交道的技术人员或数据分析师，他们熟悉业务库，能够审查生成的 SQL 和结果，直接 Text-to-SQL 是合适的起点，也便于探索新问题。

如果目标用户不懂 SQL，甚至不了解数据库概念，更适合先通过语义层整理业务口径与可查询范围，让他们围绕业务概念问数。即使使用者会 SQL，团队反复使用统一指标、需要多人稳定问数时，也值得采用语义层，减少重复定义与维护。

这有点像用户要查数据时，你是给他一个 SQL 客户端，让他自己组织查询，还是提供口径已经确认的报表，让他直接看业务结果。语义层还可以在已定义的范围内支持灵活提问和追问，而不局限于固定报表。

两种方式可以并存：技术人员先探索新问题，再把确认过的常用定义沉淀到语义层。

业务规则可以放在哪里

决定采用语义层后，先看已有工具和数据环境能提供哪些能力：

组合方式	业务定义与执行在哪里	LLM 怎样参与
复用已有分析系统	使用已经确认的维度、指标、关联和查询接口	读取元数据，选择字段与条件，调用已有查询能力
接入独立语义查询服务	由服务加载语义模型，统一校验与执行	调用语义查询工具，与报表等入口复用定义
利用数据平台内的语义能力	在数据平台中维护指标及其访问规则	围绕平台公开的业务查询面组织请求

这些方式可以组合，也可以分步建设。已有合适的语义模型与接口，就先核对能否复用；若要提供语义查询，而手头只有原始业务表，就需要补上业务定义和查询能力。不能因为产品名字里有“AI”或“语义”，就省掉实际核对。

工具接通，还要看它执行什么

CLI、API 或工具协议解决的是怎样调用能力。一个工具可能只返回表结构，也可能执行物理 SQL，或接受上一章的语义查询。这些接口都能被 LLM 调用，承担的工作却不同。

接入时至少看清三件事：

- LLM 收到的是原始表结构，还是语义模型的查询元数据。
- 查询是否经过落实口径、单位、关联和权限的执行入口。

- 返回的信息是否足以说明数据范围、结果来源和执行状态。

多步分析由谁组织

“六月客户前三名”可能一次查询就够；“销售变化主要来自哪些客户”，往往需要先比较两个月的汇总，再按客户拆解变化，核对客户差额是否与总差额一致。

负责多步任务的 AI 助手，通常称为 Agent，在这里安排步骤、调用工具并检查中间结果。它可以使用同一个语义服务，也可以组合已有分析工具。语义层则负责每次查询所用的业务定义与执行规则，两者可以配合。

用一个问题检查整条链路

仍用“2025 年六月销售额最高的三位客户”检查：LLM 提交了什么查询，系统在哪里落实有效状态和分转元，结果适用于哪个用户范围，最后的回答是否保留了这些含义。

再追问“为什么第一名更高”，看系统会不会检查可用数据、补充查询或说明证据不足。界面能回答第一句话，不等于已经具备持续分析的能力。

本章小结

选择路线时，要同时看使用者审查 SQL 与结果的能力、指标复用和交付要求。采用语义层后，可以复用已有分析系统，也可以接入独立服务；Agent 可以组织多步分析，查询仍经过相应执行入口。

技术篇到这里结束。应用篇从确定首批问题和验收范围开始，动手完成一个小型问数项目。

第 6 章 认识 Foggy，准备第一个问数项目

前五章讲清了语义层和问数系统的分工。从本章开始，使用 Foggy 把这条链路跑起来：先认识引擎，再选定工具、接入样例数据。

本章先完成环境与数据接入，暂不建模。

先认识 Foggy

Foggy Data MCP Bridge 是面向 AI 数据分析的语义层引擎。开发者用语义模型定义业务字段、指标和关联，AI 根据公开的查询元数据表达问题；引擎依据定义校验、组织查询并生成物理 SQL，再交数据库执行。

它可以作为独立服务连接已有业务库，也提供 CLI、API 和 MCP 等调用入口。本章下载发行包在本机启动服务，先接一份 SQLite 数据，不需要建设数仓或编译源码。

先对照一下前文概念与 Foggy 中的名称：

名称	在本项目中的作用
TM（表模型）	定义数据来源、物理字段映射、关联与指标
QM（查询模型）	基于 TM 组织查询面，选择公开字段并配置查询规则
DSL	表达本次要查的字段、条件、分组和排序
Foggy 服务（Runtime）	加载语义模型并处理查询
CLI	让 AI 通过命令调用接入、探查、建模校验和查询等能力
Skill	告诉 AI 操作顺序、接口用法和出错时怎样处理

下载的发行包叫 Launcher，启动后运行的服务叫 Runtime，后文统一称为“Foggy 服务”。

本章由 AI 助手调用 CLI，Foggy 服务端无须另配 AI Provider。

用 Chinook 做一个小系统

Chinook 模拟一家数字媒体商店，包含客户、账单、曲目、专辑和播放列表等数据，共 11 张业务表。它足够小，查错时还能翻回原始记录，不至于刚开始学问数，就先被数据库劝退。

样例下载：Chinook v1.4.5 SQLite 文件。后面的任务会让 Codex 下载并使用副本。整个项目的第一版范围如下：

项目	约定
使用者	开发者先验证，后续交业务用户在线问数

项目	约定
数据	先用客户、账单、账单明细、曲目和内容分类五张表
期间	固定为 2012 年；月间比较先选十月、十一月
首批问题	月度账单金额、客户排名、账单国家占比、内容分类销售、平均账单金额
暂不回答	缺少相应数据和定义的利润、回款、退款，以及对未来的确定性预测
项目验收	先核对结构化查询，再检查终端自然语言问答，最后检查浏览器问答

账单金额先看 `Invoice.Total`，明细金额先看 `UnitPrice × Quantity`，后续再核对两者是否一致。金额单位和状态重新确认，不套用技术篇的分转元或 `ISSUED` 条件。

把准备工作交给 Codex

这里用 Codex，主要因为我已有订阅，可以先用订阅内的额度，省一些额外调用 API 的 token 开销。也可以换成 DeepSeek Harness 之类的 Agent 工具：能读取操作说明、读写本地文件并执行 Foggy CLI，就可以按同样的任务推进。使用 ChatGPT 桌面端时，也要选择支持本机工具执行的任务。

下面以 Codex 为例，读者负责发出任务和确认结果，下载、安装与命令执行交给它。

读者准备什么

在 Codex 中打开一个独立练习目录，例如 `D:\ai-data`。Skill、数据库样例和 Python、Java 等依赖，都让 Codex 按下面的任务准备。

发给 Codex 什么

在练习目录中发送：

> 请从下面的入口获取 Foggy Skill，并按它的指引准备环境： https://raw.githubusercontent.com/foggy-projects/foggy-ai-analysis/main/stack/stable.json
>
> 再下载 Chinook v1.4.5 SQLite 样例，保存在当前目录，用副本接入，确认连接成功并列出表。先不建模。

Codex 会怎样处理

Codex 大致会按以下顺序处理：

1. **下载** `stable.json`。 读取当前推荐的 Skill、CLI 和服务发行包版本，以及对应的下载入口，确定本次使用的组件组合。
2. **下载并读取 Skill。** 获取正式 ZIP，校验、解压后阅读 `SKILL.md` 和接入指引。后面的安装顺序、命令用法和验证方法，都从这里获得。

3. **准备 CLI 和服务资源。** 按 Skill 指引检查 Python、Java，安装 CLI 和 Skill，再下载 Foggy 服务的 JAR 与启动脚本等资源。已有组件先检查，合适的可以复用。

4. **启动并验证服务。** 选择空闲端口启动，等待服务就绪，再查询 capabilities，确认当前服务支持数据源接入和表发现。

5. **下载并接入 Chinook。** 从官方链接下载 SQLite 样例，保留原件、使用副本。添加数据源、测试连接，再将它绑定到项目的 Namespace（命名空间），让后续操作使用同一份数据。

6. **列出表并汇总结果。** 分别通过 Namespace 和指定数据源发现表，核对两份清单，并报告服务地址、文件位置和执行结果。失败或警告也应一并说明。

完成后核对什么

读者重点看下面几项。数据源名和 Namespace 可以由 Codex 创建，表中使用本次示例名称：

检查项	应看到的结果
服务	有明确地址，就绪检查成功，所需能力可用
数据源	connected: true，连接指向 Chinook 练习副本
项目绑定	chinook → chinook-sqlite，指向刚才测试的数据源
表清单	两种方式发现的表一致，共 11 张业务表

本次结果与小结

本次实测沿用已有 Skill，CLI 和服务发行包均为 **0.1.24**，服务地址为 http://127.0.0.1:18067，只在本机提供服务。连接和绑定通过，发现 11 张业务表：

Album、Artist、Customer、Employee、Genre、Invoice、InvoiceLine、
MediaType、Playlist、PlaylistTrack、Track

重启本项目服务后，没有重新添加或绑定数据源，连接和表发现检查仍通过。验收结果与重启记录已保存；安装命令、版本细节和复跑方法放在配套环境说明中。

本章 Foggy 知识点

- **服务与 CLI：** 通过 Skill 准备发行资源，启动 Foggy 服务，用就绪检查和 capabilities 确认所需能力。
- **数据源与 Namespace：** 添加、测试 SQLite 数据源，将 Namespace 绑定到练习副本。
- **表发现：** 通过 Namespace 或指定数据源列出业务表，确认接入结果。

下一章检查五张核心表的关系、日期和金额口径，第八章再建立语义模型并完成首次问数。

第 7 章 让 AI 探查 Chinook 并确认业务口径

上一章已经接通 Chinook，发现了 11 张表。本章沿用同一环境，检查表关系、字段含义和统计方法，为下一章建模做准备。

把探查任务交给 Codex

在原来的练习目录中继续发送：

> 请用 Foggy CLI 看看 Chinook 的表和数据，整理表关系、日期范围和金额含义，核对账单与明细金额，保存结果供我确认，先不建模。

Codex 大致会这样处理：

1. **复查连接。** 确认服务可用，Namespace 仍指向上一章的 Chinook 副本。
2. **读取表结构。** 通过 `tables inspect` 查看字段、类型、主键和外键，确定一行数据代表什么。
3. **检查实际值。** 用 `sql query` 执行少量只读查询，查看日期、分类和金额，检查关联缺失与重复累计。
4. **核对并保存结果。** 逐张核对账单与明细金额，保存统计结果和需要人确认的字段含义。

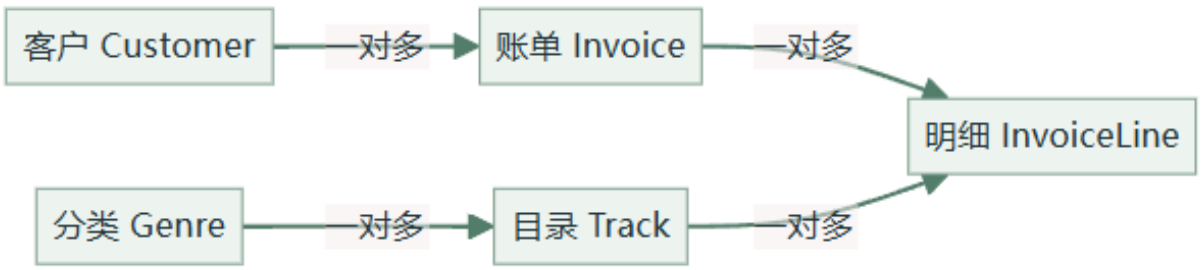
这里的 SQL 用于探查和独立核对。结构与数值可以验证，字段对应什么业务含义仍需人确认，不能看到字段名就替业务下结论。

先看一行数据代表什么

先看与账单分析有关的五张表：

表	一行代表什么	记录数
Customer	一位客户	59
Invoice	一张账单	412
InvoiceLine	一条账单明细	2240
Track	一条曲目或媒体目录记录	3503
Genre	一种内容分类	25

CLI 读到的关联可以简化为：



查询与建模流程

例如，Invoice.CustomerId 指向客户，InvoiceLine.InvoiceId 指向账单。本章检查了图中四条核心关联，没有发现空关联键或找不到目标记录的情况。

Genre 的实际值既有 Rock、Jazz，也有 TV Shows、Drama，因此建模时称为“内容分类”更准确。表名给了线索，取值才能说明它实际装了什么。

本次查到的账单日期覆盖 2009-01-01 至 2013-12-22，后续练习选用其中的 2012 年。虽然日期字段声明为 DATETIME，实际保存为文本，格式是 YYYY-MM-DD HH:MM:SS；本例按记录日期统计，不自行做时区转换。

金额要逐张对，不只看总额

从实际记录中取一张账单来看：编号 252，金额为 1.98，有两条明细：

明细 ID	单价	数量	明细金额
1367	0.99	1	0.99
1368	0.99	1	0.99

本例的账单金额取 Invoice.Total，明细金额取 InvoiceLine.UnitPrice × Quantity。两条明细合计 1.98，与账单金额一致。Track 中也有 UnitPrice，统计交易时使用账单明细上的单价。

本次把全部 412 张账单逐张对账，没有发现差额，全库合计为 2328.60。金额按样例原始单位使用，不除以 100；五张核心表没有币种字段，结果也不自行标成人民币或美元。

第二章讲过的重复累计，在这里也能复现：2012 年实际账单金额是 477.53，关联明细后直接累加账单金额，却变成 4256.25。按内容分类分析时，应从明细金额汇总。

确认字段含义与统计口径

建模前，先确认字段代表什么，以及金额、数量等怎么统计。下面是根据表结构和数据整理的说明，确认后再用于建模：

分析内容	字段含义与统计口径
统计日期	使用 Invoice.InvoiceDate；2012 年范围为大于等于 2012-01-01、小于 2013-01-01
账单金额与张数	每张账单只累计一次 Total，按 InvoiceId 计账单数
平均账单金额	当前范围的账单金额合计 ÷ 账单数
明细金额与数量	逐条累加明细的 UnitPrice × Quantity 和 Quantity
客户	用 CustomerId 区分，组合 FirstName、LastName 展示名称
账单国家	使用账单地址中的 BillingCountry，与客户地址中的 Customer.Country 区分
内容分类	通过 InvoiceLine → Track → Genre 关联，按 Genre.Name 分组

还有两处容易被 AI“偷懒”处理：样例中所有 Quantity 都是 1，数量仍应按求和定义；账单国家与客户国家目前全部相同，建模时也要保留两者的含义区别。数据里暂时相等，不代表以后可以随便互换。

保存核对结果，供建模后比较

本次分别统计账单与明细，保存了 2012 年十二个月的结果。这份核对结果就是后续比较用的“基线”。下面摘出几项，金额均为样例原始单位：

范围	账单数	明细数	账单金额	明细金额
2012 年全年	83	447	477.53	477.53
2012 年十月	7	38	42.62	42.62
2012 年十一月	7	38	37.62	37.62

这次还从只读数据文件逐笔取数，用 Python Decimal 另算了一遍，结果一致。下一章查询语义模型时，就可以与这份基线比较，而不是拿 AI 的两次回答互相作证。

如果真实数据出现日期混杂、重复键或关联缺失，先做必要的数据整理，再建模；这也是 ETL 会介入的地方。本例没有新增清洗步骤，原件与副本均未修改。

探查记录与复跑方法、完整基线和验收结果已保存。

本章 Foggy 知识点

- **表结构探查：** 用 tables inspect 查看字段、类型、主外键，梳理表关系和数据粒度。
- **建模前查数：** 用 sql query 做只读探查，检查实际取值、日期范围和关联后的金额；此时尚未创建语义层模型。

- **建模依据：** 将探查结果整理成待确认的字段含义、统计口径和核对基线，供后续建模与查询验证使用。

下一章据此编写 TM（表模型）和 QM（查询模型），完成第一次语义问数。

第 8 章 让 AI 生成语义模型并完成首次问数

上一章完成了数据探查和口径整理，还没有创建语义层模型。本章先创建账单分析的语义层模型，再查询客户排名；随后增加明细模型，与原始数据核对。

第一步：创建账单分析的语义层模型

在原来的练习目录和会话中继续发送：

> 请按上一章整理的口径创建账单分析的语义层模型，校验并加载，列出可查询的字段。

如果换了会话，把上一章保存的字段说明和核对结果一起交给 Codex。

Codex 需要先读取 foggy-ai-analysis Skill，编写两个文件：

文件	本次定义什么
ChinookInvoiceModel.tm	以 Invoice 为主表，关联客户，定义账单日期、金额和张数
ChinookInvoiceQueryModel.qm	公开本次分析要用的字段，并说明适用范围

TM、QM 使用类似 JavaScript 的配置语法。下面是本次验证通过的完整源码，可展开查看。

账单 TM：ChinookInvoiceModel.tm（完整源码）

打开源文件

```
export const model = {
    name: 'ChinookInvoiceModel',
    caption: 'Chinook 账单',
    description: '一行是一张账单，不关联明细后累计账单金额。金额沿用样例原始单位，币种未指定。',
    tableName: 'Invoice',
    idColumn: 'InvoiceId',
    properties: [
        { column: 'InvoiceId', name: 'invoiceId', caption: '账单 ID', type: 'INTEGER' },
        { column: 'CustomerId', name: 'customerId', caption: '客户 ID', type: 'INTEGER',
          description: '客户识别键，排名时与姓名一起分组，避免同名客户合并。' },
        { column: 'InvoiceDate', name: 'invoiceDate', caption: '账单日期', type: 'DATETIME',
          description: '按账单记录日期统计，不自行做时区转换。' },
        { column: 'BillingCountry', name: 'billingCountry', caption: '账单国家', type: 'STRING',
          description: '账单地址中的国家，不是客户地址国家。' }
    ],
    dimensions: [
        {
            name: 'customer', caption: '客户', tableName: 'Customer',
            foreignKey: 'CustomerId', primaryKey: 'CustomerId',
            captionColumn: 'LastName',
            keyCaption: '客户 ID',
            description: '用客户 ID 区分客户，姓名仅用于展示。',
            properties: [
                { column: 'FirstName', name: 'fullName', caption: '客户姓名', type: 'STRING',
                  description: 'FirstName 与 LastName 拼接；展示时与客户 ID 一起使用。',
                  formulaDef: { builder: (alias) => `${alias}.FirstName || ' ' || ${alias}.LastName` } },
                { column: 'Country', name: 'country', caption: '客户国家', type: 'STRING' }
```

```
            ]
        }
    ],
    measures: [
        { column: 'Total', name: 'invoiceAmount', caption: '账单金额', type: 'MONEY', aggregation: 'sum',
          description: '每张账单只累计一次。样例原始单位，不除以 100，不指定币种。' },
        { column: 'InvoiceId', name: 'invoiceCount', caption: '账单数', type: 'INTEGER', aggregation: 'count',
          description: '按账单 ID 计数，只在账单分析中统计。' }
    ]
};
```

账单 QM：ChinookInvoiceQueryModel.qm（完整源码）

打开本章版本

```
const invoice = loadTableModel('ChinookInvoiceModel');

export const queryModel = {
    name: 'ChinookInvoiceQueryModel',
    caption: 'Chinook 账单分析',
    description: '按账单日期、客户和账单国家分析账单金额与张数。金额沿用样例原始单位，不指定币种、不做分元换算，不代表
  利润、回款或退款。分类分析请用明细分析，不按分类分摊账单总额。',
    loader: 'v2',
    model: invoice,
    columnGroups: [
        { caption: '账单与客户', items: [
            { ref: invoice.invoiceId },
            { ref: invoice.invoiceDate },
            { ref: invoice.billingCountry },
            { ref: invoice.customerId },
            { ref: invoice.customer$fullName },
            { ref: invoice.customer$country }
        ] },
        { caption: '账单指标', items: [
            { ref: invoice.invoiceAmount },
            { ref: invoice.invoiceCount }
        ] }
    ],
    accesses: []
};
```

这里将 Total 映射为 invoiceAmount，姓名由 FirstName、LastName 拼接。QM 中的 loadTableModel 引用已有 TM，columnGroups 决定哪些字段可以查询。

校验并加载

Codex 接着通过 CLI 完成三步：

1. **校验模型。** 用 models validate 检查 TM/QM，出错先修正。
2. **加载模型。** 将模型目录登记为 Bundle（模型资源包），再用 models refresh 加载。
3. **确认公开字段。** 用 models describe 实时读取查询说明，确认字段符合定义，供下一步查询使用。

AI 查询时读到什么

Markdown 查询说明由引擎根据当前已加载的 TM/QM 代码自动生成。AI 需要时通过 CLI 实时获取，读者无须提前生成或维护 Markdown 文件。 修改 TM/QM 后，先校验、刷新，再读取更新后的说明。

下面是 `models describe ChinookInvoiceQueryModel --format markdown` 的实际返回节选，对应第三章所讲的查询元数据。保存这份返回内容只是为了方便对照阅读。

实时获取的账单 QM 查询说明（Markdown 节选）

完整返回记录

```
# ChinookInvoiceQueryModel - Chinook 账单分析

## 模型信息
- 表名: Invoice
- 主键: InvoiceId
- 说明: 按账单日期、客户和账单国家分析账单金额与张数。金额沿用样例原始单位，不指定币种、不做分元换算，不代表利润、回款或退款。分类分析请用明细分析，不按分类分摊账单总额。

## 时间维度与字段 (Time Dimensions & Fields)
> **提示**: 在进行时间趋势分析、同环比或窗口函数(Window Functions)时，请优先使用 `business_date` 角色的字段。

| 字段名 | 名称 | 类别 | 时间角色 | 推荐用途 | 说明 |
|--------|------|------|----------|----------|------|
| invoiceDate | 账单日期 | 属性 | business_date | - | 按账单记录日期统计，不自行做时区转换。 |

## 属性字段
| 字段名 | 名称 | 类型 | 说明 |
|--------|------|------|------|
| invoiceId | 账单 ID | 文本 |  |
| billingCountry | 账单国家 | 文本 | 账单地址中的国家，不是客户地址国家。 |
| customerId | 客户 ID | 文本 | 客户识别键，排名时与姓名一起分组，避免同名客户合并。 |
| customer$fullName | 客户姓名 | 文本 | FirstName 与 LastName 拼接；展示时与客户 ID 一起使用。 |
| customer$country | 客户国家 | 文本 |  |

## 度量字段
| 字段名 | 名称 | 类型 | 聚合 | 说明 |
|--------|------|------|------|------|
| invoiceAmount | 账单金额 | 金额 | SUM | 每张账单只累计一次。样例原始单位，不除以 100，不指定币种。 |
| invoiceCount | 账单数 | 文本 | COUNT | 按账单 ID 计数，只在账单分析中统计。 |
```

查询时，AI 根据说明中的模型名称、字段含义和聚合方式编写 DSL。比如 `customer$fullName` 是公开的语义字段，数据库里没有新增这一列。

当前版本的 Markdown 将部分整数字段显示为“文本”，实际数据类型以结构化 `describe` 输出中的 `type` 为准。

第二步：查询客户排名

确认模型加载成功，公开字段符合上一章的定义后，再发送查询任务：

> 请查出 2012 年账单金额最高的五位客户，核对并保存结果。

Codex 使用 `foggy-semantic-query` Skill，通过 CLI 指定 `ChinookInvoiceQueryModel`，生成下面的 DSL 请求：

```
{
  "columns": ["customerId", "customer$fullName", "sum(invoiceAmount) as totalAmount"],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "groupBy": ["customerId", "customer$fullName"],
```

```
  "orderBy": ["-totalAmount", "customerId"],
  "limit": 5
}
```

请求按客户汇总 2012 年金额，金额降序，同额时按客户 ID 升序，取前五位。表关联与姓名拼接由语义模型处理。

请求先经过 query validate，通过后再 query execute。本次实际返回：

客户 ID	客户姓名	账单金额
26	Richard Cunningham	25.84
34	João Fernandes	24.77
13	Fernanda Ramos	24.75
51	Joakim Johansson	24.75
55	Mark Taylor	22.77

用 Python Decimal 从原始记录独立累计后，姓名、金额和排序均一致。回答保留“2012 年”“账单金额”和样例原始单位，不改称利润或自行补上币种。

第三步：创建明细分析的语义层模型

确认账单结果后，继续发送：

> 请按上一章整理的口径创建明细分析的语义层模型，校验并加载，列出可查询的字段。

Codex 增加明细 TM/QM，沿用第一步的校验、加载流程。两个查询模型的基础记录不同：

查询模型	基础记录	金额来源
ChinookInvoiceQueryModel	一张账单	Invoice.Total
ChinookInvoiceLineQueryModel	一条明细	InvoiceLine.UnitPrice × Quantity

明细 TM/QM 的完整源码如下：

明细 TM：ChinookInvoiceLineModel.tm（完整源码）

打开本章源码快照

```
export const model = {
    name: 'ChinookInvoiceLineModel',
    caption: 'Chinook 账单明细',
    description: '一行是一条账单明细。分类分析累计交易明细金额，不累计账单总金额，不改用目录单价。',
    tableName: 'InvoiceLine',
    idColumn: 'InvoiceLineId',
    properties: [
        { column: 'InvoiceLineId', name: 'invoiceLineId', caption: '明细 ID', type: 'INTEGER' },
```

```
        { column: 'InvoiceId', name: 'invoiceId', caption: '账单 ID', type: 'INTEGER' },
        { column: 'TrackId', name: 'trackId', caption: '曲目或媒体 ID', type: 'INTEGER' },
        { column: 'UnitPrice', name: 'unitPrice', caption: '交易单价', type: 'MONEY',
          description: '账单明细上的交易单价，不是 Track 目录单价。样例原始单位，币种未指定。' }
    ],
    dimensions: [
        {
            name: 'invoice', caption: '账单', tableName: 'Invoice',
            foreignKey: 'InvoiceId', primaryKey: 'InvoiceId', captionColumn: 'InvoiceId',
            properties: [
                { column: 'InvoiceDate', name: 'invoiceDate', caption: '账单日期', type: 'DATETIME',
                  description: '按账单记录日期统计，不自行做时区转换。' },
                { column: 'BillingCountry', name: 'billingCountry', caption: '账单国家', type: 'STRING' },
                { column: 'CustomerId', name: 'customerId', caption: '客户 ID', type: 'INTEGER' }
            ],
            dimensions: [
                {
                    name: 'customer', caption: '客户', tableName: 'Customer',
                    foreignKey: 'CustomerId', primaryKey: 'CustomerId', captionColumn: 'LastName',
                    properties: [
                        { column: 'FirstName', name: 'fullName', caption: '客户姓名', type: 'STRING',
                          description: 'FirstName 与 LastName 拼接；与客户 ID 一起使用。',
                          formulaDef: { builder: (alias) => `${alias}.FirstName || ' ' || ${alias}.LastName` } }
                    ]
                }
            ]
        },
        {
            name: 'track', caption: '曲目或媒体', tableName: 'Track',
            foreignKey: 'TrackId', primaryKey: 'TrackId', captionColumn: 'Name',
            properties: [
                { column: 'Name', name: 'title', caption: '曲目或媒体名称', type: 'STRING' }
            ],
            dimensions: [
                {
                    name: 'genre', caption: '内容分类', tableName: 'Genre',
                    foreignKey: 'GenreId', primaryKey: 'GenreId', captionColumn: 'Name',
                    description: '包含音乐和影视内容；分类缺失不应丢弃交易金额。',
                    properties: [
                        { column: 'GenreId', name: 'genreId', caption: '分类 ID', type: 'INTEGER' },
                        { column: 'Name', name: 'genreName', caption: '内容分类名称', type: 'STRING' }
                    ]
                }
            ]
        }
    ],
    measures: [
        { column: 'UnitPrice', name: 'lineAmount', caption: '明细金额', type: 'MONEY', aggregation: 'sum',
          description: '交易单价乘数量，沿用样例原始单位，不除以 100，不指定币种。',
          formulaDef: { builder: (alias) => `${alias}.UnitPrice * ${alias}.Quantity` } },
        { column: 'Quantity', name: 'salesQuantity', caption: '销售数量', type: 'INTEGER', aggregation: 'sum',
          description: '累计 Quantity；本样例全部为 1，也不能将数量定义成明细条数。' }
    ]
};
```

明细 QM：ChinookInvoiceLineQueryModel.qm（完整源码）

打开本章源码快照

```
const line = loadTableModel('ChinookInvoiceLineModel');

export const queryModel = {
    name: 'ChinookInvoiceLineQueryModel',
```

```
    caption: 'Chinook 明细分析',
    description: '按账单日期、客户、曲目或媒体、内容分类分析明细金额和销售数量。每行是一条明细，不提供可累计的账单总金额。金额沿用样例原始单位，不指定币种、不做分元换算，不代表利润、回款或退款。',
    loader: 'v2',
    model: line,
    columnGroups: [
        { caption: '交易明细', items: [
            { ref: line.invoiceLineId },
            { ref: line.invoiceId },
            { ref: line.trackId },
            { ref: line.unitPrice },
            { ref: line.invoice$invoiceDate },
            { ref: line.invoice$billingCountry },
            { ref: line.invoice$customerId },
            { ref: line.invoice.customer$fullName },
            { ref: line.track$title },
            { ref: line.track.genre$genreId },
            { ref: line.track.genre$genreName }
        ] },
        { caption: '明细指标', items: [
            { ref: line.lineAmount },
            { ref: line.salesQuantity }
        ] }
    ],
    accesses: []
};
```

lineAmount 的 formulaDef 定义交易单价乘数量，查询时直接使用这个字段。明细 QM 不公开可累计的账单总金额，减少误用。

加载后，AI 同样实时获取明细 QM 的说明。嵌套维度的公开名称以返回结果为准，例如源码中的 line.track.genre$genreName，对应查询字段 track_genre$genreName：

实时获取的明细 QM 查询说明（Markdown 节选）

完整返回记录

```
# ChinookInvoiceLineQueryModel - Chinook 明细分析

## 模型信息
- 表名: InvoiceLine
- 主键: InvoiceLineId
- 说明: 按账单日期、客户、曲目或媒体、内容分类分析明细金额和销售数量。每行是一条明细，不提供可累计的账单总金额。金额沿用样例原始单位，不指定币种、不做分元换算，不代表利润、回款或退款。

## 时间维度与字段 (Time Dimensions & Fields)
> **提示**: 在进行时间趋势分析、同环比或窗口函数(Window Functions)时，请优先使用 `business_date` 角色的字段。

| 字段名 | 名称 | 类别 | 时间角色 | 推荐用途 | 说明 |
|--------|------|------|----------|----------|------|
| invoice$invoiceDate | 账单日期 | 属性 | business_date | - | 按账单记录日期统计，不自行做时区转换。 |

## 属性字段
| 字段名 | 名称 | 类型 | 说明 |
|--------|------|------|------|
| invoiceLineId | 明细 ID | 文本 |  |
| invoiceId | 账单 ID | 文本 |  |
| trackId | 曲目或媒体 ID | 文本 |  |
| unitPrice | 交易单价 | 金额 | 账单明细上的交易单价，不是 Track 目录单价。样例原始单位，币种未指定。 |
| invoice$billingCountry | 账单国家 | 文本 |  |
| invoice$customerId | 客户 ID | 文本 |  |
| invoice_customer$fullName | 客户姓名 | 文本 | FirstName 与 LastName 拼接；与客户 ID 一起使用。 |
```

```
| track$title | 曲目或媒体名称 | 文本 | |
| track_genre$genreId | 分类 ID | 文本 | |
| track_genre$genreName | 内容分类名称 | 文本 | |

## 度量字段
| 字段名 | 名称 | 类型 | 聚合 | 说明 |
|--------|------|------|------|------|
| lineAmount | 明细金额 | 金额 | SUM | 交易单价乘数量，沿用样例原始单位，不除以 100，不指定币种。 |
| salesQuantity | 销售数量 | 文本 | SUM | 累计 Quantity；本样例全部为 1，也不能将数量定义成明细条数。 |
```

第四步：按内容分类查询

确认明细模型加载成功、日期与分类字段可用后，再发送：

> 请按内容分类统计 2012 年的明细金额，核对全部分类的金额合计是否与同期账单一致，保存结果。

这次使用 ChinookInvoiceLineQueryModel，同时返回销售数量供核对。实际执行的 DSL 如下：

```
{
  "columns": [
    "track_genre$genreId",
    "track_genre$genreName",
    "sum(lineAmount) as totalAmount",
    "sum(salesQuantity) as quantity"
  ],
  "slice": [
    {"field": "invoice$invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "groupBy": ["track_genre$genreId", "track_genre$genreName"],
  "orderBy": ["-totalAmount", "track_genre$genreId"],
  "limit": 100
}
```

日期仍按账单日期筛选，金额改为按分类累计 lineAmount。limit: 100 足以返回本例全部分类，便于核对合计。请求同样先校验，再执行。

本次返回 22 个有交易的分类，下面节选前三项：

分类 ID	内容分类	明细金额	销售数量
1	Rock	162.36	164
3	Metal	65.34	66
7	Latin	63.36	64

全部 22 类金额逐类核对一致，合计 477.53，与同期全部账单金额及第七章基线一致。

本次 SQLite 聚合的部分返回值带有浮点尾数，原始响应保留，金额按两位小数核对和展示。

写错字段时，先修正再执行

这次还故意把客户姓名写成不存在的 clientName。query validate 返回 FIELD_NOT_FOUND，提示重新读取字段说明，错误查询没有执行。

重新 describe，改回 customer$fullName，再次校验、执行，排名一致。字段合法只代表可以查询，是否符合用户问题仍需检查。

完成后检查什么

本次 4 个模型文件校验通过，两个查询模型均可查询，结果与独立核对一致。

读者重点检查：定义是否符合上一章的口径，查询期间是否正确，结果能否与已有基线对上。

完整模型、请求和验证记录见配套实操说明。

本章 Foggy 知识点

- **TM 与 QM：** TM 定义字段映射、维度关联、公式与聚合，QM 选择公开字段；账单与明细按各自粒度建模。
- **模型加载与元数据：** 校验 TM/QM，将目录登记为 Bundle 后刷新加载，用 models describe 实时获取引擎自动生成的查询说明。
- **基础 DSL：** 用 columns、slice、groupBy、orderBy 和 limit 表达返回字段、筛选、分组、排序与条数。
- **查询校验：** 先 query validate 再 query execute；字段错误时重新读取元数据，修正后再执行。

下一章沿用账单查询模型，学习普通筛选、条件组合与聚合后筛选。

第 9 章 让 AI 正确筛选数据：slice 与 having

第八章已经建立了语义层模型。本章沿用 ChinookInvoiceQueryModel，学习普通筛选、条件组合和聚合后筛选。金额仍用样例原始单位，国家均指账单地址中的国家。

先筛选原始记录：slice

在原来的练习目录和会话中发送：

> 查一下 2012 年账单国家为美国的记录。

Codex 读取 foggy-semantic-query Skill，通过 CLI 实时获取查询模型说明，确认日期、国家和金额字段。Markdown 由引擎自动生成，不用读者提前准备。

字段说明能告诉 AI 用 billingCountry，却不一定告诉它“美国”存的是 USA 还是 United States。本次先用语义查询列出国家取值，确认美国为 USA、加拿大为 Canada，再构造请求：

```
{
  "columns": ["invoiceId", "invoiceDate", "billingCountry", "invoiceAmount"],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]},
    {"field": "billingCountry", "op": "=", "value": "USA"}
  ],
  "orderBy": ["invoiceId"],
  "limit": 100
}
```

slice 筛选原始账单，数组中的两个条件默认按 AND 组合：日期在 2012 年内，且账单国家为美国。这里查询的是逐张账单，没有分组求和。

请求经 query validate 校验后执行，共返回 **21 张账单**，以下节选前三张：

账单 ID	账单日期	账单金额
255	2012-01-24	5.94
256	2012-01-27	8.91
265	2012-03-11	0.99

本次结果未分页截断，逐条与原始记录核对一致。不是字段名写对了就万事大吉，筛选值也得先认准。

组合条件：AND、in 与 OR

继续发送：

> 2012 年美国或加拿大的账单，单张金额超过 10 的有哪些？

保留上一条请求的返回字段和排序，将 slice 换成：

```
[
  {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]},
  {"field": "billingCountry", "op": "in", "value": ["USA", "Canada"]},
  {"field": "invoiceAmount", "op": ">", "value": 10}
]
```

三个条件要同时满足：国家条件内部使用 `in`，表示美国、加拿大任选其一。本次校验、执行后返回全部 5 张账单：

账单 ID	账单国家	账单金额
278	Canada	13.86
298	USA	10.91
299	USA	23.86
311	USA	11.94
320	USA	13.86

同一字段的多个候选值用 `in` 就很简洁。需要明确表达“或者”时，也可以把国家条件替换为：

```
{
  "$or": [
    {"field": "billingCountry", "op": "=", "value": "USA"},
    {"field": "billingCountry", "op": "=", "value": "Canada"}
  ]
}
```

日期与金额条件仍放在 `$or` 外面，让两个国家都受它们约束。本次分别执行 `in` 和 `$or` 两种写法，返回的账单完全一致。

筛选汇总结果：having

再换一个问题：

> 2012 年所有客户中，累计账单金额超过 20 的有哪些？

这次范围是所有国家，筛选对象也从“一张账单”变成“一个客户的全年累计金额”。Codex 构造的请求如下：

```
{
  "columns": ["customerId", "customer$fullName", "sum(invoiceAmount) as totalAmount"],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "groupBy": ["customerId", "customer$fullName"],
  "having": [
    {"field": "totalAmount", "op": ">", "value": 20}
  ],
  "orderBy": ["-totalAmount", "customerId"],
  "limit": 100
}
```

执行顺序可以理解为：slice 选出全年账单 → 按客户累计 → having 筛选累计金额 → 排序返回。

having 引用的是本次求和的别名 totalAmount。实际返回 **5 位客户**，金额和顺序与第八章的前五位恰好一致；这次数量由金额条件决定，查询上限是 100。

以 Richard Cunningham 为例，全年累计是 25.84。如果错把“金额超过 20”放进 slice，先丢掉小额账单再汇总，就只剩 23.86，其他四位客户也会漏掉。两种写法都能执行，却回答了不同的问题：

筛选方式	本次结果
单张账单金额超过 20	1 张账单，金额 23.86
客户全年累计金额超过 20	5 位客户，最高累计 25.84

所以，slice 与 having 的差别是筛选发生在哪一步。校验能检查请求是否合法，不能替用户决定“超过 20”指单张还是累计；提问含糊时，AI 应先确认。

文字搜索与排除

发送：

> 查一下 2012 年账单国家名称包含 United 的记录。

保留逐张账单查询的字段、日期和排序，增加这个 slice 条件：

```
{"field": "billingCountry", "op": "like", "value": "United"}
```

本次 like 自动生成 %United%，返回英国的 2 张账单，ID 为 261、283。这里匹配的是数据库中的英文取值；中文名称与原始值的字典映射，第十五章再讲。大小写和通配符行为需按所用数据库验证。

要排除美国和加拿大，则改用：

```
{"field": "billingCountry", "op": "not in", "value": ["USA", "Canada"]}
```

同样限定 2012 年，实际得到 53 张账单。not in 不等于“包括所有未填写的记录”，空值应单独处理。

未填写，用空值条件查询

现有 QM 尚未公开账单的州或地区。先发送：

> 给账单分析增加账单州或地区字段，校验并加载。先不查询。

Codex 在 ChinookInvoiceModel.tm 的 properties 中增加：

```
{ column: 'BillingState', name: 'billingState', caption: '账单州或地区', type: 'STRING',
  description: '账单地址中的州或地区；NULL 表示未填写，不能据此推断客户没有所属地区。' }
```

再在账单 QM 的 columnGroups →“账单与客户”中增加 { ref: invoice.billingState }。校验、刷新后，实时元数据应出现 billingState。

确认加载成功，再发送：

> 查一下 2012 年没有填写账单州或地区的账单。

请求选择 invoiceId、billingState，保留日期范围与 ID 排序，增加：

```
{"field": "billingState", "op": "is null"}
```

本次返回 39 张；改为 is not null 返回 44 张，两者合计为全年 83 张。响应中空值字段可能省略，不能因此认定请求没有选择它。

NULL、空字符串和 0 是不同的值。当前版本把 value: "" 的等值筛选判为无效请求，不能拿它代替 is null；该错误请求未继续执行。筛选文本值 "0" 返回空列表，也不代表未填写。

核对什么

原有实操通过公开 CLI 完成了 8 个查询用例，包含取值确认、三个业务问题、等价 OR 写法、单张与累计对照，以及空日期范围。用只读原始记录和 Python Decimal 独立核对，账单 ID、日期、国家、金额、客户范围及排序均一致；该组查询未修改模型或数据。

读者重点看三件事：**筛选值是否真实存在，AND / OR 的范围是否正确，条件应该在聚合前还是聚合后生效。** 如果结果为空，也要如实说明没有符合条件的记录；本次查询无数据的日期范围时，执行成功，返回空列表。

完整请求、响应和复跑方式见第九章实操记录。

本轮另补文字、排除与空值的 5 项查询及 1 项预期校验拒绝，均与只读原始记录核对；只新增公开州字段，数据未修改。见基础功能补充记录。

本章 Foggy 知识点

- **聚合前筛选：** slice 筛选原始记录，支持日期区间、等值、数值比较和 in。
- **条件组合：** 条件数组默认按 AND 组合，$or 表达或者，嵌套位置决定哪些条件共同生效。
- **聚合后筛选：** having 按聚合别名筛选汇总结果，区别于单张记录的金额筛选。
- **文字与空值：** like、not in、is null / is not null，区分未填写、空字符串与实际值。

下一章继续聚合与计算指标。

第 10 章 聚合与计算指标：数清楚，再算平均和占比

上一章解决“筛选哪些数据”，本章继续使用 ChinookInvoiceQueryModel，学习求和、计数、去重计数和计算字段。先在请求中完成临时计算，再把常用的平均金额固化为语义层指标。

账单数，不等于客户数

在原来的练习目录和会话中发送：

> 统计一下 2012 年的账单金额、账单数和涉及的客户数。

Codex 读取 foggy-semantic-query Skill，实时获取模型说明，确认金额、账单 ID 和客户 ID，再构造请求：

```
{
  "columns": [
    "sum(invoiceAmount) as totalAmount",
    "count(invoiceId) as billCount",
    "count_distinct(customerId) as buyerCount"
  ],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "limit": 2
}
```

三种聚合各管一件事：sum 累计金额，count 数账单，count_distinct 对客户 ID 去重。同一个客户有多张账单，客户数仍只算一位。

请求先校验，再执行，实际返回：

账单金额	账单数	涉及的客户数
477.53	83	47

模型以一张账单为基础记录，因此可以直接数 invoiceId。样例客户表有 59 位客户，但本次统计的是 2012 年有账单的 47 位，不能拿客户表总行数代替。

这里没有 groupBy，结果是整个筛选范围的一行汇总；limit 约束返回行数，不会把参与求和的账单限制成两张。

平均金额：先确认分子与分母

继续发送：

> 2012 年平均每张账单金额是多少？

本例定义是“同期账单金额合计 ÷ 账单数”。下面通过 calculatedFields 把它写成本次查询的计算字段：

```
{
  "columns": [
    "sum(invoiceAmount) as totalAmount",
    "count(invoiceId) as billCount",
    "temporaryAverage"
  ],
  "calculatedFields": [
    {
      "name": "temporaryAverage",
      "expression": "sum(invoiceAmount) / nullif(count(invoiceId), 0)"
    }
  ],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "limit": 2
}
```

name 是本次计算字段的名字，expression 是由引擎解析、执行的表达式。temporaryAverage 只在这次请求中使用，不会给 QM 或数据库新增字段。

实际结果为 477.53 ÷ 83 = 5.753373…，展示为 5.75，金额仍用样例原始单位。当前模型一行一张账单，直接使用 avg(invoiceAmount) 也得到相同结果。

同类聚合还有 min(invoiceAmount)、max(invoiceAmount)，本次最小、最大单张金额分别为 0.99、23.86。

从分组均额计算整体均额时，要考虑各组账单数。例如，先算出各国平均金额，再把这 20 个均额直接平均，会得到约 5.84；各国账单数不同，整体均额仍应按总金额除总账单数计算。

没有账单时怎么办

nullif(count(invoiceId), 0) 将零分母转为空值。本次把日期改到没有记录的 1990 年，实际返回账单数 0，金额与均额字段未返回。应解释为“该期间没有账单，无法计算平均金额”，不能将均额补成 0。

常用指标，固定下来

临时计算适合一次性需求。如果平均账单金额经常要查，就可以固化到语义层模型。本例 TM 已有金额和账单 ID，只需在 QM 中新增计算指标 averageAmount。

第一步：修改并加载语义层模型

发送：

> 把平均账单金额保存为固定指标，校验并加载。先不查询。

Codex 读取建模 Skill，在账单 QM 的“账单指标”分组中增加 averageAmount：

```
{
    name: 'averageAmount',
    caption: '平均账单金额',
    formula: 'sum(invoiceAmount) / nullif(count(invoiceId), 0)',
    type: 'MONEY',
```

```
    description: '当前筛选与分组范围的账单总金额除以账单数；无账单时为空，不累计或简单平均分组均额。样例原始单位，币种未指定。'
}
```

公式没有变，变化的是维护位置：从每次查询的 calculatedFields 移到 QM。完整源码见账单 QM，数据库没有新增物理列。

Codex 校验模型、加载刷新，再实时获取查询说明。本次四个建模文件校验通过，返回的元数据确认 averageAmount 是预定义计算字段，可直接引用。查询说明由引擎根据已加载的代码自动生成，不需要提前维护 Markdown。

第二步：使用固定指标查询

确认加载成功后，再发送：

> 用固定指标查询 2012 年的平均账单金额。

此时 DSL 不再携带公式：

```
{
  "columns": ["averageAmount"],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "limit": 2
}
```

返回均额仍为 5.75。按国家分组的 20 个均额、无账单期间的空值，也都与临时计算一致。预定义公式已经包含聚合，直接选择 averageAmount 即可，不要再对它求和或平均，也不要在 calculatedFields 中重定义同名字段。

LLM 仍需判断日期和分组范围，但计算公式已由语义层模型统一定义。这样可以减少 LLM 的推导过程，降低公式写错或遗漏空值处理的概率。

金额占比：分母跟着什么范围走

再发送：

> 按账单国家统计 2012 年的金额，以及占全年账单总额的比例。

先按国家求和，再计算各国金额占全部分组金额的比例。本次使用 calculatedFields 中的 ratio_to_total：

```
{
  "columns": ["billingCountry", "sum(invoiceAmount) as totalAmount", "amountShare"],
  "calculatedFields": [
    {"name": "amountShare", "expression": "ratio_to_total(totalAmount)"}
  ],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "groupBy": ["billingCountry"],
  "orderBy": ["-totalAmount", "billingCountry"],
  "limit": 100
}
```

ratio_to_total 使用聚合后的金额计算占比。这次只限定 2012 年，分母为 477.53。实际返回 20 个有账单的国家，下面节选前三项：

账单国家	账单金额	占比
USA	127.98	26.80%
Brazil	53.46	11.20%
Canada	42.57	8.91%

引擎返回的是比例，例如美国约为 0.268004，展示成百分数时乘以 100。完整占比合计为 1；展示值经过四舍五入，合计可能略有差异。

筛选国家与只展示前三名，含义不同

如果在 slice 中先限定美国、加拿大，分母会变为两国金额合计 170.55，对应美国 75.04%、加拿大 24.96%。回答时就应说“占美加两国合计的比例”。

如果只把原请求的 limit 改为 3，分母仍是全年 477.53，前三项占比与上表一致。这三项只是展示了部分国家，不应将它们重新归一化成 100%。两种情况都已实际执行核对。

只累计满足条件的账单

发送：

> 统计 2012 年单张金额至少 10 的账单数和金额。

本例用 if 表达条件，再交给聚合函数累计。保留全年日期范围，将返回列与计算字段改为：

```
{
  "columns": ["largeBillCount", "largeBillAmount"],
  "calculatedFields": [
    {"name": "largeBillCount", "expression": "sum(if(invoiceAmount >= 10, 1, 0))"},
    {"name": "largeBillAmount", "expression": "sum(if(invoiceAmount >= 10, invoiceAmount, 0))"}
  ]
}
```

这是请求片段，完整请求仍带日期 slice，limit 为 2。实际得到 15 张、216.08。条件命中时累计 1 或金额，否则累计 0；这不是先汇总再筛选的 having。

只查这一类账单，用 slice 限定金额后再求和也可以。条件聚合的价值是能在同一范围同时计算不同条件的指标；常用条件应像前面的均额一样固化，并明确门槛。这里的 10 只是练习约定，不是业务上统一的“大额”标准。

完成后检查什么

本次 11 个查询用例均通过校验、执行与原始数据的独立核对，覆盖计数、临时与固定均额、原生 AVG/MIN/MAX、国家均额、占比分母和空期间。模型仅在账单 QM 中新增了 averageAmount，样例数据未修改。

读者重点检查：**数的是账单还是客户，平均值的分母是什么，占比基于哪个数据范围，空值能不能解释成零。**

完整请求、结果和复跑方式见第十章实操记录。

新增条件聚合也已通过 CLI 校验、执行与原始金额独立核对，见基础功能补充记录。

本章 Foggy 知识点

- **聚合函数：** sum、count、count_distinct、avg、min、max，区分账单数与去重客户数。
- **计算指标：** 用 calculatedFields 临时计算，通过 nullif 处理零分母；常用公式可固化到 QM，校验加载后直接引用。
- **分组占比：** ratio_to_total 计算分组金额占比；slice 改变参与统计的范围，limit 只限制展示条数。
- **条件聚合：** sum(if(...)) 累计满足条件的数量或金额，区分条件聚合与聚合后筛选。

下一章继续按日期与月份汇总。

第 11 章 按日期与月份汇总数据

上一章讲了聚合与计算指标，本章给账单增加年月字段，再按月汇总。仍使用 Chinook 和样例原始金额单位；以下示例沿用 Launcher 0.1.25、CLI 0.1.24 的实操记录。

第一步：准备时间分析的语义层模型

在原来的练习目录和会话中发送：

> 为账单建立按年、按月分析的语义层模型，校验并加载。先不查询。

原账单模型只公开了原始日期，本章新增一对专门用于时间分析的建模文件：

文件	定义内容
ChinookInvoiceTimeModel.tm	一行一张账单，定义日期维度、年月字段、金额与张数
ChinookInvoiceTimeQueryModel.qm	公开本次时间分析所需的字段

Foggy 支持直接从业务日期派生年月，**不需要独立的日期维表**。本例由 Codex 把 InvoiceDate 配置为 DATETIME 自维度，Invoice 表保持原样。数据量大、时间分析频繁时，建议考虑独立日期维表，预存年月等属性，减少查询时重复计算；实际性能收益仍取决于数据库、索引和关联计划。

下面是**本章新增的年月属性**，位于 lab/models/model/ChinookInvoiceTimeModel.tm 的 dimensions → invoiceDate → properties 中，其余配置省略：

```
export const model = {
    name: 'ChinookInvoiceTimeModel',
    // 其他模型配置省略
    dimensions: [
        {
            name: 'invoiceDate', caption: '账单日期',
            foreignKey: 'InvoiceDate', primaryKey: 'InvoiceDate', captionColumn: 'InvoiceDate',
            type: 'DATETIME', timeRole: 'business_date',
            properties: [
                // year、month 属性省略
                {
                    column: 'InvoiceDate', name: 'yearMonth', caption: '账单年月', type: 'STRING',
                    description: 'YYYY-MM，自然月分组键，保留年份。',
                    dialectFormulaDef: {
                        sqlite: { builder: (alias) => `strftime('%Y-%m', ${alias}.InvoiceDate)` }
                    }
                }
            ]
        }
    ]
};
```

日期表达式由建模人员维护，LLM 查询时直接选用语义字段；更换数据库时需调整相应方言。

Codex 校验、加载并刷新后，实时获取引擎根据代码生成的查询说明，无须提前维护 Markdown。公开的时间字段包括：

字段	含义与用途
invoiceDate$id	原始账单日期，用于日期范围筛选，不是账单编号
invoiceDate$year	年份，例如 2012
invoiceDate$month	月份序号 1-12
invoiceDate$yearMonth	年月，例如 2012-11，用于自然月分组

后面的 CLI 查询均指定 ChinookInvoiceTimeQueryModel，字段名以实时元数据为准。给日期字段随手加个 $month，引擎可不会自动领会。

第二步：按月统计

确认模型可用后，再发送：

> 按月统计 2012 年的账单金额和张数。

Codex 使用年月分组，按年月升序查询：

```
{
  "columns": ["invoiceDate$yearMonth", "sum(invoiceAmount) as totalAmount", "count(invoiceId) as billCount"],
  "slice": [
    {"field": "invoiceDate$id", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "groupBy": ["invoiceDate$yearMonth"],
  "orderBy": ["invoiceDate$yearMonth"],
  "limit": 100
}
```

日期范围决定“统计哪些账单”，年月字段决定“怎样分组”。原始日期放进 groupBy，不会自动变成月度汇总。

请求先校验，再执行。本次完整返回十二个月，下面节选四项：

年月	账单金额	账单数
2012-09	46.71	6
2012-10	42.62	7
2012-11	37.62	7
2012-12	37.62	7

十二个月合计 **477.53、83 张账单**，与第七章基线和上一章全年统计一致。

跨年分析要保留年份。单独按 month 分组，会把不同年份的一月合在一起；使用 yearMonth，或者同时按 year、month 分组，才能区分。

完成后检查什么

重点检查：**业务日期是否选对，筛选范围是否明确，月份是否保留年份，分月结果能否对回全年。** 月份没有记录，不代表可以直接补零；先确认它表示没有交易，还是缺少数据。

本章保留日期建模与月度汇总。同比、环比、累计、滚动及 LAG 集中到高级篇的“时间分析”大章，届时再讲期间比较和窗口口径。

已有模型、查询和实操记录保留，见0.1.25 实操记录。其中还包含已迁出的期间比较与 LAG 验证，不能把原来的全部用例数视为当前正文覆盖范围；迁出素材留作后续备课。本轮仅调整编排，未重跑查询。

本章 Foggy 知识点

- **日期自维度：** 用主表日期建立 DATETIME 自维度，无须为本例新增日期表。
- **派生属性：** 在 TM 中定义年、月、年月，由 QM 公开，查询字段以实时元数据为准。
- **范围与分组：** 原始日期限定统计范围，年月决定分组；跨年分析保留年份，汇总结果与全年基线核对。

下一章从日期转向业务对象，讲维度成员、多级关联与逐层查询。

第 12 章 从艺术家查到专辑：维度成员与多级关联

上一章按日期与月份汇总金额，这一章看看哪些艺术家卖得多，又卖了哪些专辑、曲目。沿用第八章的明细模型，增加两级关联，金额仍按交易明细累计。

第一步：扩展专辑与艺术家维度

在原来的练习目录和会话中发送：

> 给明细分析增加专辑和艺术家，校验并加载。先不查询。

Codex 先通过 CLI 查看表结构，确认关系：

```
InvoiceLine.TrackId → Track.TrackId
Track.AlbumId       → Album.AlbumId
Album.ArtistId      → Artist.ArtistId
```

每条明细通过曲目找到专辑，再找到专辑所属艺术家。这里不使用 Track.Composer 作曲者字段；金额仍按明细的 UnitPrice × Quantity 计算。

修改已有的两个文件：

文件	修改位置
ChinookInvoiceLineModel.tm	dimensions → track → dimensions 内，与 genre 并列增加 album，再在其下增加 artist
ChinookInvoiceLineQueryModel.qm	columnGroups 中公开专辑、艺术家维度

下面从 TM 顶层 dimensions 展开：track 是已有维度，album、artist 是本章新增的两层；其他配置省略。

```
export const model = {
    name: 'ChinookInvoiceLineModel',
    tableName: 'InvoiceLine',
    // 其他模型配置省略
    dimensions: [
        // 已有的 invoice 维度省略
        {
            name: 'track', caption: '曲目或媒体', tableName: 'Track',
            foreignKey: 'TrackId', primaryKey: 'TrackId', captionColumn: 'Name',
            // track 的属性省略
            dimensions: [
                // 已有的 genre 维度省略，与 album 并列
                {
                    name: 'album', caption: '专辑', tableName: 'Album',
                    foreignKey: 'AlbumId', primaryKey: 'AlbumId', captionColumn: 'Title',
                    dimensions: [
                        {
                            name: 'artist', caption: '艺术家', tableName: 'Artist',
                            foreignKey: 'ArtistId', primaryKey: 'ArtistId', captionColumn: 'Name'
                        }
                    ]
                }
            ]
```

```
        }
    ]
};
```

每层的 foreignKey 都取自直接上一级的表：track 用 InvoiceLine.TrackId，album 用 Track.AlbumId，artist 用 Album.ArtistId；primaryKey 则是当前维表的关联键。

QM 公开专辑、艺术家的 ID 和名称。ref 绑定来源，name 用于查询，alias 决定结果键；本例让后两者相同：

```
{ caption: '专辑与艺术家', items: [
    { ref: line.track.album$id, name: 'albumId', alias: 'albumId' },
    { ref: line.track.album$caption, name: 'albumName', alias: 'albumName' },
    { ref: line.track.album.artist$id, name: 'albumArtistId', alias: 'albumArtistId' },
    { ref: line.track.album.artist$caption, name: 'albumArtistName', alias: 'albumArtistName' }
] }
```

六个模型文件校验通过、刷新后，Codex 实时读取引擎生成的查询说明。Markdown 无须提前生成或维护，新增查询名如下：

字段	用途
albumArtistId、albumArtistName	艺术家 ID、名称
albumId、albumName	专辑 ID、名称

这些名称配置在具体列上；整维度 ref 的 alias 不会重命名所有展开字段。旧的 artist$id、artist$caption 及完整路径名仍兼容，**后文统一使用上表中的查询名**。名称用于展示，分组和筛选保留 ID，避免合并同名成员。

第二步：找出当年有成交的艺术家

“维度成员”就是该维度的具体对象，例如艺术家 ID 1 对应 AC/DC。查找可选对象时，Codex 可以调用 members list ChinookInvoiceLineQueryModel albumArtistId，本例的艺术家目录总数是 **275 位**。

要看某年有哪些艺术家有成交，还需要按日期查询交易记录。

模型加载成功后，再发送：

> 2012 年哪些艺术家有成交？

Codex 查询 ChinookInvoiceLineQueryModel，按艺术家分组并统计明细数：

```
{
  "columns": ["albumArtistId", "albumArtistName", "count(invoiceLineId) as lineCount"],
  "slice": [
    {"field": "invoice$invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "groupBy": ["albumArtistId", "albumArtistName"],
  "orderBy": ["albumArtistId"],
  "limit": 500,
  "returnTotal": true
}
```

先校验再执行，得到 **119 位当年有成交的艺术家**，其中 AC/DC 有 3 条成交明细。它们是目录中的一部分，不能把目录总数当作成交人数。

第三步：按艺术家排名，分清一页和全部

继续发送：

> 按 2012 年销售金额给艺术家排名，每页看 5 位。

Codex 查询 ChinookInvoiceLineQueryModel，仍使用账单日期筛选：

```
{
  "columns": [
    "albumArtistId", "albumArtistName",
    "sum(lineAmount) as totalAmount", "sum(salesQuantity) as quantity"
  ],
  "slice": [
    {"field": "invoice$invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "groupBy": ["albumArtistId", "albumArtistName"],
  "orderBy": ["-totalAmount", "albumArtistId"],
  "start": 0,
  "limit": 5,
  "returnTotal": true
}
```

请求只表达日期、分组、指标和排序，曲目到艺术家的关联由引擎按 TM 展开。金额降序、ID 升序，用来固定同金额时的次序。

艺术家 ID	名称	金额	数量
90	Iron Maiden	33.66	34
150	U2	27.72	28
156	The Office	25.87	13
50	Metallica	25.74	26
22	Led Zeppelin	23.76	24

Chinook 也包含影视内容，所以榜单中会出现 The Office 等条目。金额仅在展示时保留两位小数，数量累计的是 Quantity。

这里有三个不同的数：

- **本页 5 条**：这次返回了多少个艺术家分组。
- total: 119：同一筛选范围内共有多少个分组，不是账单数，也不是金额。
- **全年 477.53**：另行核对的全部成交金额，不能把本页五位的合计 136.75 当作全年金额。

再发送：

> 看下一页。

Codex 保留日期、分组和排序，把 start 改为 5。第二页从 Lost 开始，total 仍为 119。limit 限制展示条数，统计范围保持不变。

第四步：从实际结果继续查专辑和曲目

先发送：

> 看看排名第一的艺术家当年卖了哪些专辑。

Codex 从完整排名取到第一位 Iron Maiden 的 **ID 90**，沿用 2012 年日期范围，增加筛选：

```
{"field": "albumArtistId", "op": "=", "value": 90}
```

分组改为 albumId、albumName，指标仍是明细金额与数量，按金额降序、专辑 ID 升序。本次得到 **19 张当年有成交的专辑**，金额合计 33.66，与艺术家汇总一致。

排在第一的是 Powerslave，ID 为 107，金额 4.95、数量 5。

再发送：

> 再看第一张专辑的成交曲目。

Codex 保留艺术家和日期条件，增加 albumId = 107，按 trackId、track$title 分组。得到 Aces High、2 Minutes To Midnight 等 **5 首有成交的曲目**，金额合计 4.95，与专辑汇总一致。

这就是“下钻”：保留期间和已选对象，把分组从艺术家换成专辑，再换成曲目。

去重列表、合计与小计

前面用成员接口查看艺术家目录。若只想返回交易中出现过的不同取值，可以用 distinct。以账单国家为例，发送：

> 列出 2012 年有账单的国家，不重复。

Codex 改查 ChinookInvoiceQueryModel：

```
{
  "columns": ["billingCountry"],
  "slice": [{"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}],
  "distinct": true,
  "orderBy": ["billingCountry"],
  "limit": 100,
  "returnTotal": true
}
```

实际返回 20 个国家。distinct 对选中列的组合去重，返回不同的值；第十章的 count_distinct 返回数量。若再选账单 ID，组合就变了，不能指望仍只有 20 行。去重也不能补救错误关联造成的金额重复。

让结果附带总计

发送：

> 按国家汇总 2012 年账单金额和张数，显示全部国家并附总计。

仍查询账单 QM，保留日期，将请求改为：

```
{
  "columns": ["billingCountry", "sum(invoiceAmount) as totalAmount", "count(invoiceId) as billCount"],
  "groupBy": ["billingCountry"],
  "orderBy": ["billingCountry"],
  "limit": 100,
  "returnTotal": true,
  "withSubtotals": true
}
```

这是保留日期 slice 后的变更片段，不再使用 distinct。本次返回 20 行国家数据和 1 行总计，按 _rowType 区分：普通行为 data，总计行为 grandTotal。总计 **477.53、83 张**；total: 20 仍是国家分组数，不包含总计行，也不是金额。

总计范围要特别核对。 本版本的 withSubtotals 根据返回的分组生成汇总；同一请求把 limit 改成 5，总计便是这 5 国的 **137.61、25 张**，total 仍为 20。需要全年金额时，应像前文一样另查全范围总额，或确认已完整返回全部分组，不能仅凭 grandTotal 这个名字判断范围。

再按 billingCountry、customerId 两级分组，本次返回 47 行客户数据、20 行国家 subtotal 和 1 行 grandTotal。国家小计均与独立国家汇总一致。这里只讲可加的金额和张数；平均值、去重人数和比例不能直接相加。

完成后检查什么

新增关联后，完整艺术家汇总仍是 **477.53、数量 447**，与原明细汇总一致；专辑合计能对回艺术家，曲目合计能对回专辑。旧分类查询也通过回归，样例数据未修改。

读者重点检查：**成员 ID 是否来自实际结果，期间和上层筛选是否保留，关联是否重复累计或丢失记录，返回的是一页还是全部。** 本样例关联完整，尚未构造缺失专辑、孤立艺术家等异常数据测试。

本章基于 Launcher 0.1.30、CLI 0.1.25 实测：11 项查询先校验、再执行，结果与原始数据基线一致；成员接口的总数与返回页也已独立核对。请求、元数据和复跑方式见实操记录。

本轮另验证去重、普通分组、完整总计、分页总计与两级小计，共 5 项，见基础功能补充记录。

本章 Foggy 知识点

- **多级维度与字段：** TM 嵌套声明 Track → Album → Artist，QM 用具体列的 name/alias 公开查询名与结果键。
- **成员与标识：** 按公开 ID、名称发现成员，用 ID 筛选，区分成交范围与完整目录。
- **逐层分析：** 保留日期与上层对象，从艺术家查到专辑、曲目，金额逐层核对。

- **排序与分页：** 用 orderBy、start、limit 和 returnTotal，区分页内条数、分组总数与全范围金额。
- **去重与汇总行：** distinct 返回不同组合；withSubtotals 追加小计/总计，按 _rowType 识别并核对实际范围。

下一章继续讲查询范围与权限。

第 13 章 让 AI 只查询用户可见的数据

上一章能从艺术家查到专辑和曲目，接下来要问：这些数据，当前用户都能看吗？

本章仍用 Chinook，做三个本地演示身份：美国、加拿大、未授权。前两个分别只看账单国家为 USA、Canada 的交易，都不能直接查询交易单价；未授权身份不能使用这个查询模型。

第一步：准备受限模型与演示身份

在原练习会话中发送：

> 为 Chinook 建一个权限演示模型：美国身份只看美国，加拿大身份只看加拿大，并隐藏交易单价。先不查询。

Codex 读取建模 Skill，准备隔离 Runtime 与本地权限服务，再编写模型。本例另用 Namespace chinook-scope，原教学模型保持不变：

文件	作用
ChinookScopedInvoiceLineModel.tm	沿用明细粒度、关联和金额定义
ChinookScopedInvoiceLineQueryModel.qm	增加模型、字段与行权限

本地权限服务只用于教学，按演示身份返回是否允许访问及账单国家。身份值由配套脚本临时生成，通过 CLI 的 FOGGY_RUNTIME_AUTHORIZATION 传递，不写进模型或查询 JSON。

这里让 Codex 切换身份是为了验证效果。真实应用应从登录会话取得可信身份，用户在聊天中说“我是管理员”不构成授权。业务身份使用 Authorization，与管理 Runtime 的 --auth-code 分开。

谁能查、能查什么，由 QM 落实

下面节选 QM 内的权限配置，line 是已加载的明细 TM：

```
modelPermissions: {
    mode: 'resolver',
    resolver: (context) => {
        const decision = post({
            url: 'http://127.0.0.1:18070/permissions',
            headers: { Authorization: context.authorization },
            body: {
                namespace: context.namespace,
                model: context.model,
                action: context.action
            },
            responseType: 'map'
        });
        if (decision.allowed !== true) {
            return { allow: false };
        }
        return {
            allow: true,
```

```
            attributes: { country: decision.country },
            rowPredicates: [
                context.predicate.eq(line.invoice$billingCountry, decision.country)
            ],
            policyVersion: 'chapter-13-demo-v1'
        };
    }
},
fieldPermissions: {
    defaultVisible: true,
    hiddenFields: [{ fields: ['detailPrice'] }]
}
```

modelPermissions 决定当前操作是否允许；rowPredicates 把允许的账单国家变成查询约束；fieldPermissions 隐藏 detailPrice。这个字段来自 line.unitPrice，在 QM 中同时配置了 name: 'detailPrice' 和 alias: 'detailPrice'。

两个模型文件校验通过、加载后，Codex 按不同身份实时读取元数据：

- 美国、加拿大身份能发现并描述模型，金额等允许字段仍可用，单价不出现在公开字段中。
- 未授权或未带身份时，模型不出现在列表里；直接指定模型名读取说明，也返回 MODEL_ACCESS_DENIED。
- 请求 detailPrice 在查询校验时被拒绝；换回旧名 unitPrice 也不能查出单价。

字段隐藏同时约束元数据和查询入口，不能只把页面上的一列藏起来。

第二步：同一个问题，换两个身份

确认模型加载成功后，再发送：

> 分别用美国和加拿大身份，查询 2012 年的销售金额。

Codex 对两个身份提交同一份 DSL，只通过调用环境切换身份：

```
{
  "columns": ["sum(lineAmount) as amount", "count(invoiceLineId) as lines"],
  "slice": [
    {"field": "invoice$invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "limit": 2
}
```

请求里没有国家条件，结果却不同：

调用身份	允许的账单国家	金额	明细条数
美国	USA	127.98	102
加拿大	Canada	42.57	43

日期来自本次问题，国家范围来自权限决策，两者共同限制查询。LLM 不必每次记得补上国家条件，漏写也不会把范围扩大为全部国家。

这两个金额与原始表按对应账单国家独立核算一致。再从美国切到加拿大、切回美国，结果仍分别对应各自范围。

第三步：试着越过范围

再发送：

> 用美国身份看看加拿大 2012 年的客户排名。

Codex 在客户排名请求里增加 invoice$billingCountry = 'Canada'，但美国身份的权限条件仍是 USA。两者取交集，实际返回空列表；这不能解释为加拿大没有成交数据。

允许范围内的客户排名也已核对：美国 13 位、加拿大 4 位；每页看 3 位时，翻页与分组总数都保持各自范围。

未授权身份的查询会在校验阶段被拒绝；本次没有继续执行这些请求。演示权限服务临时返回错误时，查询校验同样失败，不会自动降级为查询全部数据，也不能换用原来的公开模型继续尝试。

成员列表也要遵守权限

上一章的成员接口同样携带当前身份。本例增加了指向账单维度 ID 的 billId，美国身份查到的账单成员都属于美国，加拿大身份同理。目录本身不继承刚才查询中的 2012 年筛选。

但不能把任何行条件都直接搬到任意维度上。账单国家存在于 Invoice 表，可以用于限制账单目录；同一个条件无法直接映射到 Artist 表。本次用受限身份查询艺术家目录时，引擎返回 MODEL_PERMISSION_RESOLUTION_FAILED，没有退回整个艺术家目录。

所以，成员接口拒绝不一定表示“没有成员”。还要看当前权限能否在这个目录上正确执行；无法保证时，应明确拒绝。

完成后检查什么

重点核对四件事：**模型不能靠猜名字绕过，隐藏字段不能换个名字查出，聚合与分页不扩大行范围，成员接口不丢掉权限条件。** 模型元数据和查询应使用同一个调用身份。

本章用 Launcher 0.1.30、CLI 0.1.25 实操：11 项查询执行并核对通过，5 项查询按预期在校验时拒绝，另验证了模型发现、描述及成员范围。原数据和前面章节的模型未修改，身份值未保存在证据中。复跑、源码与实际诊断见实操记录。

本例验证的是 QM 数据面。发布模型、原始 SQL 探查等管理能力不能作为普通用户问数入口；真实身份验证和管理接口隔离由业务平台负责。

本章 Foggy 知识点

- **模型权限：** modelPermissions 按当前身份和操作解析是否允许访问。
- **字段权限：** fieldPermissions 同时影响公开字段与查询校验。

- **行权限：** typed rowPredicates 与业务筛选共同生效，约束聚合、排名和分页。
- **成员范围：** 复用来源模型的权限；无法安全映射的条件明确拒绝。

下一章讲如何在同一个 QM 中关联多个 TM，以及为什么常常需要先聚合再 JOIN。模型维护与回归放在其后。

第 14 章 在 QM 中关联多个 TM：普通 JOIN 与聚合 JOIN

前面的 TM 已经映射了事实与维度，也能描述曲目、专辑、艺术家这样的雪花式关联。但现实分析还经常需要关联多个事实，例如订单与支付、账单与明细。它们各有粒度，关联前要先确定：按什么键连接，是否需要先汇总。

本章把账单 TM 与明细 TM 的关联定义在**同一个 QM** 中。准备好模型后，AI 仍然提交普通 JSON DSL，由引擎按 QM 展开 JOIN；需要先聚合的规则也写在 QM 中。

先确定结果粒度，再选主 TM

主 TM 是查询起点，先看最终希望“一行代表什么”：

查询目标	主 TM 与常用关联方式
每张账单及其明细合计	以 ChinookInvoiceModel 为主，明细先按账单 ID 聚合再 JOIN，保持一账单一行
每条明细及所属账单的日期、客户等属性	以 ChinookInvoiceLineModel 为主，按账单唯一键做多对一关联；可在 TM 中定义账单维度，也可在 QM 中 JOIN

前面明细 TM 的 invoice 维度就是第二种做法。若要展开明细，直接 JOIN 也合理；但主 TM 不保证结果保持原粒度，取到的账单总额也不能当作明细金额累计。

本章先用普通 JOIN 看清行数变化，再改成聚合 JOIN，保持账单粒度。

第一步：创建普通关联 QM，先不查询

沿用已有的 ChinookInvoiceModel 和 ChinookInvoiceLineModel，在原练习目录和会话中发送：

> 以账单为主，建一个关联明细的 QM，校验并加载。先不查询。

Codex 读取建模 Skill，检查两侧字段，在 ChinookInvoiceLineJoinQueryModel.qm 中定义：

```
const invoice = loadTableModel('ChinookInvoiceModel');
const line = loadTableModel('ChinookInvoiceLineModel');

export const queryModel = {
    name: 'ChinookInvoiceLineJoinQueryModel',
    caption: '账单与明细直接关联',
    description: '在一个 QM 中关联账单和明细 TM；有明细时一行一条明细，账单金额会重复，不能直接累计。',
    loader: 'v2',
    model: invoice,
    joins: [invoice.leftJoin(line).on(invoice.invoiceId, line.invoiceId)],
    columnGroups: [{ caption: '账单与明细', items: [
        { ref: invoice.invoiceId },
        { ref: invoice.invoiceDate },
        { ref: invoice.customerId },
```

```
        { ref: invoice.invoiceAmount },
        { ref: line.invoiceLineId },
        { ref: line.lineAmount },
        { ref: line.salesQuantity }
    ] }],
    accesses: []
};
```

model 指定主 TM；leftJoin 保留左侧账单，on(...) 指定两侧账单 ID。columnGroups 选择这个 QM 对外公开的字段。

加载后，Codex 实时获取引擎根据代码生成的字段说明，确认账单与明细字段。Markdown 不需要提前维护。

第二步：查账单及明细，看清关联后的粒度

确认 QM 可用后，再发送：

> 查 2012 年的账单及明细，保留账单金额和明细金额。

Codex 指定 ChinookInvoiceLineJoinQueryModel，提交普通 DSL：

```
{
  "columns": ["invoiceId", "customerId", "invoiceAmount", "invoiceLineId", "lineAmount", "salesQuantity"],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "orderBy": ["invoiceId", "invoiceLineId"],
  "limit": 1000
}
```

请求先校验、再执行，完整返回 **447 条明细**。账单 250 的金额为 **13.86**，包含 14 条明细，每条明细行上都会带着这张账单的 13.86。

关联后是一明细一行：累计明细金额为 **477.53**，直接累计重复出现的账单金额却得到 **4256.25**。第二章的重复累计，在语义模型中也一样需要处理。

第三步：先聚合明细，再定义关联

要保持账单粒度，一种常见方案是通过 ETL，将 InvoiceLine 按账单 ID 加工成汇总表，再与 Invoice 关联。小数据量下，额外维护汇总表、刷新任务和变更逻辑，成本未必划算。

聚合 JOIN 是一种折中：把汇总规则定义在 QM 中，查询时先聚合明细再关联。 它省去汇总表的维护，但仍有查询时聚合的开销；数据量或查询频率增加后，再评估预汇总方案。

按这个方案，让 Codex 新建 QM：

> 把明细先按账单汇总，再关联账单，另建一个用于对账的 QM。先不查询。

Codex 新建 ChinookInvoiceLineAggregateJoinQueryModel.qm：

```
const invoice = loadTableModel('ChinookInvoiceModel');
const line = loadTableModel('ChinookInvoiceLineModel');

const detail = line
```

```
    .groupBy(line.invoiceId)
    .sum(line.lineAmount, 'detailAmount')
    .sum(line.salesQuantity, 'detailQuantity')
    .count('detailCount')
    .as('detailByInvoice');

export const queryModel = {
    name: 'ChinookInvoiceLineAggregateJoinQueryModel',
    caption: '账单与明细聚合关联',
    description: '明细先按账单 ID 汇总，再在当前 QM 中关联账单 TM，一行一张账单；按账单日期筛选，金额沿用样例原始单位
  。',
    loader: 'v2',
    model: invoice,
    joins: [invoice.leftJoin(detail).on(invoice.invoiceId, detail.invoiceId)],
    columnGroups: [{ caption: '账单对账', items: [
        { ref: invoice.invoiceId },
        { ref: invoice.invoiceDate },
        { ref: invoice.customerId },
        { ref: invoice.invoiceAmount },
        { ref: detail.detailAmount, name: 'detailAmount', alias: 'detailAmount' },
        { ref: detail.detailQuantity, name: 'detailQuantity', alias: 'detailQuantity' },
        { ref: detail.detailCount, name: 'detailCount', alias: 'detailCount' }
    ] }],
    accesses: []
};
```

这里的 detail 是在 QM 中定义的聚合关系：

- groupBy(line.invoiceId)：按账单 ID 收敛到一账单一行。
- sum(...)、count(...)：定义明细金额、数量和条数，结果名分别为 detailAmount、detailQuantity、detailCount。
- as('detailByInvoice')：给这份聚合关系命名，随后通过 invoice.leftJoin(detail) 关联。

两个 TM 保持原样，聚合关系随新 QM 加载，由引擎在查询时展开。

这里的 groupBy 发生在 JOIN 之前。 查询 DSL 的 groupBy 则作用于关联结果；若账单金额已经随明细重复，外层分组求和仍会重复累计。

第四步：使用新 QM 查询与核对

新 QM 校验、加载后，再发送：

> 用对账 QM 查询 2012 年每张账单的金额、明细合计和条数。

Codex 实时获取新 QM 的元数据，确认聚合字段已经公开，再提交：

```
{
  "columns": ["invoiceId", "customerId", "invoiceAmount", "detailAmount", "detailQuantity", "detailCount"],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "orderBy": ["invoiceId"],
  "limit": 100
}
```

本次返回 **83 张账单**，每张账单金额与明细合计都一致。以账单 250 为例：

账单 ID	账单金额	明细合计	销售数量	明细条数
250	13.86	13.86	14	14

全年核对结果：

核对项	结果
账单金额合计	477.53
明细金额合计	477.53
账单数	83
明细条数 / 销售数量	447 / 447

现在两侧都是一账单一行，可以分别累计账单金额和明细合计。两列用于对账，不能相加当作新的销售额。

如果还要按客户统计，可以在这个 QM 上继续用普通 DSL，按 customerId 分组，分别求和 invoiceAmount 和 detailAmount。本次得到 **47 位客户**，两列金额逐客户一致；完整请求见按客户对账示例。

完成后检查什么

重点检查：**关联键是否正确，右侧聚合键是否唯一，结果行数与两侧金额能否对回原始记录。**

日期仍按账单日期筛选，金额沿用 Chinook 原始单位。若换成订单与支付，订单日期和支付日期往往不同，不能直接照搬本例口径。

正常样例的账单都有明细。另用测试模型将右侧筛空时，83 张左侧账单仍保留，右侧字段未返回；这类空值不能直接解释成金额为零。

本章使用 Launcher 0.1.30、CLI 0.1.25，10 项单 QM 查询结果与只读原始记录、Decimal 独立核对一致；错误字段在查询校验时被拒绝。当前版本的聚合关系仅支持 LEFT JOIN，改用 INNER JOIN 会在模型校验时明确拒绝。原账单、明细与月度查询的三项回归也通过。

完整模型、实时元数据、DSL、解释结果和复跑方式见第十四章实操记录。

本章 Foggy 知识点

- **一个 QM 关联多个 TM：** 用 model 指定主 TM，joins 声明关联，columnGroups 公开字段。
- **聚合关系：** 在 QM 中用 groupBy、sum、count、as 定义右侧，再 JOIN 到主 TM。
- **统一查询面：** AI 选择这个 QM，按实时元数据提交普通 DSL，无须每次重写关联和预聚合规则。
- **粒度核对：** 关联前聚合与关联后汇总是两个阶段，模型能执行不等于金额就能随意累计。

下一章讲完善语义模型后，怎样确认已有问题仍能答对。

第 15 章 用字典连接业务名称与数据取值

第九章查询“美国”时，先确认数据库里存的是 USA。这种稳定的取值与名称对应关系，可以用字典写进语义层模型，减少 AI 每次猜测。

本章给账单国家增加中文字典，不改数据库记录，也不改变账单金额的计算。

第一步：定义字典，先不查询

在原练习目录和会话中发送：

> 给账单国家增加中文名称字典，保留原始值，校验并加载。先不查询。

Codex 根据 Invoice.BillingCountry 的真实取值整理中文名称，交人审核。样例共有 24 种国家取值，全部纳入字典；2012 年有账单的是其中 20 国。

在模型目录新增 dicts.fsscript，以下节选两项，完整内容见字典源码：

```
import { registerDict } from '@jdbcModelDictService';

export const dicts = {
    billingCountry: registerDict({
        id: 'chinook_billing_country', caption: 'Chinook 账单国家',
        items: [
            { value: 'USA', label: '美国' },
            { value: 'Canada', label: '加拿大' }
            // 其他已确认的国家见完整文件
        ]
    })
};
```

value 是数据库中的原始值，label 是对应的业务名称。显示中文不需要修改数据库记录。

在 model/ChinookInvoiceModel.tm 文件开头导入字典，再给已有的 billingCountry 属性增加 dictRef：

```
import { dicts } from '../dicts.fsscript';

export const model = {
    name: 'ChinookInvoiceModel',
    // 其他配置省略
    properties: [
        // 其他属性省略
        { column: 'BillingCountry', name: 'billingCountry', caption: '账单国家', type: 'STRING',
          dictRef: dicts.billingCountry,
          description: '账单地址中的国家，不是客户地址国家。' }
    ]
};
```

账单 QM 已有 { ref: invoice.billingCountry }，保留即可。完整 TM 见账单模型源码。

第二步：确认实时元数据

模型校验、刷新后，Codex 通过 CLI 实时获取 ChinookInvoiceQueryModel 的说明，确认字典已生效。以下是引擎生成的 Markdown 节选：

```
| 字段名 | 名称 | 类型 | 说明 |
|--------|------|------|------|
| billingCountry | 账单国家 | 文本 | 账单地址中的国家，不是客户地址国家。（字典:chinook_billing_country） |
| billingCountry$caption | 账单国家(名称) | 文本 | 字典 label；仅支持等值/集合过滤，不支持排序 |

## 字典定义
| ID | 名称 | 取值 |
|----|------|------|
| chinook_billing_country | Chinook 账单国家 | Argentina=阿根廷, …, USA=美国, United Kingdom=英国 |
```

这份 Markdown 由引擎根据已加载代码自动生成，查询时实时获取，无须提前编写。上面省略了部分字典取值，完整返回见实时 Markdown。

现在，AI 既能看到 USA=美国，也知道可用 billingCountry$caption 取得名称。这里的名称来自字典；第十二章的维度名称则来自关联表。

第三步：查询原值与名称

确认元数据中已有字典和名称字段，再发送：

> 查一下 2012 年美国的账单，同时显示国家原值和中文名称。

Codex 构造请求：

```
{
  "columns": ["invoiceId", "billingCountry", "billingCountry$caption"],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]},
    {"field": "billingCountry", "op": "=", "value": "USA"}
  ],
  "orderBy": ["invoiceId"],
  "limit": 100
}
```

先校验、再执行，返回 21 张账单。例如账单 255 的 billingCountry 为 USA，billingCountry$caption 为“美国”。

中文标签能直接筛选吗

在当前 Launcher 0.1.30 中，这两种写法也已验证，返回相同的 21 张账单：

```
{"field": "billingCountry", "op": "=", "value": "美国"}
```

```
{"field": "billingCountry$caption", "op": "=", "value": "美国"}
```

两种写法最终都转换为参数 USA，转换由引擎依据字典完成。billingCountry 的 in 值写为 ["美国", "加拿大"] 也已验证，返回共 30 张账单。本章主示例使用原始值，方便核对；标签筛选按元数据支持的操作符使用。

字典只转换已收录的名称。例如“美利坚”未收录，用它等值筛选时，当前版本校验、执行成功，但返回空列表。AI 应先核对字典，必要时请用户确认，不能据此回答“美国没有账单”。

完成后检查什么

重点检查：**字典是否对应真实取值，定义是否进入实时元数据，筛选与显示是否符合预期。**

本章 7 项查询均与只读原始记录核对；原有 7 项查询回归结果一致，错误字段仍被拒绝，数据库未修改。

步骤与证据见基础功能补充记录。本次由当前 Codex 会话实际调用 CLI，未另开干净会话验收短提示词。

本章 Foggy 知识点

- **字典定义：** registerDict 维护稳定的 value/label 对应，通过 TM 属性的 dictRef 引用。
- **实时上下文：** 引擎自动提供字典定义与名称字段，减少 LLM 对业务取值的猜测。
- **查询与显示：** 原值用于稳定核对，$caption 显示名称；标签筛选按支持范围转换，未收录名称不能擅自解释。

下一章用模型维护与旧问题回归，为基础篇收尾。

第 16 章 模型维护与自动化回归

前面的查询已经逐项核对过。本章让 Codex 把它们写成自动化回归脚本，以后改完模型，直接复跑，减少临时重写请求和人工核对的遗漏。

第一步：让 Codex 生成回归脚本

在原练习目录和会话中发送：

> 把前面核对过的问题写成 Python 回归脚本，覆盖数值、国家字典和错误请求。先不运行。

Codex 整理查询请求与预期，再编写脚本。本章已有可运行的脚本和题库，主要检查：

范围	自动断言
总额与计数	2012 年金额 477.53、83 张账单、47 位购买客户
平均、月份与关联	均额按账单数计算；月度合计及账单/明细金额能对回全年
原值与中文标签	USA 与“美国”命中同一批 21 张账单，合计 127.98
名称列与集合	原值 USA 对应“美国”；美加集合命中 30 张账单
空结果	未收录名称返回空列表；无账单期间均额不补零
错误请求	拼错字段返回 FIELD_NOT_FOUND，拒绝后不继续执行

预期来自前面独立核对的数据，不是脚本运行后，再让 AI 看着结果判断“应该没问题”。题库里的问题用于说明用例，真正复跑的是固定 DSL 与断言。

生成的脚本做什么

本例使用 Python 标准库调用 Foggy CLI，不需要另外安装测试框架。一轮运行包括：

1. **确认环境。** 获取 capabilities 和实时元数据，保存当前源码、请求及数据摘要。
2. **执行测试。** 每条请求先 query validate，通过后才 query execute；预期拒绝的用例检查错误码。
3. **校验、刷新模型。** 调用 models validate、models refresh，重新获取实时元数据。
4. **复跑并断言。** 使用相同请求，检查两次结果，并与只读原始记录、Decimal 计算独立核对。
5. **保存证据。** 每次创建独立目录，留下响应、命令与验收报告；失败时非零退出。

下面节选脚本中的字典断言，results 是本轮各用例的返回行：

```
# 中文标签筛选必须与原值筛选得到同一批记录
assert results['09-dict-label'] == results['03-usa']

# 返回名称必须与已确认的字典对应
```

```
assert all(
    row['billingCountry$caption'] == '美国'
    for row in results['10-dict-display']
)

# 未收录名称的当前行为也是回归检查的一部分
assert results['13-dict-unknown'] == []
```

完整脚本先检查行数和 ID，避免空列表也通过 `all(...)`；金额比较允许微小浮点误差。

第二步：运行脚本，留下基线

发送：

> 运行回归脚本，保存报告，有失败先定位原因。

Codex 在项目根目录执行：

```
python lab/checks/run-chapter-16-regression.py
```

脚本适用于前面已经准备好的教学环境，不负责安装或启动服务。它会校验、刷新现有模型，不修改模型源码或数据库。实时 Markdown 仍由引擎根据已加载代码生成，脚本只获取和检查。

本轮在 Launcher 0.1.30、CLI 0.1.25 上实际运行。校验刷新前后各 **12 项查询通过、1 项错误字段按预期拒绝**，结果一致，模型资源与数据库摘要未变。均额说明、24 项国家字典与名称字段也都正确。

证据目录见最近一次报告，具体步骤见实操记录。脚本遇到首个失败会停止，按报错和用例编号查看对应响应。最近一次验收报告只在成功时更新，失败时应查看本次输出，避免误读旧报告。

第三步：维护模型后，再跑同一脚本

以 `averageAmount` 为例，如果“平均账单金额”的说明还不明确，可以发送：

> 把平均账单金额的说明写清楚，和人均消费区分开，字段名和公式不变。先不查询。

这次只完善分母说明，公式仍为：

```
sum(invoiceAmount) / nullif(count(invoiceId), 0)
```

修改完成后，再发送：

> 再运行回归脚本，检查旧查询是否仍然正确。

只改说明，原有数值和公开字段应保持兼容；改字典，则重点检查取值转换、名称显示和未收录名称；改公式或统计范围，先确认新的业务定义与预期，再调整断言并记录原因。

当前模型已经具备完善后的均额说明，因此本轮没有再次修改源码。实际完成的是当前模型在校验刷新前后的复验；早先的说明修改过程保留在原维护记录。

注意，本脚本每轮比较的是当前模型刷新前后的结果，不会自动比较两次运行之间的源码变更。维护后的正确性由固定业务断言检查，历史报告用于追溯。

自动化测试覆盖到哪里

这套脚本验证固定请求下的模型与查询行为。“客单价是多少”是否会主动澄清、“利润是多少”是否会如实说明缺少成本，仍是两项未执行的回答审阅题，不能因为脚本通过就算 AI 也通过。

后续在线问数时，再沿用这些预期，单独验收自然语言理解、追问与最终回答。本轮由当前 Codex 会话实际调用 CLI，未另开干净会话测试短提示词。

本章 Foggy 知识点

- **自动化回归：** 把已核对的 DSL、预期结果和错误行为保存为可重复执行的测试。
- **模型维护：** 校验、刷新后实时读取元数据，再检查定义和查询结果。
- **字典与边界：** 同时断言数值、ID、名称转换、空结果与预期拒绝。
- **分层验收：** 脚本验证固定查询，AI 的理解与回答另外验收。

基础篇到这里收尾。留下的除了语义模型，还有一套维护后可以重复运行的测试。

第 17 章 时间分析：比较、累计与滚动窗口

第十一章已经能按月汇总。本章继续问：十一月比十月怎样，今年累计有多少，最近七天又是多少？沿用 Chinook 和 ChinookInvoiceTimeQueryModel，先确认时间口径，再让引擎计算。

本系列使用 Launcher 0.1.38 预发布版、CLI 0.1.25，见环境与复跑。本章沿用已有模型，从查询开始。

先确认时间模型

在原练习目录和会话中发送：

> 检查账单时间模型的日期、年月和金额字段，先不查询。

Codex 实时读取模型说明，确认 invoiceDate$id 是账单日期，invoiceDate$year、invoiceDate$month 用于年月分组，invoiceDate$yearMonth 表示 YYYY-MM，invoiceAmount 默认求和。

日期沿用记录值，不自行转换时区；金额用样例原始单位。Markdown 由引擎根据已加载代码自动生成，查询时实时获取。

一、同比与环比：先认准比较对象

发送：

> 看看 2012 年十月、十一月的账单金额，以及环比变化。

Codex 使用 timeWindow：

```
{
  "columns": [
    "invoiceDate$year",
    "invoiceDate$month",
    "invoiceAmount",
    "invoiceAmount__prior",
    "invoiceAmount__diff",
    "invoiceAmount__ratio"
  ],
  "groupBy": ["invoiceDate$year", "invoiceDate$month"],
  "orderBy": ["invoiceDate$year", "invoiceDate$month"],
  "timeWindow": {
    "field": "invoiceDate$id",
    "grain": "month",
    "comparison": "mom",
    "range": "[)",
    "value": ["2012-10-01", "2012-12-01"],
    "targetMetrics": ["invoiceAmount"]
  },
  "limit": 500
}
```

mom 比较上月；__prior 是上期金额，__diff 是本期减上期，__ratio 是差额除以上期金额。先校验、再执行，得到：

月份	本期金额	上月金额	差额	环比变化
2012-10	42.62	46.71	4.09	8.76%
2012-11	37.62	42.62	5.00	11.73%

value 与 [) 限定当前输出为十月、十一月，引擎还会读取比较期，所以十月可以取得九月金额。额外的原始记录过滤可能截掉上期数据，累计和滚动时也要留意。

再发送：

> 再看这两个月的同比。

同一请求改为 comparison: "yoy"，比较上年同月。十一月上年同期是 23.76，增加 13.86，同比约 58.33%。比例由引擎计算，LLM 说明比较对象，展示百分数时才乘以 100。

跨年环比也已核对：2012 年一月与 2011 年十二月均为 37.62，变化为 0。没有上年记录时，上期、差额和比例为空或省略，不能当成零。若上期确实为零，增长比例需要按业务口径解释；本轮未另造零上期样例测试。

二、累计：展示范围不等于累计起点

比较两期能看增减，累计则回答“从期间起点到现在一共有多少”。

发送：

> 看看 2012 年十月、十一月的账单金额和年内累计。

YTD 从年初累计到当前期间。请求只需选择年、月和指标，累计所需的内部日期依赖由引擎处理：

```
{
  "columns": ["invoiceDate$year", "invoiceDate$month", "invoiceAmount", "invoiceAmount__ytd"],
  "groupBy": ["invoiceDate$year", "invoiceDate$month"],
  "orderBy": ["invoiceDate$year", "invoiceDate$month"],
  "timeWindow": {
    "field": "invoiceDate$id",
    "grain": "month",
    "comparison": "ytd",
    "range": "[)",
    "value": ["2012-10-01", "2012-12-01"],
    "targetMetrics": ["invoiceAmount"]
  },
  "limit": 500
}
```

实际返回两个月，仍保持每月一行：

月份	当月金额	年内累计
2012-10	42.62	402.29
2012-11	37.62	439.91

引擎保留年初以来的计算输入，再按 value 只展示十月、十一月。

月内累计：MTD

发送：

> 看看十一月下半月各交易日的账单金额和月内累计。

改用日粒度和 mtd：

```
{
  "columns": ["invoiceDate$id", "invoiceAmount", "invoiceAmount__mtd"],
  "groupBy": ["invoiceDate$id"],
  "orderBy": ["invoiceDate$id"],
  "timeWindow": {
    "field": "invoiceDate$id",
    "grain": "day",
    "comparison": "mtd",
    "range": "[)",
    "value": ["2012-11-15", "2012-12-01"],
    "targetMetrics": ["invoiceAmount"]
  },
  "limit": 500
}
```

实际返回：

日期	当日金额	月内累计
2012-11-27	3.96	27.72
2012-11-28	3.96	31.68
2012-11-29	5.94	37.62

虽然只展示 15 日以后，累计仍从月初计算。没有交易的日期不会自动补行，所以本次只有三个交易日。

这里要分清两层：timeWindow.value **控制展示范围，**slice **限制可参与计算的原始数据。** 若用 slice 先排除十一月以前的记录，再做 YTD，剩下的数据无法代表全年累计。权限范围同样不能为补历史而放宽。

全年与跨年/跨月累计也已核对：2012 年末累计为 477.53，YTD 按年、MTD 按月重新开始。

三、LAG：取上一行，不一定是上个月

有时只需要取得有序结果的上一行，例如在月度汇总后继续计算，这可以用 LAG。

发送：

> 按月汇总 2012 年金额，再显示上一行金额。

Codex 保留第十一章的月度查询，先产生 sum(invoiceAmount) as totalAmount，再把 previousAmount 加入返回列，并增加：

```
{
  "calculatedFields": [
    {
      "name": "previousAmount",
      "expression": "lag(totalAmount, 1)",
      "windowOrderBy": [{"field": "invoiceDate$yearMonth", "dir": "asc"}]
    }
  ]
}
```

这是月度请求的补充片段。按年月排序后，十一月取得十月的 42.62；一月前面没有结果行，因此为空，不会自动读取去年十二月。

若有意只保留九月、十一月，十一月的上一行便是九月 46.71。这用筛选制造了结果缺口，说明比较自然月与读取上一行是两种需求：前者用环比，后者用 LAG。

lag(sum(invoiceAmount), 1) 的嵌套写法会被校验拒绝，提示先给聚合结果起别名。按诊断调整后重新校验，不继续执行被拒绝的请求。

四、滚动窗口：calendar 与 observed

需要汇总最近一段数据时，可以用滚动窗口。窗口按自然日还是实际数据桶计算，由 rollingMode 区分。

本例先按 day 汇总，每个有账单的日期是一桶；同一天多张账单仍只算一个日期桶。

模式	窗口口径
calendar	包含当天的 N 个自然日；近七天为 D 6 至 D，本章使用 day 粒度
observed	当前桶及之前最多 N 1 个实际数据桶，可能跨越更长的日历范围

查询七个自然日

发送：

> 查十一月下半月各交易日的账单金额，以及含当天的近七个自然日合计。

Codex 显式选择 calendar：

```
{
  "columns": ["invoiceDate$id", "invoiceAmount", "invoiceAmount__rolling_7d"],
  "groupBy": ["invoiceDate$id"],
  "orderBy": ["invoiceDate$id"],
  "timeWindow": {
    "field": "invoiceDate$id",
    "grain": "day",
    "comparison": "rolling_7d",
    "range": "[)",
    "value": ["2012-11-15", "2012-12-01"],
    "targetMetrics": ["invoiceAmount"],
    "rollingMode": "calendar"
  },
  "limit": 500
}
```

本次只返回 27、28、29 日。

改成最近七个交易日期桶

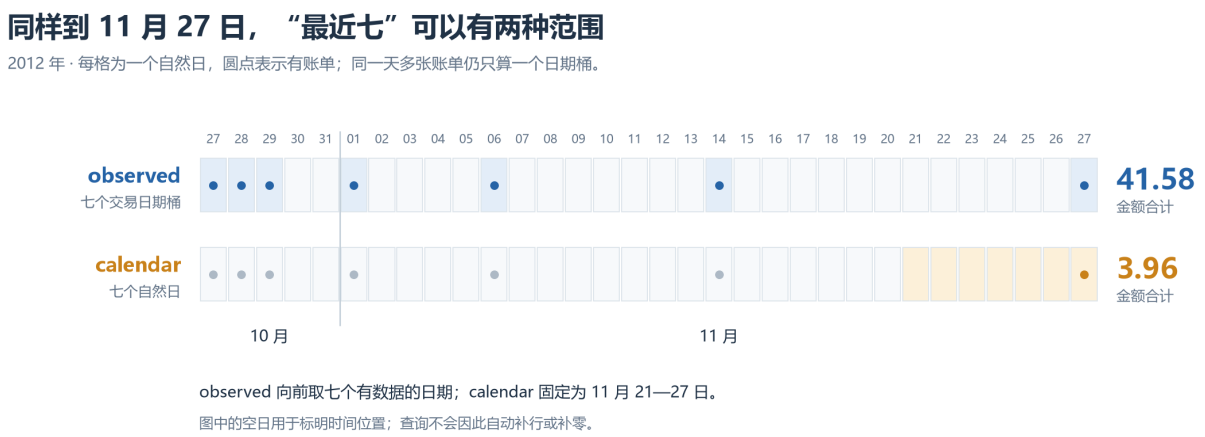
再发送：

> 改成最近七个有账单日期的合计，其他条件不变。

只将 rollingMode 改为 observed，其他字段、范围与数据保持相同：

日期	当日金额	calendar：七个自然日	observed：七个交易日期桶
2012-11-27	3.96	3.96	41.58
2012-11-28	3.96	7.92	41.58
2012-11-29	5.94	13.86	43.56

以 27 日为例，同一截止日的两种窗口如下：



2012年11月27日：calendar取11月21至27日，合计3.96；observed取最近七个有账单的日期桶，合计41.58。日期按自然日等距排列。

observed 取的是交易日期桶，不能称作“近七天”或“最近七张账单”。

省略模式时默认使用 observed，并在 data.warnings 返回 TIMEWINDOW_OBSERVED_DEFAULT。查询显式写出模式，避免把默认值当成业务约定。

近三十个自然日可使用同一结构，将 comparison 和返回后缀改为 rolling_30d，保留 calendar；11 月 27 日合计为 33.66，也已独立核对。

滚动平均和计数，平均的是什么

同一请求可以设置 rollingAggregator: "avg" 或 "count"。11 月 27 日的 calendar 七日窗口只有一个有金额的日期桶，AVG 为 3.96、COUNT 为 1；observed 七桶的 AVG 为 5.94、COUNT 为 7。

这里 AVG 平均的是已经按日汇总的金额，COUNT 数的是非空指标桶，均不是原始账单统计。即使使用 calendar，缺失日期也没有自动补零，不能把 AVG 叫作完整七天的日均。要算完整日均，先按第十八章确认日历和缺失处理。

五、把期间口径放进回归脚本

沿用第十六章的做法，发送：

> 把这些时间分析加入回归脚本，核对日期范围、金额和空值。

除了金额，断言还应检查：比较取的是哪一期，累计有没有保留期间起点，LAG 的首行是否为空，以及 calendar/observed 是否按各自口径计算。预期通过只读原始记录与 Decimal 独立核算。

本章完整 JSON 示例及主要变体已在 0.1.38 教学环境复跑，金额与原始账单基线一致。当前验证保存请求、响应和复跑入口，旧版完整初验与故障记录仍保留在 lab。

本章 Foggy 知识点

- **期间比较：** mom/yoy 与 prior/diff/ratio，分清本期、上期及变化比例。
- **期间累计：** YTD、MTD 的期间重置，区分展示范围与源数据筛选。
- **有序结果：** 先汇总再 LAG，上一行不等于上个自然期间。
- **滚动模式：** calendar 按自然日，observed 按实际数据桶，显式指定口径并检查警告。
- **自动化核对：** 同时断言日期、金额、空值和预期拒绝，不能只看执行成功。

第 18 章 业务日期、日历与可比期间

上一章确定怎样计算，本章补齐计算前的业务条件：按哪个日期统计，没有记录的日期怎么办，两段时间怎样才算可比？

主例是：**截至 2012 年 11 月 15 日，账单金额比上月同期怎样？** 口径先确定，再交给引擎计算。

沿用 Launcher 0.1.38 预发布版、CLI 0.1.25，本章另建日历教学副本。

第一步：准备日历与语义模型，先不查询

在原练习目录发送：

> 用 Chinook 准备本章的独立教学副本和日期、日历模型。报告统计到 2012 年 11 月 15 日，先不查询。

Codex 用样例准备脚本生成独立数据库，再编写模型，放在 lab/chapter-18，接入 Namespace chinook-calendar。原 Chinook 数据和模型保持不变。

样例包含两部分：

数据	来源与约定
账单	复制 Chinook 的账单 ID、日期和金额；额外生成“账单发生后两天”的模拟入库时间，仅用于日期选择对照，**不是付款时间**
日历与完整性	连续自然日；周一至周五标为教学工作日，不含法定节假日规则。另设报表完整性声明：11 月 15 日及以前已确认，之后未确认

实际项目的完整性声明应由数据同步、对账或业务确认提供，不能根据“有没有记录”推断。统计截止日也不等于报告生成日。

本章创建四个文件，完整源码可直接查看：

文件	作用
TeachingInvoiceDatesModel.tm	账单日期、模拟入库时间及金额
TeachingInvoiceDatesQueryModel.qm	公开两种时间与账单指标
TeachingReportCalendarModel.tm	日期、年月、周、工作日序和报表完整性
TeachingCalendarReportQueryModel.qm	日历为主表，关联按日汇总的账单，定义报表金额

两种时间都作为普通日期属性，放在 TeachingInvoiceDatesModel.tm 的 properties 中：

```
{
    column: 'InvoiceDate', name: 'invoiceDate', caption: '账单发生时间', type: 'DATETIME',
    timeRole: 'business_date', recommendedUse: '按账单发生时间统计业务金额'
},
```

```
{
    column: 'LoadedAt', name: 'loadedAt', caption: '模拟入库时间', type: 'DATETIME',
    timeRole: 'system_time',
    recommendedUse: '仅按模拟入库时间统计，不得代替账单发生时间或付款时间'
}
```

timeRole 标明时间角色，recommendedUse 说明用途；字段的完整描述见源码。这里显式声明，避免依赖字段名称推断。

关键关联沿用第十四章的聚合 JOIN：

```
const calendar = loadTableModel('TeachingReportCalendarModel');
const invoice = loadTableModel('TeachingInvoiceDatesModel');
const daily = invoice.groupBy(invoice.invoiceDay)
    .sum(invoice.invoiceAmount, 'dayAmount')
    .count('dayCount').as('invoiceByDay');

export const queryModel = {
    // 其他配置见完整源码
    model: calendar,
    joins: [calendar.leftJoin(daily)
        .on(calendar.calendarDate, daily.invoiceDay)]
};
```

日历每天一行，账单先按日汇总，所以关联后仍是每天一行。反过来从账单出发关联日历，不能凭空补出没有账单的日期。

QM 将日汇总公开为 rawAmount，并在 columnGroups 中固定报表金额的规则：

```
{
    name: 'reportedAmount', alias: 'reportedAmount',
    caption: '已确认账单金额', type: 'MONEY',
    formula: 'if(dataComplete == 1, coalesce(rawAmount, 0), null)'
}
```

dataComplete 为 1 时保留金额，无账单则补零；未确认时保留空值。

Codex 校验四个文件、登记模型并刷新，再实时获取引擎生成的 Markdown，无须提前生成或维护。本次 JSON 与 Markdown 均保留了两种时间的角色和用途，后面的字段名按实际元数据使用。

第二步：先选对日期

先用完整样本中的十一月演示日期差异，暂不套用后面的报告截止。发送：

> 分别按账单日期和模拟入库时间，统计 2012 年十一月的账单金额。

Codex 在 TeachingInvoiceDatesQueryModel 中，分别用 invoiceDate 和 loadedAt 筛选十一月。角色与说明帮助 AI 选字段，实际查询仍要引用正确日期。

统计日期	十一月金额	账单数
账单日期	37.62	7
模拟入库时间	31.68	6

例如账单 325 的金额为 5.94：账单日期是 11 月 29 日，模拟入库时间是 12 月 1 日。两种查询都能正确执行，回答的却是不同问题。

下单、开票、付款同样如此。引擎不会因为它们都是日期就知道该选哪一个；没有付款数据，也不能拿入库时间凑数。

第三步：没有账单，不一定是零

发送：

> 把十一月的账单金额按日列全，区分没有账单和数据尚未确认。

Codex 改查 `TeachingCalendarReportQueryModel`。主要字段为：

字段	含义
calendarDate	日历日期
dataComplete	此日的数据是否已确认完整，1 为是，0 为未确认
rawAmount	账单按日汇总的原始金额；没有匹配记录时为空
reportedAmount	已确认的报表金额：完整且无账单时为零，未确认时为空

补零规则已经固定，查询只需选择字段。下面是实际请求，用 1-17 日看清边界：

```
{
  "columns": ["calendarDate", "dataComplete", "rawAmount", "reportedAmount"],
  "slice": [
    {"field": "calendarDate", "op": "[)", "value": ["2012-11-01", "2012-11-18"]}
  ],
  "orderBy": ["calendarDate"],
  "limit": 50
}
```

结果节选，空值统一记作“未确认”或“无记录”：

日期	已确认完整	原始金额	报表金额
11 月 1 日	1	8.91	8.91
11 月 2 日	1	无记录	0
11 月 14 日	1	0.99	0.99
11 月 15 日	1	无记录	0
11 月 16 日	0	无记录	未确认

即使未确认日已有部分记录，也不能把它当作完整日总额。本次还检查了 11 月 27-29 日：原始样本有金额，但按教学完整性声明，报表金额仍为空。

日历表提供连续日期，完整性声明决定能不能补零；第十七章的 calendar 滚动窗口只规定自然日范围，**不会自动补齐日期或补零**。

第四步：比较同样长的期间

发送：

> 截至 2012 年 11 月 15 日，账单金额比上月同期怎样？

这里先约定“上月同期”为 **10 月 1-15 日**，比较 **11 月 1-15 日**。如果业务要比相同工作日数，需要另行说明。

Codex 先检查两段日历各有 15 天、也各有 15 天已确认完整，再查询账单 QM：

```
{
  "columns": ["invoiceMonth", "sum(invoiceAmount) as amount", "count(invoiceId) as bills"],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-10-01", "2012-11-16"]},
    {"field": "invoiceDayOfMonth", "op": "<=", "value": 15}
  ],
  "groupBy": ["invoiceMonth"],
  "orderBy": ["invoiceMonth"],
  "limit": 5
}
```

范围写成左闭右开，11 月 16 日是排除的上界；月内日序条件同时去掉十月后半月。

期间	金额	账单数
10 月 1-15 日	28.76	3
11 月 1-15 日	23.76	3

差额 □5.00，变化约 □17.39%。差额与比例请求通过条件聚合交给引擎计算，沿用第十章的方法，AI 负责说明比较范围。

如果拿十一月前十五天和十月整月的 42.62 比，会得到约 □44.25%。算术没有错，但不能称为同口径的同期变化。

还有一个容易漏掉的问题：SQL 的 SUM 通常忽略空值。查询十一月整月，已知金额仍会得到 23.76，但日历显示 **30 天中只有 15 天已确认**。它只能叫“已确认部分合计”，不能说是十一月总额。完整性需要单独检查，不能只看返回了一个数字。

工作日、跨年周与其他边界

十一月前十五天在教学日历中有 11 个工作日。继续发送：

> 改为比较两个月的前 11 个工作日，只统计这些日期发生的账单。

Codex 在日历 QM 中筛选 isWorkday = 1、workdayIndex <= 11，得到十月 10.91、十一月 23.76，各 11 天。十月 6 日的周末账单被排除，因此结果和自然日比较不同。

工作日比较会排除周末账单，适不适合要看业务；不能据此推断音乐商店周末不营业。企业使用时，应提供实际营业、调休或节假日日历。

本章另核对了 2 月 29 日，以及 2012 年 12 月 31 日属于 ISO 2013 年第 1 周。周年与周号在准备日历时确定，查询直接使用已定义字段，不能只拼“自然年＋周号”。财年、财季、月末短月和闰日也应先确定规则，不能自动套自然年或随手减 365 天。

“最近七天”和“本周”不是同一个范围。历史练习固定日期，实际问数则先确认报告基准日、数据更新截止及业务时区；营业日跨午夜时，日归属应在数据准备或语义模型中统一。本章不展开跨时区转换和财务日历实现。

留下可复跑的核对

日历关联后的 2012 年仍为 366 天、83 张账单、477.53，没有因补齐或关联重复累计。日期角色、完整性与期间比较的代表查询已在 0.1.38 复跑，见当前验证；初验记录保留逐项独立核对。

本章 Foggy 知识点

- 普通日期属性的 timeRole、recommendedUse，以及实时元数据中的业务用途。
- 日历作为主表，账单先按日汇总再 LEFT JOIN，保留无业务日期。
- QM 固定公式统一处理已确认零值和未确认空值。
- 用普通 DSL 表达可比期间；同时核对覆盖天数、完整性和金额。

引擎执行已定义的规则；日历、完整性与比较口径仍需要业务和数据准备配合。下一专题再看另一类问题：销售额可以跨日相加，余额、库存为什么通常不行？

第 19 章 存量与快照：月底还剩多少

前两章分析一段时间内发生了多少。本章换个问题：截至某天，库存还剩多少？销售额可以跨日累加，每天的库存再加一遍，就把同一批货数了好几次。

沿用 Launcher 0.1.38 预发布版、CLI 0.1.25。Chinook 没有库存历史，本章另建少量教学数据，见实验说明。

第一步：准备历史与语义模型，先不查询

发送：

> 准备库存教学样例和语义模型，说明每张表一行代表什么。先不查询。

Codex 在独立的 time-business Namespace 中接入样例，校验、加载模型，再实时读取查询元数据。Markdown 仍由引擎根据已加载代码生成，不需要提前维护。

数据	一行代表什么	TM / QM 源码
对象目录	一个库存对象，A-D 共四个	TM / QM
变更日快照	一个对象在某日开始生效的库存，延续至下一次变更	TM / QM
变动流水	一笔增加或减少	TM / QM
连续日快照	一个对象在一个自然日的日末库存	TM / QM

变更日记录很少，直接列出来就能核对：

对象	生效日	库存
A	2024-01-01	100
A	2024-01-03	80
A	2024-01-05	120
B	2024-01-01	50
B	2024-01-04	40
C	2024-01-04	30

D 没有历史；C 在 4 日以前的状态未知。连续日快照由准备脚本按上述延续规则展开，未知保留为空。这是数据准备工作，不是引擎自动恢复了缺失历史。

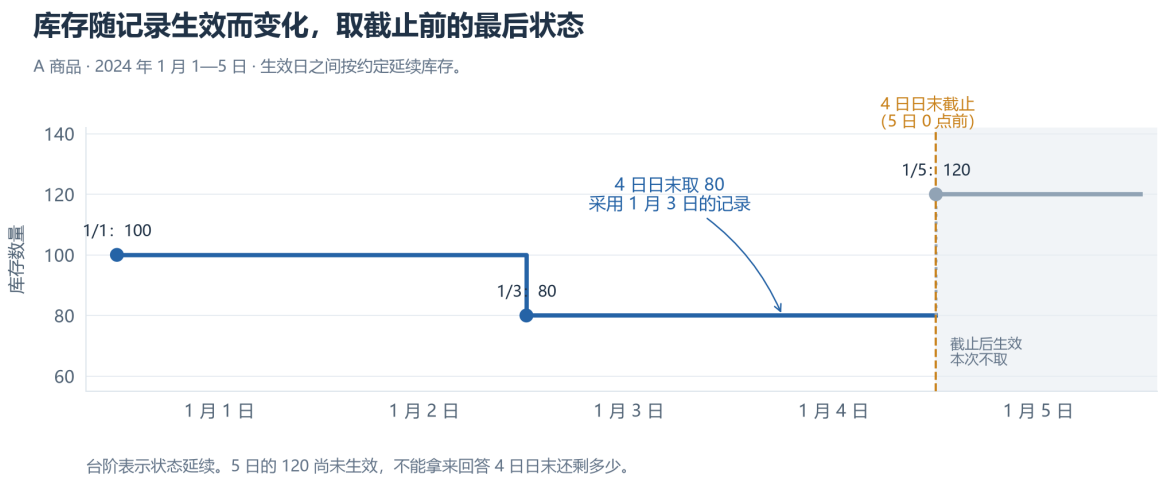
第二步：取每个对象的期末记录

发送：

> 看看截至 1 月 4 日日末各对象的库存，没有历史的也列出来。

Codex 从 TeachingStockSnapshotQueryModel 找每个对象 **1 月 5 日之前最大的生效日**，按对象与日期取回对应库存，再以 TeachingStockEntityQueryModel 的完整目录为左侧做 LEFT JOIN，保留没有历史的对象。

后一步使用前一步的结果，这就是分阶段查询。实际 Compose 脚本由引擎执行，完整语法留给组合分析系列。先看 A 的库存怎样随记录生效：



A商品库存从1月1日的100变为3日的80，再变为5日的120。查询4日日末，采用3日记录的80，5日的120不参与。

各对象结果为：

对象	实际采用的记录日	库存
A	2024-01-03	80
B	2024-01-04	40
C	2024-01-04	30
D	无	未知

已知部分合计 **150**，覆盖 3/4 个对象，不能称作全部库存。改查 5 日日末，A 变为 120，已知合计 **190**，D 仍然未知。

若截止前完全没有历史，目录中的四个对象仍会返回，库存均为空；不能补成四个零。

两个常见误区：MAX(quantity) 找到的是最大库存，不是最新库存；只取全表最大日期，又会漏掉当天没有变更的 A。若同一对象同一时刻允许多条记录，还得先约定版本或顺序，本例用唯一键避免这个歧义。

第三步：用流水核对期末

发送：

> 用 A、B 的期初和变动流水，核对 1 月 5 日日末库存。

两者期初共 **150**。A 减 20、加 40，净增 20；B 减 10。于是：

```
期初 150 + 变动 20     10 = 期末 160
```

这与快照中的 A 120、B 40 一致。C、D 缺少完整期初或流水，不能顺手算进去。没有历史，就不能靠一句“查截至当时”把历史补出来。

第四步：平均库存要按时间算

发送：

> 计算 A、B 在 1 月 1 日至 5 日的日均总库存。

连续五天，A 的库存为 100、100、80、80、120，日均 **96**；B 为 50、50、50、40、40，日均 **46**。两者每日总量的平均为 **142**。

查询先按日求两者总量，再对五个日总量求平均，见实际脚本。直接对两对象的十行求 AVG 会得到 **71**，那是“对象-日”的平均，不是日均总库存。

若只平均变更日记录，A 为 100、B 为 45，也不对：每条记录覆盖的天数不同。没有连续日快照时，应按有效时长加权。A 在这五天的计算是 $(100\times2 + 80\times2 + 120\times1)\div5 = 96$。真实业务还要确认按日末、营业时段还是连续时间取平均。

历史值还依赖历史归属

商品今天属于哪个部门，不一定等于当时的归属。查历史部门库存，需要有效期内的归属记录；只有当前部门字段，无法还原当时。

同样，事后补录后的历史重算值，可能不同于当时发布的报表。需要保留原发布值时，应另存报表版本或快照，不让 AI 猜过去看到了什么。

本章 Foggy 知识点

- **存量字段：** 同一时点可按对象汇总，跨日期的 SUM 不代表期末值；字段说明本身不会禁止错误查询。
- **分阶段查询：** 先按对象求最新日期，再关联对应记录；Compose 与基础篇的 QM 建模 JOIN 分开理解。
- **完整性：** 已知部分、未知对象与完整合计分别返回，未知不补零。
- **平均存量：** 先确认日粒度和完整覆盖，再算日均或按有效时长加权。

期末、无历史、流水和日均查询已在 0.1.38 教学端口回归，见当前验证。原始记录核对与旧版故障保留在实验记录中。

第 20 章 业务耗时：已完成多久，未完成等了多久

日期差看起来只是一道减法，但起止事件、截止时间和统计分母没定好，算得再准也会答错。本章用工单说明自然小时耗时、未完成等待与超时比例。

样例是 11 条虚构工单，覆盖同日、跨日、空结束、未来结束和错误时间。沿用上一章独立环境，Launcher 0.1.38 预发布版、CLI 0.1.25。

第一步：固定统计口径并创建模型

发送：

> 为工单样例建立耗时模型，按 2024-02-02 12:00 截止，超过 24 个自然小时算超时。先不查询。

Codex 读取样例和字段，编写TM与QM，校验、加载，再获取引擎实时生成的查询说明。

本例把分析截止存为固定的 analysisAt，保证复跑一致。调整截止时，应生成另一份分析输入或使用经验证的参数方案，不能把这个教学常量当成系统当前时间。

字段	含义
eligible	截至统计时已开始，且起止时间合法
completed	合法且实际完成时间不晚于截止
completedHours	仅已完成工单的实际耗时，其他为空
waitingHours	仅未完成工单截至统计时的等待，其他为空
elapsedHours	已完成取实际耗时，未完成取截至时等待
overdue	合法且已开始，已耗时严格大于 24 小时

缺开始时间、结束早于开始，单独列为异常；截止以后才开始的，不进入分母。恰好 24 小时不算超时。全部时间使用同一约定时区，本例不做跨时区转换。

把业务耗时固定在模型中

本例时间按秒记录，TM 使用 SQLite 的 julianday 相减后乘以 24，保留六位小数小时。完成时间在截止之后时，仍按未完成计算到截止，避免拿后来的结果回答“截至当时”。

示意公式是：

```
有效结束 = 截止前已完成 ? 实际完成时间 : 统计截止
已耗小时 = (有效结束     开始时间)换算为小时
```

实际表达式与异常处理放在 TM；查询时 LLM 选择语义字段。换数据库时适配 TM 的方言表达式，查询仍使用相同语义字段。

第二步：先分清日期差与完整小时

发送：

> 对比几条工单的日历日期差和完整小时数。

查询也可以临时使用日期差函数。Codex 选取工单 1-3，通过计算字段表达：

```
{
  "columns": ["ticketId", "dayDifference", "wholeHours"],
  "slice": [{"field": "ticketId", "op": "in", "value": [1, 2, 3]}],
  "calculatedFields": [
    {"name": "dayDifference", "expression": "DATEDIFF(finishedAt, startedAt)"},
    {"name": "wholeHours", "expression": "TIMESTAMPDIFF('HOUR', startedAt, finishedAt)"}
  ],
  "orderBy": ["ticketId"],
  "limit": 10
}
```

工单	日历日期差	完整小时
1	0	2
2	1	12
3	1	24

DATEDIFF(结束, 开始) 比较日历日期，跨午夜两分钟也可能相差 1 天；TIMESTAMPDIFF(单位, 开始, 结束) 返回完整单位，向零截断。参数顺序不同，不要写反。

完整小时不等于小数小时。**24 小时零 1 秒仍只返回 24 个完整小时**，若拿它判断“超过 24 小时”，就会漏报。因此后面的超时和平均仍使用模型中保留小数的耗时字段，口径不随查询函数变化。

非法单位、参数数量以及 SQLite 不支持的微秒单位会在校验阶段拒绝，错误定位到表达式；单位错误还返回适用选项。按诊断调整后重新校验，不执行被拒绝的请求。

第三步：分别查看完成与等待

发送：

> 查看已完成的平均耗时，再列出还没完成的工单和等待时长。

查询 TeachingWorkTicketQueryModel。截至固定时点，**8 条**合法且已开始，其中完成 **5 条**、未完成 **3 条**。

已完成耗时分别为 2、12、24、25、48 小时，平均 **22.2 小时**。未完成单列：

工单	截至时等待小时	已超时
5	28	是
6	2	否

工单	截至时等待小时	已超时
7	22	否

工单 7 实际在截止后两小时完成，但截至时仍未完成；不能提前把最终耗时拿来计算。

若把未完成都当成零，均值会被压低；若直接丢掉它们，又看不到积压。本章分别回答“已完成处理了多久”和“未完成等了多久”，不拿两类平均混用。

第四步：算超时比例，先说清分母

发送：

> 截至这个时点，已开始的有效工单中，超时占多少？

实际查询：

```
{
  "columns": [
    "sum(eligible) as eligible_count",
    "sum(completed) as completed_count",
    "sum(overdue) as overdue_count",
    "completedAverageHours",
    "overdueRate"
  ],
  "limit": 10
}
```

超时 3/8 = 37.5%，包含已完成超时的 2 条和仍在等待的 1 条。这与“已完成中的最终超时比例” 2/5 = 40% 是两个问题。

QM 的固定指标采用：

```
formula: 'sum(overdue) * 1.0 / nullif(sum(eligible), 0)'
```

乘以 1.0 避免 SQLite 的整数除法；分母为零时返回空。指标写进模型后，LLM 不必每次重新推导分母和除法，但仍要正确理解用户问的是哪一种超时。

第五步：看分布与异常

发送：

> 按 24 小时以内和超过 24 小时分组，另列时间异常的记录。

8 条有效工单中，5 条在 24 小时以内、3 条超过。这里分组的是**截至时已耗时**，包括未完成等待，不是完成耗时分布。

另外两条时间异常单列，一条尚未开始被排除。没有有效对象时，平均耗时和超时比例为空，不能补零。

默认响应会省略 NULL 字段。HTTP 调用加 `X-Foggy-Null-Output: explicit` 后，这次空指标查询明确返回 `{"completedAverageHours":null,"overdueRate":null}`。它只改变输出格式，不补零，也不额外生成结果行。

自然小时不扣周末、假期和暂停。若业务要求“两个工作日内处理”，还需要工作日历、上下班时间和暂停记录；不能直接把 48 小时改个名称。

本章 Foggy 知识点

- **模型方言表达式：** 在 TM 固化日期差、异常与固定截止规则，查询使用公开语义字段。
- **日期差函数：** DATEDIFF 的日历日期差、TIMESTAMPDIFF 的完整单位，与模型的小数耗时分开。
- **条件指标：** 用合法/已完成/超时标记聚合，明确各自分母。
- **固定计算字段：** QM 封装平均与比例，处理整数除法和零分母。
- **边界回归：** 空值、未来事件、恰好阈值和异常顺序都要验，执行成功不等于口径正确。

耗时、日期差与 24 小时零 1 秒的边界已独立核对，并在 0.1.38 教学端口回归。实操与复跑保存空值输出和错误诊断的实际响应。

第 21 章 同期群：同一批客户后来还买吗

月度销售额看整体变化，同期群则跟踪同一批对象：一月首次购买的客户，二月还有多少人购买？本章只做购买同期群和次月复购，不把购买记录解释成登录留存。

沿用 Launcher 0.1.38 预发布版、CLI 0.1.25。主例来自 Chinook 原始账单；三名虚构客户另表保存，用于核对重复购买和跨年。

第一步：准备购买历史，先不查询

发送：

> 为 Chinook 购买记录建立同期群模型，先不查询。

Codex 从账单副本准备 Purchase，增加年月及 monthIndex，创建TM与QM，校验、加载后实时获取查询说明。Markdown 由引擎生成，不另写一份静态说明。

```
月份序号 = 年 × 12 + 月    1
购买后第几个月 = 购买月序号    首购月序号
```

这样十二月到次年一月仍相差 1，不能直接把 202401 202312 当成月份差。

Chinook 只有样本内购买历史，没有完整获客记录。因此下面称**样本内首购**，不声称这些都是真实新客。若用户只能看部分历史，首次购买也只能按该可见范围定义，不能为补齐历史越权查询。

第二步：分阶段计算同期群

发送：

> 观察到 2009 年六月底，按样本内首购月份统计客户数，以及首购当月和随后两个月的购买人数。

截止固定为 2009-07-01 之前。本例把已提供样本在此前各自然月视为完整，这是教学观察约定；生产中须另确认数据完整性，不能从“最后一条交易日期”推断。

查询 TeachingPurchaseQueryModel 时，需要先确定每人首购，再按首购后的月份统计。这里用 Compose 分阶段完成，见可执行脚本：

1. 按客户取历史月份最小值，得到每人的样本内首购月。
2. 把购买记录去重到“客户-月”。同月买三次也只算一个购买客户。
3. 按客户关联两份结果，计算相对月份，再按首购月和相对月份统计人数。
4. 从首购结果单独统计同期群规模，作为固定分母。

与第十九章的库存查询一样，这里先理解各阶段的业务含义，Compose 完整语法留到组合分析系列。

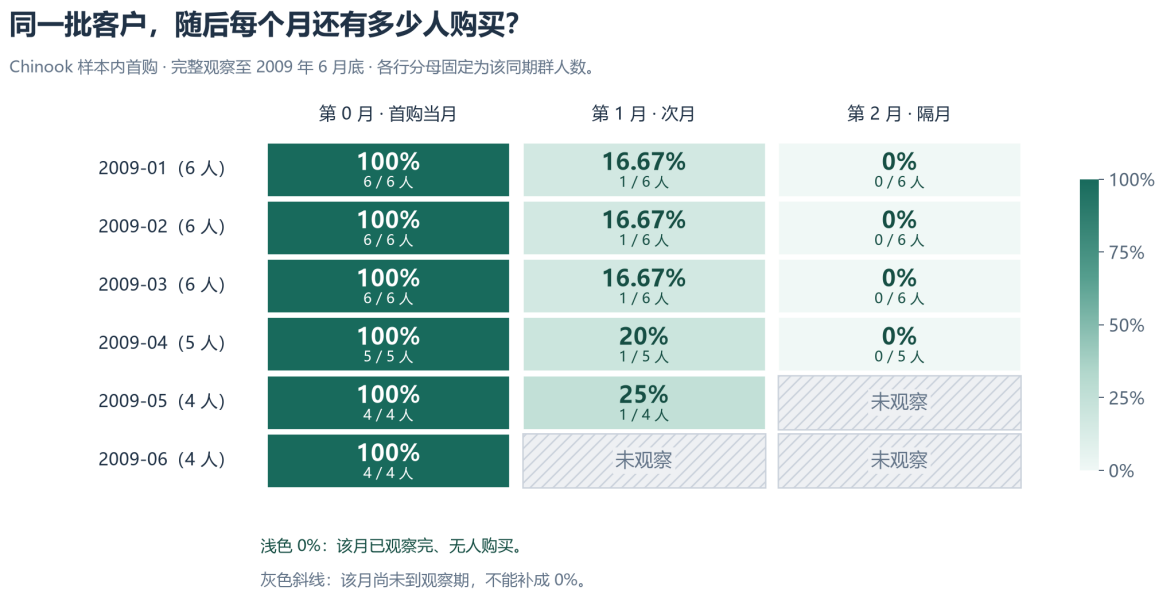
第三步：固定分母，区分零与未观察

发送：

> 整理成同期群表，计算复购比例，还没到观察期的标为未知。

引擎返回同期群规模、各月实际购买人数；结果整理脚本据固定截止补上 0-2 月格子，再计算人数比例。空格判断和展示是结果处理工作，不是一个原生留存算子。

主例用颜色展示购买比例，格内同时标出人数与固定分母：



样本内首购同期群热力图：一月至四月的第2月均为已观察的0%；五月第2月和六月第1、2月尚未观察，以灰色斜线区分。

一月同期群 6 人，二月有 1 人再次购买，次月复购比例为 1/6。分母不能换成二月全部活跃客户，也不能用账单张数替代购买人数。

把未观察的灰格补零，会让越新的同期群看起来越差。

第 0 月是进入同期群当月，不是“进入后的第一个完整月”。本章统计的是在对应自然月内是否购买，不是未来 30 天购买、连续每月购买或截至某月曾经复购。

第四步：核对客户，而不只看比例

发送：

> 抽查客户的首购和后续购买月份，再检查同月重复购买、跨年和观察不足的情况。

例如真实客户 2 在样本中的首购为 2009 年一月，二月再次购买；客户 4 在一月首次出现、四月再次购买，其四月活动属于第 3 月，不该填进第 2 月。

单独的教学边界模型有三名虚构客户：

客户	购买月份
1	2023-12 两次、2024-01 两次
2	2023-12 一次、2024-02 一次
3	2024-01 两次

截至一月底，十二月同期群 2 人、次月购买 1 人，比例 50%，没有被重复购买翻倍；第 2 月尚未观察。观察到二月底后，第 2 月变为 50%，一月同期群的次月则为已观察的 0%。跨年月份差、空样本也已核对。

还有一个容易藏在筛选里的错误：先把输入裁成 2012 年，再取最早购买，只能得到“2012 年第一次出现”，无法判断是否新客。展示近期同期群，应先在允许的历史范围内确定首购，再筛选要展示的首购月份。

“没再买”还不等于永久流失

低频采购客户连续一个月没买很正常。若要讨论流失或回流，要另定多久未购买、哪些行为算活跃，以及观察期是否足够。现有购买数据不能解释客户为什么离开，也不能证明以后不会回来。

本章 Foggy 知识点

- **阶段聚合：** MIN 确定首购、按客户与月份去重，再关联和汇总。
- **Compose：** 复用 QM 的公开字段，分阶段结果与普通单 QM 查询分工明确。
- **固定业务口径：** 连续月份序号、固定同期群分母与可见历史范围。
- **结果完整性：** 已观察且无购买补零，未到观察期保留空值；比例的整理步骤明确可复跑。

主例与边界样例已逐客户核对，主报告及两个观察期边界在 0.1.38 教学端口回归通过。实操与复跑、当前验证。

第 22 章 父子维：

“看看 Nancy 负责范围内的账单”，这里的范围是她本人，还是包括下属？本章把这个区别写进语义模型，让 AI 明确选择，再交给引擎执行。

继续使用 Chinook，当前教学环境为 Launcher 0.1.39、CLI 0.1.25。本例统计的是**按当前客户支持归属划分的账单金额**；样例没有历史人员归属，不能据此判断当时是谁完成了销售。

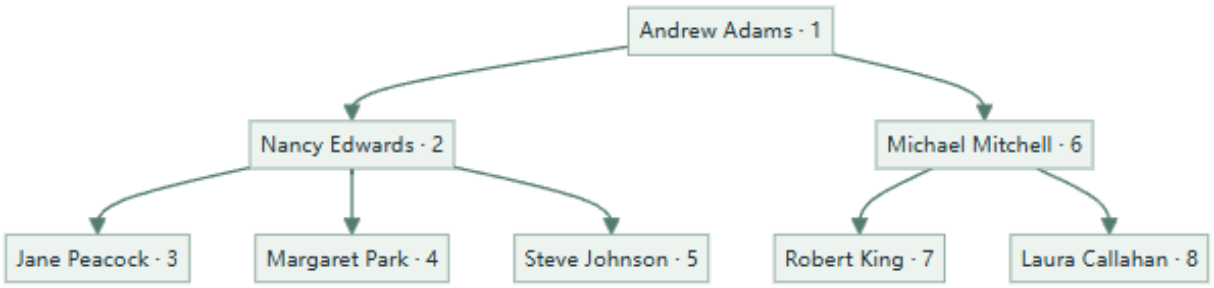
第一步：准备员工层级与模型，先不查询

在原练习目录发送：

> 用独立 Chinook 副本建立员工层级和账单范围分析模型，校验并加载，先不查询。

Codex 用准备脚本复制数据库，在副本中新增闭包表 EmployeeClosure；原账单、客户和员工记录不改。项目位于 lab/multidim，接入 Namespace chinook-multidim。

Chinook 已有 8 名员工，ReportsTo 指向上级。下面的箭头从上级指向下级：



查询与建模流程

第十二章的“曲目 → 专辑 → 艺术家”是多张表逐层关联；这里是同一员工表中的自引用树，深度由数据决定。

从一层关系，准备全部路径

ReportsTo 直接告诉我们 Jane 的上级是 Nancy。要问 Jane 还属于哪些上级范围，准备脚本会沿树向上走，将路径存入闭包表。例如：

AncestorId：上级（含本人）	EmployeeId：所属人员	distance：相隔层数
2	2	0，自身
2	3	1，Nancy → Jane
1	3	2，Andrew → Nancy → Jane

本例共 20 条闭包记录，包含 8 条自身记录；自身记录保证“本人及下属”不会漏掉本人。脚本同时检查上级是否存在、有无环路、上下级关系对是否重复。这是数据准备工作，不能用一句提示词代替。

把层级接到业务查询

本章创建两对模型，完整源码如下：

模型	主表与用途	源码
员工目录	Employee，列出员工和上下级，包含没有账单的成员	TM / QM
支持范围账单	Invoice，经客户的当前支持人员归属账单	TM / QM

账单 TM 的关联配置节选如下，从主表保留外层结构；姓名展示、属性与指标见完整源码：

```
export const model = {
    tableName: 'Invoice',
    dimensions: [{
        name: 'customer', tableName: 'Customer',
        foreignKey: 'CustomerId', primaryKey: 'CustomerId',
        dimensions: [{
            name: 'support', tableName: 'Employee',
            foreignKey: 'SupportRepId', primaryKey: 'EmployeeId',
            closureTableName: 'EmployeeClosure',
            parentKey: 'AncestorId', childKey: 'EmployeeId'
        }]
    }]
};
```

SupportRepId 来自直接上一层的客户表；parentKey、childKey 则分别指闭包表的上级列与所属人员列，其中也保留本人到本人的记录。配置关联后，QM 通过 {ref: invoice.customer.support} 公开支持人员维度。

四个文件校验、加载后，Codex 实时获取查询说明。Markdown 由引擎根据已加载代码生成，不提前维护。本次公开字段包括 support$id、support$caption、invoiceDate、invoiceAmount；JSON 元数据还标明 support$id 是层级维度，并列出允许的层级操作符。

第二步：先确认成员，再查金额

发送：

> 先列出员工目录，确认 Nancy Edwards 的 ID、上级和直接下级。

Codex 查 ChinookEmployeeDirectoryQueryModel：Nancy 的 ID 为 2，上级是 Andrew（1），直接下级是 Jane（3）、Margaret（4）、Steve（5）。后面的筛选使用这些 ID，姓名用于展示。

目录从员工表出发；账单分析从账单表出发。账单里没查到某人，不代表员工不存在。

第三步：比较本人和本人及下属

再发送：

> 比较 Nancy 本人，以及本人和所有下属负责的客户在 2012 年的账单金额与张数。

Codex 查询 ChinookSupportInvoiceQueryModel。日期、指标不变，只改变支持人员筛选：

操作符	范围
=	只匹配本人
childrenOf	直接下级，不含本人
descendantsOf	全部下属，不含本人
selfAndDescendantsOf	本人及全部下属

例如“本人及所有下属”的实际 DSL：

```
{
  "columns": ["sum(invoiceAmount) as amount", "count(invoiceId) as bills"],
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]},
    {"field": "support$id", "op": "selfAndDescendantsOf", "value": 2}
  ],
  "limit": 10
}
```

只查本人时，将 op 改成 =。两条请求分别校验、执行，结果为：

Nancy 的范围	账单张数	金额
仅本人	0	空值
本人及所有下属	83	477.53

样例没有客户直接归属 Nancy，账单由她的三名下属负责。因此本人查询的 COUNT 为 0、SUM 为空；默认响应省略了空的 amount 字段，不能把它误读成查询失败。

AI 提交的是成员范围，客户关联与闭包查询由引擎落实；无需让 AI 每次重写递归 SQL。普通 = 也不会因为配置了父子维，就自动包含下属。

第四步：直接下级不等于所有下属

继续发送：

> 再比较 Andrew 的直接下级和全部下属，看看 2012 年账单范围有什么不同。

Andrew（1）的直接下级只有 Nancy（2）、Michael（6）；Jane 等支持人员在更深一层。目录与账单结果分别为：

范围	员工目录中的 ID	账单张数	金额
直接下级 childrenOf	2、6	0	空值

范围	员工目录中的 ID	账单张数	金额
全部下属 descendantsOf	2、3、4、5、6、7、8	83	477.53

层级条件匹配的是账单当前归属的人员，不会自动把每个直接下级的全部下属一起算入。

没有下属的员工同样能说明“包含自身”的作用：Jane（3）本人有 **28 张、146.60**；只查她的下属没有账单，查“本人及下属”仍返回她本人的结果。

用 maxDepth 限制层数

再发送：

> 比较 Andrew 下方一层、两层的员工和账单范围。

maxDepth 限制经过几层上下级关系，起点算第 0 层。先查员工目录 ChinookEmployeeDirectoryQueryModel：

```
{
  "columns": ["employee$id", "employee$caption", "employee$parentId"],
  "slice": [
    {"field": "employee$id", "op": "descendantsOf", "value": 1, "maxDepth": 1}
  ],
  "orderBy": ["employee$id"],
  "limit": 50
}
```

一层返回 Nancy（2）、Michael（6），与直接下级一致；改成两层，返回 ID 2-8。它限制的是关系层数，不是返回几个人。

账单改查 ChinookSupportInvoiceQueryModel，沿用前面的金额、张数请求，将 support$id 条件改为 descendantsOf、值为 1，并加同样的 maxDepth。2012 年范围内，一层为 **0 张、金额空值**，两层为 **83 张、477.53**。真实负责客户的三人都在 Andrew 下方第二层。

第五步：向上查各级上级

发送：

> 看看 Jane 的各级上级，再把她本人也列出来。

回到上面的完整员工目录请求，只将 slice 替换为：

```
[
  {"field": "employee$id", "op": "ancestorsOf", "value": 3}
]
```

图中箭头表示向上查找，层数从 Jane 算起：



查询与建模流程

操作	返回范围
ancestorsOf	Nancy、Andrew，不含 Jane
selfAndAncestorsOf	Jane、Nancy、Andrew
ancestorsOf 加 maxDepth: 1	只到直接上级 Nancy
selfAndAncestorsOf 加 maxDepth: 1	Jane 与 Nancy

这里按关系顺序列名字；实际请求按 ID 排序，展示顺序不代表层级。只需直接上级 ID 时，也可读取 employee$parentId。

查出 Jane 的上级，与统计这些上级负责范围的账单，是两件事。 本节返回员工；下一章才把账单分别汇总到本人及各级上级。

留下边界核对

没有下属、没有账单和员工不存在，要分别判断。本次也核对了叶子节点和不存在的 ID 999，避免把空业务结果误当成员目录。

范围用原始 ReportsTo 独立遍历，金额用原客户归属、账单行和 Decimal 核对。原初验记录保留；本轮在 **0.1.39 验证了 12 项向上/层数查询、3 项下钻衔接回归**，均通过，模型与数据库未改。请求、响应与复跑。

本章 Foggy 知识点

- **父子维建模：** 用闭包表记录各级上下级及本人关系，区别于固定的多表嵌套关联。
- **显式查询范围：** 区分本人、直接下级、全部下属和各级上级，明确是否包含本人；用 maxDepth 限制关系层数。
- **目录与事实：** 先确认成员，再查业务；无结果不直接等于成员不存在。
- **独立核对：** 用原关系遍历验证范围，用原账单核对张数与金额。

下一章用这些人员范围完成层级汇总与下钻。

第 23 章 沿层级汇总与下钻：为什么总计不能再相加

上一章确定“选哪些人”，包括向下查、向上查和限制层数。本章再问：“这些账单，要汇总到谁名下？”成员范围与汇总分组是两件事。

仍用独立的 chinook-multidim 项目，当前为 Launcher 0.1.39、CLI 0.1.25。人员归属沿用当前客户支持分配，不能当作历史销售业绩。

第一步：公开层级汇总字段，先不查询

发送：

> 为现有账单模型增加层级汇总查询入口，校验并加载，先不查询。

原 TM 和闭包表可以复用。本章新增 ChinookSupportRollupQueryModel.qm，同时公开层级汇总对象与实际归属人员。汇总对象包括人员本人及各级上级，其中两项配置为：

```
{ref:'customer.support$hierarchy$id', name:'ancestorId', alias:'ancestorId'},
{ref:'customer.support$hierarchy$caption', name:'ancestorName', alias:'ancestorName'}
```

这两项放在 QM 的 columnGroups[].items 中。ref 引用 TM 的层级视角，name、alias 明确查询字段和返回名称。完整外层结构见源码。

校验、刷新后，Codex 实时获取查询说明，确认 ancestorId、ancestorName 已公开，再写 DSL。两者分别表示汇总对象的 ID 和名称；本例名称取员工姓氏，完整姓名可按 ID 对照上一章员工目录。

第二步：比较实际归属与层级汇总

发送：

> 看看 2012 年账单按实际支持人员、按本人及各级上级汇总，结果有什么区别。

按 support$id、support$caption 分组，得到实际负责客户的三人：

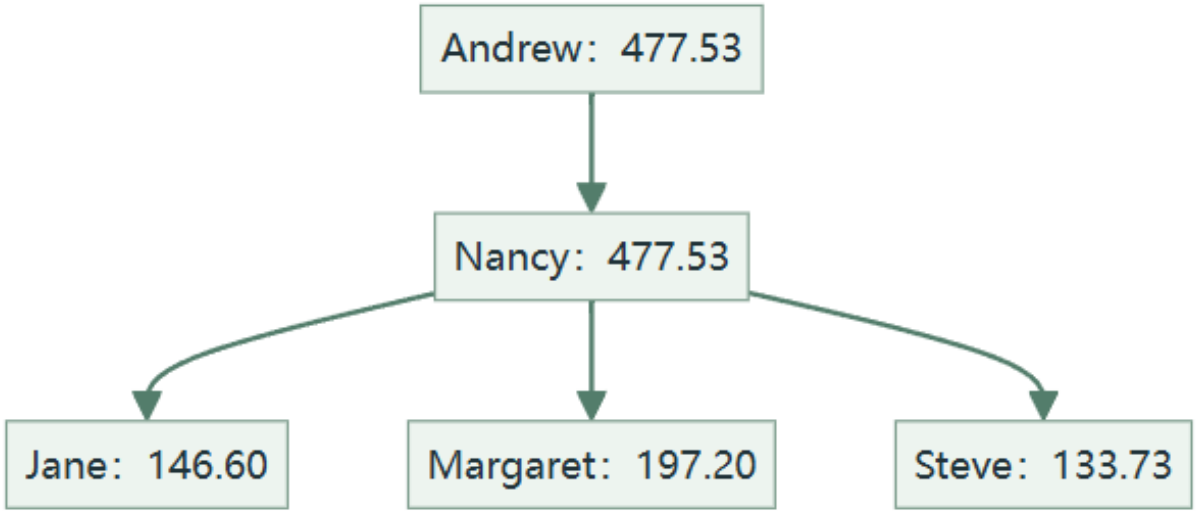
人员	金额	账单数
Jane Peacock（3）	146.60	28
Margaret Park（4）	197.20	29
Steve Johnson（5）	133.73	26

改按本人及各级上级汇总，使用：

```
{
  "columns": ["ancestorId", "ancestorName", "sum(invoiceAmount) as amount", "count(invoiceId) as bills"],
```

```
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}
  ],
  "groupBy": ["ancestorId", "ancestorName"],
  "orderBy": ["ancestorId"],
  "limit": 50
}
```

实际返回五个有业务的节点：Andrew（1）477.53、Nancy（2）477.53，以及上面三人的金额。没有账单的分支不会因为员工目录里存在，就自动返回一行。



查询与建模流程

同一张 Jane 负责的账单，会计入 Jane、Nancy 和 Andrew 三个节点。五行金额相加是 1,432.59，却不代表发生了这么多业务；它把同一批账单算了三遍。

本例三名支持人员互不重叠，合计能对回 Nancy 的 477.53。但这不是“父级永远等于下级之和”：如果经理本人也直接负责客户，还要加上经理自身的账单。平均值、去重人数也不能照搬金额相加的做法。

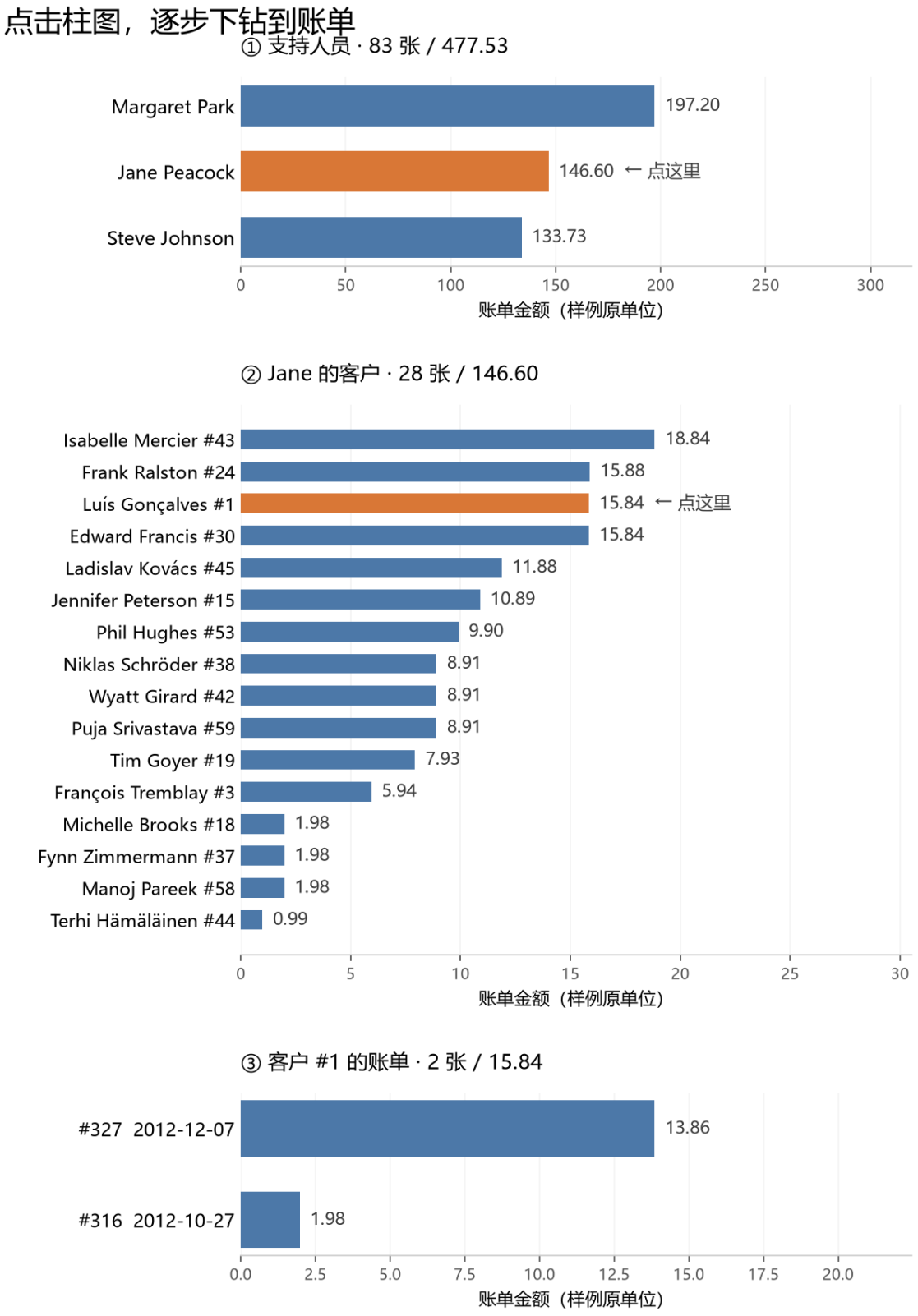
本章另用员工目录计数验证了这个边界：Nancy 子树共 4 人，包含她本人和 3 名下属。没有往原账单中补造经理业务。

第三步：从上级范围下钻到账单

发送：

> 展开 Nancy 范围内各支持人员的金额，再查 Jane 对应的客户和账单，保持 2012 年范围。

Codex 先用 support$id selfAndDescendantsOf 2 限定 Nancy 范围，按实际支持人员分组。下面用柱图演示“点哪里，看到什么”；橙色表示本次点击项。



日期始终为 2012 年；橙色表示本次点击项。当前客户支持归属，不代表历史销售业绩。

点击人员柱查看客户，再点击客户柱查看账单

打开可点击演示：点 Jane，看到她在 2012 年有账单的 16 位客户，合计 28 张、146.60；再点 Luís Gon□alves（客户 #1），看到他的 2 张账单、15.84。点击上方路径可返回；其他人员和客户也可点选。

每次点击都保留 2012 年日期条件：选 Jane 时增加 support$id = 3，选客户时再增加 customerId = 1。选中一根柱子会缩小范围；若将当前层全部柱子展开到下一层，金额仍应对回。读取明细时不再按各级上级展开，避免同一账单出现多行。

图表负责点选与返回，查询端负责带上日期和成员条件。这份演示预载已验证结果，可离线操作；接入实际问数时，每次点击再提交对应查询。

还有一个容易混淆的地方：把 Andrew 和 Nancy **合并为一个成员筛选范围**，不等于把两个节点的汇总行相加。本次使用 selfAndDescendantsOf、值 [1, 2]，仍返回 **83 张、477.53**；同一账单只需属于范围中的任一节点。

本章核对与 Foggy 知识点

本章 **7 项查询**通过：层级汇总、单一经理、人员展开、客户汇总、账单明细、重叠范围合并及内部节点包含自身。核对依据为原始 ReportsTo 遍历、客户归属与账单行，金额用 Decimal；完整请求与复跑入口见实操记录。

- **层级汇总：** 用闭包关系将账单汇总到人员本人及各级上级，QM 可用明确别名公开字段。
- **范围与粒度：** 层级操作符决定取哪些事实，分组字段决定按谁展示。
- **下钻：** 保留日期与范围，逐步切换人员、客户和账单粒度。
- **避免重复：** 本人与各级上级的汇总行存在重叠，不能直接相加；合并成员范围按事实集合核对。

下一章改看另一种组织方式：把支持人员、国家和月份排成交叉表。

第 24 章 Pivot：把多个维度放到一张交叉表

上一章沿员工树查看业务范围。本章换个问题：“各支持人员负责的账单，在不同国家、不同月份分别有多少？”这里需要的是交叉表。

沿用 chinook-multidim 项目，Launcher 0.1.39、CLI 0.1.25。

第一步：准备月份字段，先不查询

发送：

> 复用支持归属账单分析，增加保留年份的账单月份字段，校验并加载，先不查询。

本章新增 ChinookSupportPivotModel.tm 和 ChinookSupportPivotQueryModel.qm，原模型不改。TM 沿用已有客户、人员关联，在 properties 中增加：

```
{
    column:'InvoiceDate', name:'yearMonth', caption:'账单年月', type:'STRING',
    description:'YYYY-MM，按账单日期取自然月，保留年份。',
    dialectFormulaDef:{sqlite:{
        builder:(alias)=>`strftime('%Y-%m', ${alias}.InvoiceDate)`
    }}
}
```

QM 公开 yearMonth，查询时 AI 直接选字段；月份表达式由建模代码维护，更换数据库时需调整方言。

Codex 校验、刷新，再实时获取查询说明，确认人员、国家、年月与金额、账单数均已公开。Markdown 由引擎按已加载代码生成，不必提前制作。

第二步：先确认要放哪些行、列

发送：

> 比较 2012 年十月、十一月各支持人员的账单金额与张数，再按账单国家展开。

先用普通分组确认金额与张数，再与 Pivot 对照。只看人员和月份，长表转成交叉表，就是把月份移到列上：

长表 → 交叉表：金额没变，月份从行变成列

2012 年 10—11 月 · 先只按支持人员和月份汇总 · 金额沿用样例原单位

支持人员	月份	金额
Jane	10 月	24.78
Jane	11 月	7.92
Margaret	10 月	10.91
Margaret	11 月	24.75
Steve	10 月	6.93
Steve	11 月	4.95

→

支持人员	10 月	11 月
Jane	24.78	7.92
Margaret	10.91	24.75
Steve	6.93	4.95

相同的 6 个汇总值，只改变摆放方式。张数可作为第二个指标一起放入。

相同金额从人员月份长表转为交叉表

加上国家后，普通长表共有 **14 个有业务的“人员 × 国家 × 月份”组合**。Pivot 把它们放到三个位置：

位置	本例选择
行轴 rows	支持人员、账单国家
列轴 columns	账单年月
指标 metrics	账单金额、账单数

同一国家可出现在多人下面，行身份要同时包含人员和国家。

第三步：用 Pivot 表达交叉分析

再发送：

> 把刚才的结果做成交叉表：按支持人员、账单国家分行，月份分列，显示金额和张数。

实际请求如下，目标 QM 为 `ChinookSupportPivotQueryModel`：

```
{
  "pivot": {
    "rows": ["support$id", "billingCountry"],
    "columns": ["yearMonth"],
    "metrics": ["invoiceAmount", "invoiceCount"],
    "outputFormat": "flat"
  },
  "slice": [
    {"field": "invoiceDate", "op": "[)", "value": ["2012-10-01", "2012-12-01"]}
  ]
}
```

日期由 slice 限定，metrics 使用已建模的原生指标。pivot.columns 定义列轴，与普通查询的顶层 columns 含义不同，两者不能同时出现。Pivot 也不与 timeWindow 混用。

flat 返回逐行记录，金额与张数逐格对回普通分组。改成 grid 后，返回行头、列头和单元格矩阵：14 行、4 个指标列，即两个月各放金额、张数。缺少的组合为 null，展示时保留为空。

第四步：增加小计、总计与 TopN

基础矩阵通过后，再发送：

> 增加人员小计和总计，再单独查看两月合计金额最高的两名支持人员。

在 pivot.options 加入 rowSubtotals: true、grandTotal: true，返回人员小计与各月总计；响应中的 _sys_meta 用于区分这些汇总行。

展开交叉表：看 Jane 各国账单落在哪个月

2012 年 10—11 月 · 色深只表示金额，张数不参与色阶

人员 / 账单国家	10 月金额	张数	11 月金额	张数
Jane 合计	24.78	4	7.92	2
Brazil	1.98	1	—	—
Canada	3.96	1	—	—
France	16.86	1	—	—
Germany	—	—	1.98	1
Hungary	—	—	5.94	1
India	1.98	1	—	—
Margaret 合计	10.91	1	24.75	3
Steve 合计	6.93	2	4.95	2

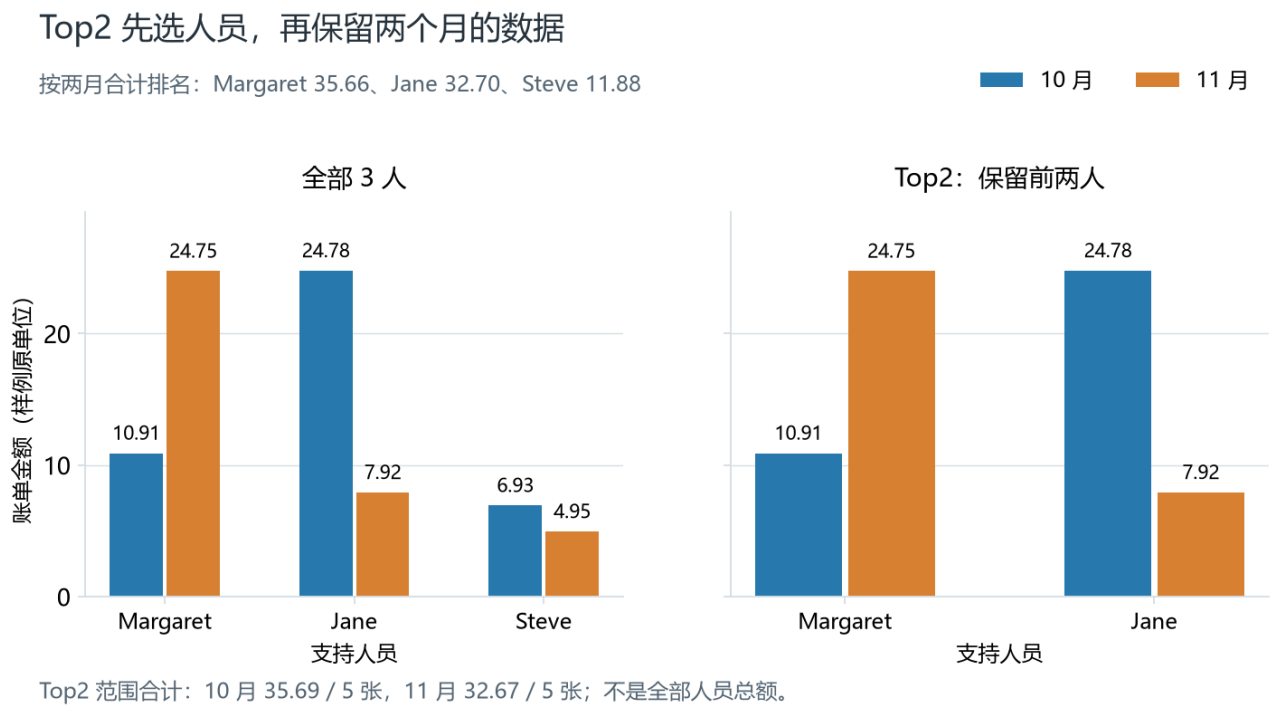
— 表示该组合没有返回数据，不改写成 0；人员合计不能再与展开的国家行相加。
点击演示可展开其他人员。这里的“人员 → 国家”是分组顺序，国家不是员工下级。

展开 Jane 的国家明细，金额用色深表示，张数单独显示

TopN 要配置在轴上。本例将 rows 改为：

```
[
  {"field": "support$id", "orderBy": ["-invoiceAmount"], "limit": 2},
  "billingCountry"
]
```

引擎按两个月合计选择人员，保留入选人员全部月份的国家单元格。顶层 limit 截取记录不能替代轴 TopN；加总计时，也只统计入选范围。



Top2 按两月合计保留 Margaret 和 Jane，仍展示两人各月金额

打开可点击演示：点击人员展开国家，或切换下方柱图的“全部 3 人 / Top2”。演示预载已验证结果，不实时调用引擎；展开不改变范围，Top2 只改变下方柱图的人员范围。

选讲：树形输出不等于层级汇总

上面按国家分组；若要沿员工上下级排列结果，可另用树形输出。

先发送创建任务：

> 为支持人员建立树形查询入口，校验并加载，先不查询。

本例新增 ChinookSupportTreeQueryModel.qm，复用本章 TM。嵌套人员的上级属性原名为 customer_support$parentId；树形入口将它明确公开为 support$parentId，与人员 ID 配套：

```
{ref:invoice.customer.support$parentId,
 name:'support$parentId', alias:'support$parentId'}
```

加载并读取实时说明后，再发送：

> 按员工层级展示 2012 年十月、十一月合计的账单金额。

本次行轴配置 hierarchyMode: "tree"，输出用 tree，不加交叉补全或总计。结果中 Jane、Margaret、Steve 分别为 32.70、35.66、11.88；Nancy 没有直属账单，不会凭树形格式自动生成一笔下属合计。

配套测试另用员工目录核对了完整 8 节点树。树形输出组织节点；需要汇总到人员本人及各级上级，仍使用上一章的层级视角。

本章 Foggy 知识点

- **月份属性：** 在 TM 中定义方言计算，QM 公开后由 AI 直接引用。
- **Pivot 结构：** 用行轴、列轴和原生指标表达交叉分析，区别于员工父子树。
- **范围与展示：** 先筛选事实，再核对每格、缺组合、合计与轴 TopN。

本章结果已与原始账单、员工关系独立核对，完整请求与复跑方法见实操记录。下一章继续在交叉表上计算占比与基准比值。

第 25 章 透视结果再计算：占父级多少，基准又是谁

上一章把金额排成了交叉表。本章再看两种关系：各国家占所属人员的多少，各月相对基准月又是多少。两者都要先确定分母。

环境沿用 Launcher 0.1.39、CLI 0.1.25。

第一步：回到完整交叉表

复用 ChinookSupportPivotQueryModel，无需新建模型。回到上一章第三步的完整矩阵：按支持人员、国家分行，年月分列，期间为 2012 年十月、十一月，先不沿用 TopN 或树形入口。Codex 实时读取查询说明后，再构造请求。

这里的“父级”指**行轴的上一层分组**：某国家所属的支持人员，而非员工树中的上级。

第二步：国家金额占所属人员多少

发送：

> 加上每个国家占所属支持人员当月账单金额的比例。

在上一章 pivot.metrics 中保留 invoiceAmount，再加入：

```
{
  "name": "countryShare",
  "type": "parentShare",
  "of": "invoiceAmount",
  "axis": "rows",
  "level": "billingCountry",
  "parentLevel": "support$id"
}
```

分子与分母保留同一月份：

```
某国家占比 = 该人员在该国家的当月金额 ÷ 该人员在所有纳入国家的当月金额
```

引擎返回 Jane 十月法国金额 16.86、占比约 68.04%，与原始账单的 16.86 ÷ 24.78 一致。十一月的父级金额是 7.92，分母要跟着月份变化。

level、parentLevel 指定相邻行轴层级，of 引用同一指标列表中的原生可加度量。完整请求见配套 JSON。

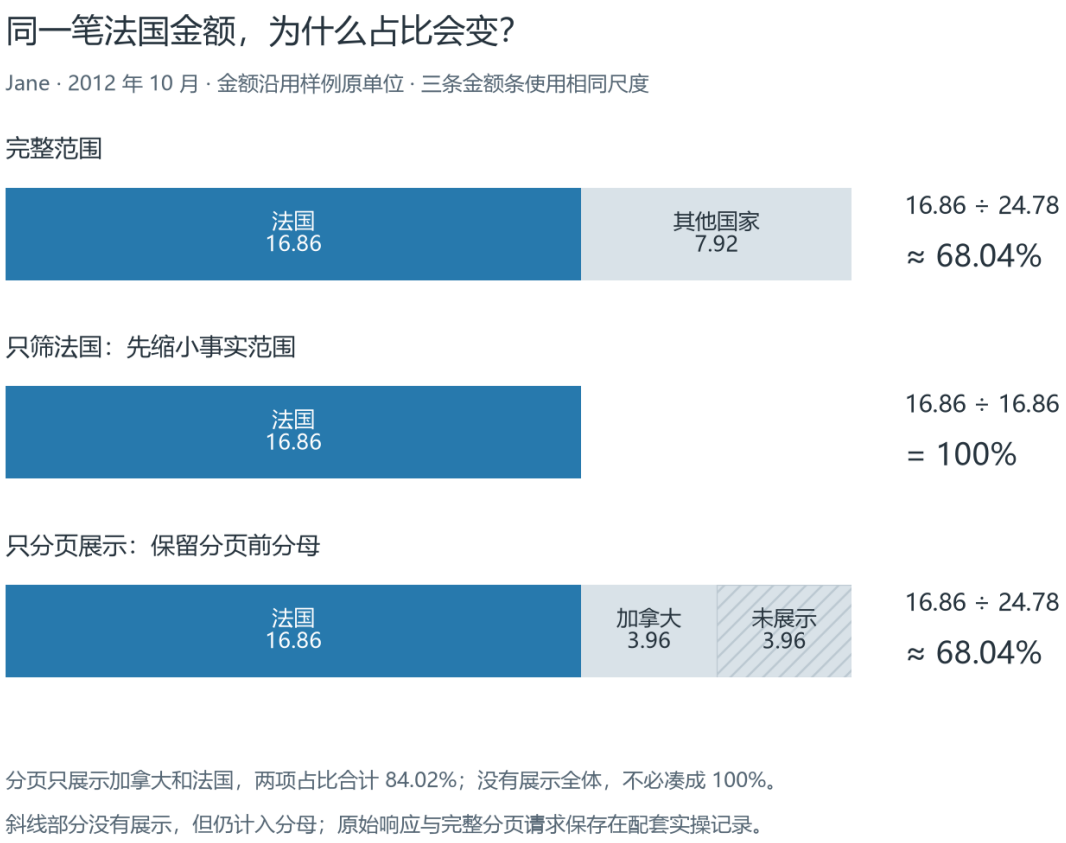
第三步：区分筛选与展示分页

再发送：

> 只看法国，重新查询各支持人员当月的金额占比。

slice 只留下法国后，分子与父级金额都只含法国，有业务的月份占比变为 100%。若想比较“占所有国家”，就要保留完整分母，不能假设 parentShare 会忽略筛选。

如果只想截取展示项，就不要先筛掉事实。恢复全部国家后，本次分页请求对每人的国家项跳过一项、取两项，并在占比指标中设置 denominatorScope: "prePageParent"，使用分页前的父级金额作分母；缺少这项声明，请求会在校验阶段拒绝。



完整范围、只筛法国与仅分页展示时的分母对照

筛选决定哪些事实参与计算，分页决定展示哪些项。只有完整、互斥且使用同一分母的子项，占比才适合按 100% 核对。

第四步：相对首月，不是环比

上面在同一个月内比较国家与所属人员；下面改为同一个人员跨月比较，分母换成基准月金额。

再发送：

> 恢复全部国家，比较各支持人员十一月相对十月的金额比值。

行轴只保留 support$id，列轴为 yearMonth。指标列表改为：

```
[
  "invoiceAmount",
  {
    "name": "index",
    "type": "baselineRatio",
```

```
    "of": "invoiceAmount",
    "axis": "columns",
    "baseline": "first"
  }
]
```

本次列序为十月、十一月，Jane 十一月返回 0.3196…，即 7.92 ÷ 24.78，表示十月的约 31.96%。若要表达增长率，还要减 1。

last 改用十一月作分母，Jane 十月返回约 3.1288。首列或末列按整张表的列轴确定，不会改成每个人各自有数据的第一列或最后一列。

本次另用独立的 9 行教学数据核对边界：分母为零、正负抵消为零或基准缺失时，比例为空，默认响应省略该字段；分母有效而当前金额为零时，才返回 0。人工数据不混入原 Chinook。

“相对首月”始终对比同一基准，环比则逐月换分母；同比、累计和滚动仍回到时间分析系列。

本章 Foggy 知识点

- parentShare： 按相邻行轴分组定义父级分母，列轴与筛选范围也参与口径。
- baselineRatio： 相对列轴首项或末项的比值，区别于增长率和环比。
- **如实解释：** 先确认分母、列序和缺值，再把数字写成业务结论。

完整请求、原始响应、边界数据与独立核对脚本见实操记录。

第 26 章 把查询结果交给下一步：Compose 与 CTE

前面主要在一个 QM 中完成分析。接下来，有时要先得到一组结果，再筛选、关联或继续计算。Foggy 用 Compose 组织这些查询阶段。

本章先做一个小例子：按客户汇总 2012 年账单，筛出年度金额至少为 10 的客户，再带上姓名。这个问题也能用普通分组与 having 完成；这里拆开，是为了看清阶段之间怎样传递字段。后两章再处理多个查询结果的组合。

环境使用 Launcher 0.1.42、CLI 0.1.25，本章原请求已复跑通过；金额仍用 Chinook 样例原单位。

第一步：准备本系列的练习目录

在原练习目录中发送：

> 复用客户目录和账单模型，准备 Compose 练习项目，校验并加载，先不查询。

本章配套项目为 lab/composition，Namespace 为 chinook-composition。模型定义复用前面的内容，副本加载到这个 Namespace：

查询入口	本章使用的公开字段	对应 TM
ChinookInvoiceQueryModel	customerId、invoiceDate、invoiceAmount	账单 TM
ChinookCustomerQueryModel	customerId、customerName	客户目录 TM

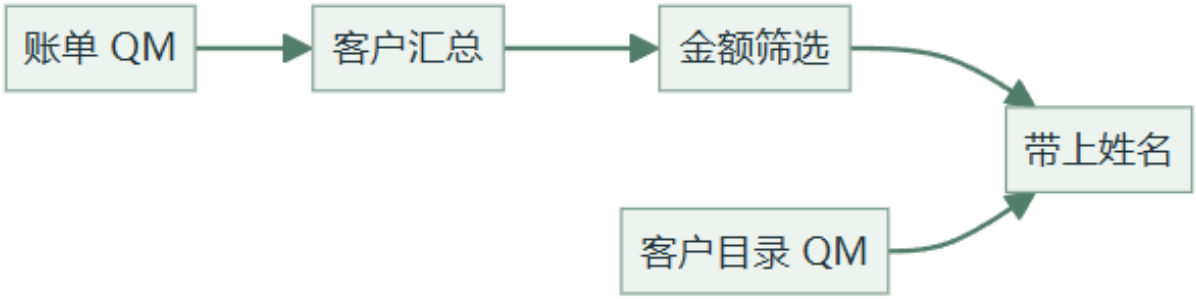
Codex 校验、加载后，实时读取这两个 QM 的查询说明。Markdown 仍由引擎按已加载代码生成，无须提前编写。已有自己的项目时，使用实际绑定的 Namespace。

第二步：用 Compose 串起查询

准备完成后发送：

> 用 Compose 查出 2012 年账单合计至少为 10 的客户，带上姓名。

查询分成下面几步：



查询与建模流程

Codex 生成的完整 fsscript 脚本如下，由 CLI 的 compose 入口执行：

```
const totals = dsl({
    model: 'ChinookInvoiceQueryModel',
    columns: ['customerId as customer_id', 'sum(invoiceAmount) as total_amount'],
    slice: [{ field: 'invoiceDate', op: '[)', value: ['2012-01-01', '2013-01-01'] }],
    groupBy: ['customerId']
});

const selected = totals.query({
    columns: ['customer_id', 'total_amount'],
    slice: [{ field: 'total_amount', op: '>=', value: 10 }]
});

const customers = dsl({
    model: 'ChinookCustomerQueryModel',
    columns: ['customerId as directory_customer_id', 'customerName as customer_name']
});

const result = selected.join(customers, 'inner', [
    { left: 'customer_id', op: '=', right: 'directory_customer_id' }
]).query({
    columns: ['customer_id', 'customer_name', 'total_amount'],
    orderBy: ['customer_id'], limit: 100
});

return { plans: result };
```

dsl(...) 从 QM 建立输入计划；totals.query(...) 在客户汇总上继续筛选；join(...) 按客户 ID 带入目录中的姓名。两侧的 ID 使用不同别名，最后只保留需要返回的列。

在这次投影中，customerId 输出为 customer_id，汇总金额输出为 total_amount。后续使用这两个阶段字段，未输出的 invoiceDate 不会自动传下去。

注意这里的两次 slice：第一处筛选原账单日期，第二处筛选已经汇总出的 total_amount。数据所处的阶段不同，条件的含义也不同。

第三步：先校验、看预览，再执行

Codex 按 compose validate → compose preview → compose execute 处理。校验检查字段与计划是否成立，预览给出生成的 SQL 和参数，通过后再执行。

CTE（公用表表达式）是 SQL 中用 WITH 给一个查询结果命名，再供后续查询引用的写法。本次预览的 SQL 使用了 CTE 与子查询，日期和金额门槛作为参数传入。

Compose 描述语义查询阶段，CTE 属于生成的 SQL。totals 等脚本变量保存查询计划，SQL 中的 CTE 名由引擎生成；计算交给数据库，AI 无须抄写中间客户 ID。CTE 也不保证中间结果会物化成临时表。

逐阶段独立核对的结果为：

阶段	客户数	账单金额合计
年度汇总 totals	47	477.53
金额至少为 10 的 selected	20	344.84
关联姓名后的 result	20	344.84

最终结果节选：

客户 ID	客户姓名	年度账单金额
1	Luís Gonçalves	15.84
5	František Wichterlová	18.84
7	Astrid Gruber	11.88

本例账单客户都有目录记录，inner 关联没有减少入选人数。把门槛改为 1000 后，完整链路正常返回空列表；它表示没有符合条件的客户。

字段写错时怎么处理

本次故意在后续阶段选择 invoiceDate，校验返回 COMPOSE_SCRIPT_INVALID，指出该字段不在来源输出中，未进入执行。需要限定账单期间，就把条件放在输入账单阶段；若改为按日分析，再相应调整分组与输出。

换个问题：没有账单的客户也保留

再发送：

> 列出所有客户及其 2012 年账单合计，没有账单的客户也保留。

这时应以完整客户目录作左侧，left 关联未加金额门槛的年度汇总。本次请求返回 **59 位客户：47 位有账单，12 位没有当期账单**。未匹配的金额为空，不能解释成一张金额为零的账单。

第十四章把常用多 TM 关联定义在 QM 中；这里组合的是本次查询的阶段结果。两种方式都需要先确认每行代表什么，以及关联键是否唯一。

本章 Foggy 知识点

- **Compose：** 用 dsl、派生 query 和 join 组合已有 QM 的查询计划。
- **阶段字段：** 后续使用前一阶段输出的列及别名，筛选条件放在对应的数据阶段。
- **CTE 与执行：** 通过预览查看 SQL 的阶段组织，校验通过后再执行，不把查询计划当作已取回的数据。

本章 5 项查询执行与 1 项预期校验拒绝已实际验证，金额和客户集合独立核对。源码、实时元数据、响应与复跑方式见本章实操记录。下一章分别汇总账单与明细，再关联两个结果核对金额。

第 27 章 先汇总，再关联两个查询结果

上一章把查询结果交给下一步。本章让两个 QM 各自汇总，再把结果放在一起核对：**2012 年每位客户的账单金额，与明细金额是否一致？**

沿用 chinook-composition 项目，Launcher 0.1.42、CLI 0.1.25，本章原请求已复跑通过。这里仍是 Chinook 的账单与明细，金额沿用样例原单位。

第一步：确认两侧口径，先不查询

发送：

> 准备账单和明细分析，复用已有模型，校验并加载，先不查询。

本次复用 账单 QM 和 明细 QM，已有模型直接使用。Codex 实时读取查询说明，确认：

查询入口	客户键	日期	金额
账单 QM	customerId	invoiceDate	invoiceAmount，来自账单总额
明细 QM	invoice$customerId	invoice$invoiceDate	lineAmount，交易单价乘数量

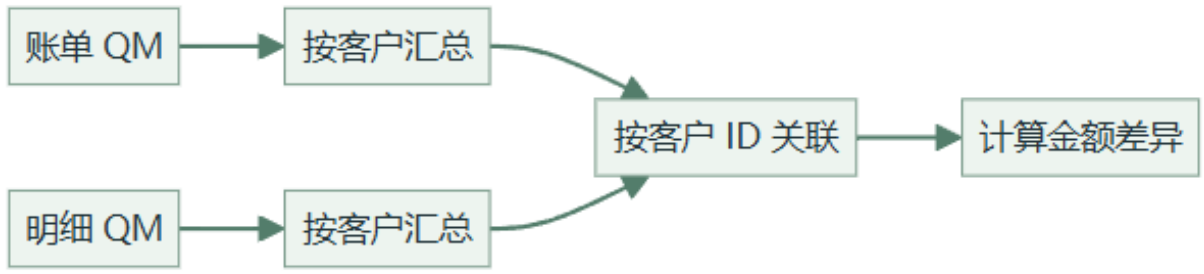
两侧都按账单日期限定 2012 年，再按客户 ID 汇总。原来一行分别代表一张账单、一条明细；汇总后，**两侧都变成一位客户一行**，再按客户键关联。

第十四章将聚合 JOIN 规则预先定义在 QM 中；本章临时组合两个 QM 的查询结果，汇总粒度由这次分析决定。

第二步：分别汇总，再关联

准备好后发送：

> 用 Compose 分别按客户汇总 2012 年账单和明细，关联后核对金额差异。



查询与建模流程

实际执行的完整脚本如下：

```
const bills = dsl({
    model: 'ChinookInvoiceQueryModel',
    columns: ['customerId as bill_customer_id', 'sum(invoiceAmount) as bill_amount'],
    slice: [{ field: 'invoiceDate', op: '[)', value: ['2012-01-01', '2013-01-01'] }],
    groupBy: ['customerId']
});

const lines = dsl({
    model: 'ChinookInvoiceLineQueryModel',
    columns: ['invoice$customerId as line_customer_id', 'sum(lineAmount) as line_amount'],
    slice: [{ field: 'invoice$invoiceDate', op: '[)', value: ['2012-01-01', '2013-01-01'] }],
    groupBy: ['invoice$customerId']
});

const joined = bills.join(lines, 'left', [
    { left: 'bill_customer_id', op: '=', right: 'line_customer_id' }
]);

const compared = joined.query({
    columns: ['bill_customer_id', 'line_customer_id', 'bill_amount', 'line_amount',
        'ROUND(bill_amount - line_amount, 2) as amount_difference'],
    orderBy: ['bill_customer_id'], limit: 100
});

return { plans: compared };
```

bills、lines 分别产生客户金额合计，joined 以账单侧为左侧关联。两侧客户 ID 都保留，便于判断有没有匹配到另一侧；不同别名也让金额来源更清楚。

差异列在关联后用列内表达式计算。本例按两位小数核对，先汇总再对差额 ROUND(..., 2)，避免浮点求和的小尾数被当成金额差异；实际项目按约定精度处理。

第三步：先核对金额，再查未匹配项

Codex 先校验、预览，再执行。本次两侧客户键均唯一，关联后完整返回 47 行。下面按两位小数展示：

客户 ID	账单金额	明细金额	差异
1	15.84	15.84	0.00
2	0.99	0.99	0.00
3	5.94	5.94	0.00
全部 47 位合计	477.53	477.53	0.00

再发送：

> 查出金额有差异的客户，再分别检查两侧有没有未匹配客户。

Codex 在关联结果上继续筛选，分别执行三项检查：

检查	方法	本次结果
已匹配但金额不同	amount_difference != 0	0 行
账单侧有、明细侧未匹配	保持账单为左侧，查 line_customer_id 为空	0 行
明细侧有、账单侧未匹配	改以明细为左侧，查 bill_customer_id 为空	0 行

金额差异与未匹配项要分开查。未匹配时，另一侧金额和差额会为空，单查“差额不为零”会漏掉它们。这里三项均为空，才说明本次客户粒度的核对一致。

请求合法，期间也可能选错

为了检查这点，我们只把明细输入改为 2012 年十一月，账单仍取全年，原始数据不变。请求通过校验，核对却返回 **5 位已匹配但金额不同的客户，以及 40 位未匹配到明细的客户**。

这些差异来自人为错配期间，应该先检查两侧日期和业务范围。引擎按请求完成计算，核对所要求的共同口径仍要由 AI 和使用者确认。

本例两侧的客户键在汇总后唯一；换成“客户 × 月份”或“客户 × 币种”时，关联键也要带上相应字段。仅有金额相等，还不足以证明分组对象一致。

本章 Foggy 知识点

- **聚合后组合：** 两个 QM 各自筛选、汇总，再通过 Compose 关联。
- **结果粒度：** 按共同业务键对齐，保留两侧标识，确认一行代表什么。
- **派生计算与核对：** 关联后计算差额，分别检查金额差异与双向未匹配记录。

本章 10 项执行已与原始账单、明细及 Decimal 独立核对。完整请求、边界变体与复跑方式见实操记录。下一章先圈选金额最高的客户，再分析他们的账单。

第 28 章 先圈选对象，再继续分析

上一章分别汇总两个 QM，再关联核对。本章换一个问题：**先找出 2012 年账单金额最高的三位客户，再查看他们当年的全部账单。**

先选客户，再取账单，两个阶段的粒度不同。这里的“前三”限制客户数，最终账单可以超过三张。

继续使用 chinook-composition，Launcher 0.1.42、CLI 0.1.25，金额沿用 Chinook 样例原单位。

第一步：确认模型，先不查询

在原项目和会话中发送：

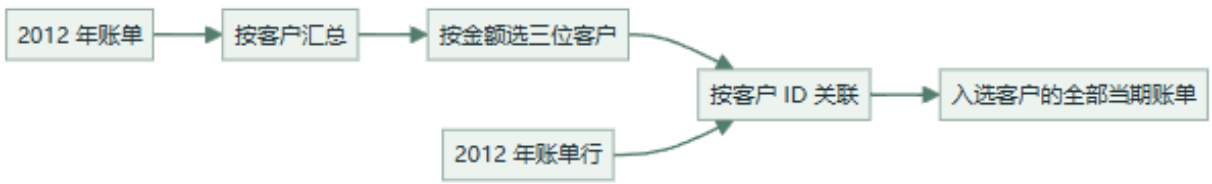
> 复用账单模型，确认字段，先不查询。

本章不新增 TM/QM。账单 QM已有客户 ID、账单 ID、日期与金额，Codex 实时读取查询说明即可。若模型还未加载，先完成校验与加载，再发送查询。

第二步：先选三位客户，再关联账单

发送：

> 用 Compose 找出 2012 年账单金额最高的三位客户，再列出他们当年的全部账单。



查询与建模流程

实际执行的完整脚本如下：

```
const totals = dsl({
    model: 'ChinookInvoiceQueryModel',
    columns: ['customerId as customer_id', 'sum(invoiceAmount) as selection_amount'],
    slice: [{ field: 'invoiceDate', op: '[)', value: ['2012-01-01', '2013-01-01'] }],
    groupBy: ['customerId']
});

const selected = totals.query({
    columns: ['customer_id', 'selection_amount'],
    orderBy: ['-selection_amount', 'customer_id'], limit: 3
});

const bills = dsl({
    model: 'ChinookInvoiceQueryModel',
    columns: ['customerId as bill_customer_id', 'invoiceId as invoice_id',
```

```
        'invoiceDate as invoice_date', 'invoiceAmount as bill_amount'],
    slice: [{ field: 'invoiceDate', op: '[)', value: ['2012-01-01', '2013-01-01'] }]
});

const result = selected.join(bills, 'inner', [
    { left: 'customer_id', op: '=', right: 'bill_customer_id' }
]).query({
    columns: ['customer_id', 'invoice_id', 'invoice_date', 'bill_amount'],
    orderBy: ['customer_id', 'invoice_date', 'invoice_id'], limit: 100
});

return { plans: result };
```

totals 按客户汇总；selected 按金额降序取三位，金额相同时按客户 ID 升序确定顺序。本例取固定三位，不额外纳入并列客户。

bills 保持一行一张账单，再用入选客户键关联。客户集合是查询计划的输出，AI 不必抄下三个 ID，再拼一个固定的 in 条件。

两处 limit 的作用不同：selected 的 3 决定入选客户；最终的 100 限制返回账单行。本例只有 7 行，未被截断；实际超过上限时，应分页取齐后再核对。若把最后的限制改为 3，只会留下前三行账单，不能说取回了三位客户的全部账单。

第三步：核对客户集合与账单

Codex 先校验、预览，再执行。预览中，客户汇总阶段保留了金额排序与 LIMIT 3，之后才关联账单。独立核对得到：

入选顺序	客户 ID	2012 年账单金额	账单张数
1	26	25.84	2
2	34	24.77	2
3	13	24.75	3
合计	-	75.36	7

最终返回账单 ID、日期、金额，按客户 ID 和日期排列，所以展示顺序不必与排名顺序相同。核对时既要检查入选的客户，也要检查他们的账单是否完整、是否重复。

脚本没有把 selection_amount 带到最终明细。它是客户年度合计，关联后会随每张账单重复；若需要一起展示，也不能再按账单行累加它。明细合计应累计 bill_amount。

直接查“金额最高的三张账单”是另一个问题。本次会得到客户 26、5、43，与前三客户的集合不同。

圈选期间与分析期间可以不同

再发送：

> 仍用这三位客户，改看 2013 年的账单。

保留 totals 和 selected 的 2012 年范围，只把 bills 的日期改为 [2013-01-01, 2014-01-01)。这样选人的依据没有变，后续分析的期间变了。

实际请求只返回客户 26 的账单 354，金额 8.91。使用 inner 时，客户 13、34 没有匹配账单，不出现在结果中；需要保留全部三位客户时，改为LEFT JOIN，未匹配账单的字段为空，不补成零金额账单。

这里描述的是样例中的记录。账单数据最晚到 2013 年 12 月 22 日，不能把样本内未匹配直接解释为现实中整年没有消费。也不要重新按 2013 年排名，那会换掉要观察的客户集合。

本章 Foggy 知识点

- **阶段 TopN：** 先按业务指标选定对象，再关联这些对象的记录。
- **范围传递：** 用阶段输出的业务键连接，分别表达圈选期间与分析期间。
- **粒度与完整性：** 客户数、账单行数及最终分页分别核对，汇总金额不随明细重复累计。

本章 10 项执行已与原始账单行及 Decimal 独立核对，包含最终截断、跨期间、保留未匹配客户和空集合。完整请求与复跑方式见实操记录。至此，Compose 系列完成了“阶段筛选、分别汇总后关联、先圈选再分析”三个常见用法。

第 29 章 从一份 Excel 开始问数

前面分析的数据已经在数据库里。这次从一份规范 Excel 开始，回答“有多少张账单、总金额多少、哪些客户金额最高”。表头混乱和多份文件合并留到后面。

第一步：准备文件，先不查询

下载 ChinookInvoices2012.xlsx，放到自己的练习目录。文件来自 Chinook v1.4.5 的 2012 年账单，只有一张“账单”工作表：

列	本例含义
账单编号	一张账单一个编号，不重复
客户编号	原 CustomerId 补足五位的教学文本编码，例如 00026，保留前导零
账单日期	原账单日期，不做时区转换
账单国家	账单地址国家
账单金额	沿用样例原单位，币种未指定，不做分元换算

换成真实文件时，也要先确认编号、日期含义和金额单位，不能只凭列名猜测。

在已准备 Foggy Skill 的项目中发送：

> 确认这份 Excel 的字段和单位，接入 Foggy，先不查询。

Codex 怎样完成接入

本系列使用 Launcher 0.1.42、CLI 0.1.25，文件接入路径如下：



查询与建模流程

Python 配合 openpyxl 读取文件，Foggy 负责语义模型与查询。此版本采用这条组合路线，未使用原生文件上传入口。

Codex 大致按下面的顺序处理：

步骤	处理内容	读者应看到什么
看文件	读取工作表、表头、类型与样例，检查编号重复和缺值	一行代表一张账单；编号保留为文本，日期和金额类型明确
准备副本	用解析脚本将固定值加载到独立 SQLite，记录原文件摘要和行号	原 Excel 保留，数据表为 file_invoices，错误行明确报出
接入数据源	用 CLI 注册、测试并绑定命名数据源	连接成功，能检查到实际表结构
建模与加载	写 TM/QM，校验后加载，再实时获取查询说明	字段已公开，模型可以查询

配套项目为 lab/files，服务端口 **18368**，本章 Namespace 为 file-invoices。后续案例各用独立 Namespace，前面章节的 18067 服务保留。读者使用自己的路径和实际服务地址即可。

FileInvoiceModel.tm映射物理字段，FileInvoiceQueryModel.qm公开业务字段和指标。后续 DSL 使用 customerCode、invoiceAmount 等语义字段名；Markdown 由引擎根据已加载代码实时生成，无须提前手写。

配套解析器只适配本例，换文件时要调整已确认的表头和类型规则。这里建立的是**普通持久语义模型**，重启后仍保留，需要管理重载与清理；完整能力边界见配套记录。

第二步：查询，再核对文件

接入完成后发送：

> 统计账单总数和总金额，再列出金额最高的三位客户。

Codex 通过 CLI 先校验，再执行。账单总数与金额的实际 DSL为：

```
{
  "columns": ["sum(invoiceAmount) as amount", "count(invoiceId) as bills"],
  "limit": 1
}
```

本次返回 **83 张账单，总金额 477.53**。再按客户编号汇总、金额降序排列，得到：

客户编号	账单金额
00026	25.84
00034	24.77
00013	24.75

结果已与文件逐行核对，并对回原始账单；前导零保留，按账单国家汇总的 20 组结果也一致。执行成功后，还要确认记录和金额与源文件对得上。

文件更新后，还要重新加载

本例按完整文件快照替换分析账单表，重复加载后数量和金额不变。文件更新时先重载数据，再查询核对；只刷新 QM 不会重新读取 Excel。多文件追加和更正留到第 31 章。

本章 Foggy 知识点

- **文件与引擎的分工：** AI 配合解析工具准备数据，再通过公开 CLI 接入数据源和语义模型。
- **字段与来源：** 将文件列映射为语义字段，保留编号类型、单位、原文件及行号。
- **加载与核对：** 区分数据重载和模型刷新，查询后核对源文件，而非只检查请求是否成功。

规范 Excel 与 CSV 的接入、查询和重复加载已经实际验证；源码、响应及复跑方法见实操记录。下一章再处理一张表里混有标题、小计和不同格式的数据。

第 30 章 不规范的 Excel 怎么整理

上一章第一行是表头，一行一张账单。这次的报表夹有标题、空行和合计，日期与金额单位也不统一，需要先整理。

文件基于同一批 Chinook 账单改编。为演示换算，本章将金额标为**教学元 / 分**，不代表 Chinook 原始币种；另加三条待确认的教学记录。

第一步：先看文件，确认规则

发送：

> 检查报表，列出需要确认的整理规则，先别导入。

Codex 检查工作表、表头位置、单元格类型和样例。本例确认下面的规则：

发现	处理约定
前两行是标题和空行	第三行才是表头，数据从第四行开始
数据区夹有空行和“合计”	按已确认标记排除，保留排除记录；合计不是一张账单
编号有文本、空格和数字格式	去首尾空格；已确认编码固定五位，才可补足前导零
日期有真实日期、年/月/日文本	按明确格式统一；03/04/2012 无法确定月日顺序，待确认
金额混有元、分	分除以 100 转为元；未知单位待确认
金额含公式	本例只接收固定值；公式结果需确认后再提交

没有“五位编码”的约定，数字 26 原来是不是 00026，AI 也猜不回来。公式则要区分公式文本、缓存值与重算结果；本例读到缓存后，仍按只接收固定值的约定保留待确认。

第二步：整理可确认部分，先不查询

规则明确后发送：

> 先接入已确认的记录，待确认的单独保留，先不查询。

这是对**部分接入**的明确选择；默认检查遇到待确认记录会停止。

Codex 使用配套整理代码，保留原文件，将转换后的数据写入独立分析副本，并保存文件名、原行号、原值和转换结果。本次整理报告为：

分类	数量
可确认账单	83

分类	数量
待确认记录	3
排除的空行	1
排除的显示合计	1

待确认条目和转换过程见整理报告。本章 Namespace 为 file-cleaning，保留 sourceRow 原行号，并补充 sourceFile 文件名；源码见 TM 和 QM。入库金额已统一为教学元，查询时不再换算。

第三步：核对金额与来源

模型校验、加载后，Codex 实时读取查询说明，再发送：

> 核对已接入的数量和金额，抽查原文件行。

CLI 校验、执行后得到 **83 张、477.53 元**。明细和国家汇总均与原始账单一致，前导零保留，每条记录能对回报表行。

最终回答要注明“仅统计已确认部分，另有 3 条未计入”。这个范围也写在 QM 说明中，避免把部分结果说成整份报表的完整结果。

本章 Foggy 知识点

- **准备层与语义层：** 文件格式、编号和单位先按确认规则整理，Foggy 查询规范化后的字段。
- **查询上下文：** 单位与接入范围写进语义说明，Markdown 按已加载代码实时获取。
- **来源追溯：** 公开文件与行号字段，查询结果能回到原记录。

本章 4 项语义查询及默认阻止部分接入的检查已实测，见系列实操。下一章处理多份文件之间的重复、追加和更正。

第 31 章 多份文件如何合并与更新

一份文件整理好了，接下来常见的是“每月再发一份”。有的文件重发旧记录，有的改了列名，还有人把更正后的金额单独发来。全部追加，数据迟早翻倍。

本章用十二份月度 CSV 和更新清单继续练习。重叠、新增和更正均为教学改编，金额沿用本系列的教学元。

第一步：确认怎样认定同一条记录

发送：

> 检查重复和更正，列出合并规则，先别导入。

本例所有文件来自同一套账单系统，账单编号在这个范围内唯一；来自多个系统时，通常还要带上来源标识，不能只碰巧编号相同就当作一张账单。

本次采用的规则是：

情况	处理
同编号、同内容	同一事实重复出现，只保留一次，记下所有来源
新编号	按明确的追加批次加入
同编号、不同内容	先报冲突；有已确认的更正声明和更高版本，才替换旧内容
某份文件改了列名	按已确认的列映射整理，缺必需列时停止
月度文件缺失	对照接收清单检查，不当作无业务或删除；撤销记录须另有明确规则

下载、复制会改变文件时间，不能据此决定更正优先级。文件齐全后，还要对照源业务控制数核对记录是否完整。

第二步：按清单合并，先不查询

确认规则后发送：

> 按规则合并，保留处理记录，先不查询。

Codex 使用合并代码读取文件清单、批次类型和版本，检查后再写入分析副本。二月文件的列名变化按映射处理，重发的一月账单只保留一次。文件名表示导出批次，统计仍以账单日期为准。

本例按清单重建完整快照，重复运行结果不变，尚未采用文件级增量加载。

Namespace 为 `file-monthly`，使用账单 TM / QM。字段含义不变时，无须每收到一个文件就重新建一套语义模型。

第三步：核对每批变更

发送：

> 核对合并后的账单数、金额，以及新增和更正的记录。

本次逐阶段实际执行得到：

输入清单	账单数	金额
十二份月度文件，含一条重复	83	477.53
加入一张 10 元的教学账单	84	487.53
原账单 250 从 13.86 更正为 14.86	84	488.53
同一最终清单再次运行	84	488.53

更正增加了 1 元，没有增加账单张数；新增账单同时改变张数和金额。原始文件仍保留，更正来源记为 correction.csv，可以通过语义字段追溯。

另用“同键金额不同但没有更正声明”和“缺少六月文件”验证，两者均在写入前被拒绝，原分析副本不变。

本章 Foggy 知识点

- **稳定语义入口：** 文件批次变化，查询仍使用同一组业务字段。
- **数据重载：** 数据副本更新后直接核对；字段定义变更时才同时调整模型。
- **可追溯查询：** 用账单键、源文件和版本记录解释结果变化。

本章 7 项查询通过，84 条更新后明细已独立核对，两项准备层拒绝也已验证，见实操。下一章把同样的口径确认方法用于遗留数据库。

第 32 章 遗留数据库如何建立语义层

文件有表头混乱，旧系统也可能有含义不明的字段、过时金额和状态码。让 AI 学会 SQL，不代表它知道 v0、v1 哪个才是现在使用的金额。

本章用同一批账单改编出教学旧库，配有字段说明。状态、历史字段及两张额外账单均为教学设定，原 Chinook 数据没有修改。

第一步：解释旧字段，先不建模

发送：

> 检查这份旧库和字段说明，整理账单口径，先不建模。

Codex 通过 CLI 查看表结构、键和少量样例，再结合提供的业务说明确认：

字段或对象	本例解释
id0	账单编号
c0	旧客户编码，通过唯一映射表转换为五位客户编号
d0	账单日期
v0	当前金额，存分，转换为教学元时除以 100
v1	人为设置的旧口径金额，不再使用
s0	V 有效、X 作废、D 草稿；本次只分析有效账单

有 JPA / MyBatis 源码、旧报表或字典时，可以让 AI 协助追溯，再由了解业务的人确认。不能只凭几条样例，就认定字段已废弃或两个编码属于同一客户。

第二步：把确认过的规则写进语义模型

发送：

> 按确认口径建立有效账单分析，校验并加载，先不查询。

本章使用 Namespace `file-legacy`。LegacyBillModel.tm通过 `viewSql` 定义模型内视图，负责客户映射、状态范围和分转元。以下是该配置中的 SQL 内容：

```
SELECT b.id0 AS invoice_id, m.customer_code,
       b.d0 AS invoice_date, b.n0 AS billing_country,
       b.v0 / 100.0 AS invoice_amount
FROM legacy_bill b
JOIN legacy_customer_map m ON b.c0 = m.old_code
WHERE b.s0 = 'V'
```

这段 SQL 是建模人员维护的固定定义。查询时 LLM 使用 invoiceAmount、customerCode 等公开字段，不必再次判断状态码或除以 100。客户映射已核对为一对一，不能改用非唯一名称关联。

本例只配置 viewSql。不要同时填写优先级更高的 tableName，否则查询仍会指向原表。

QM公开确认后的业务字段，校验、加载后实时获取查询说明。

第三步：用普通问题检查口径是否落实

发送：

> 统计有效账单数量和金额，再按客户核对。

原教学旧表有 **85 条**：83 条有效，另有一条作废和一条草稿。语义查询实际返回 **83 张、477.53 元**，47 位客户的金额逐项与原始账单一致。

直接按作废、草稿的教学编号查询，结果为空；尝试选择未公开的 v1，在校验阶段被拒绝，没有执行。前者由固定业务范围落实，后者由查询字段边界落实。旧库文件摘要没有变化。

本章 Foggy 知识点

- **模型内视图：** 用 viewSql 组织已确认的旧字段、固定范围和关联映射。
- **统一业务表达：** 将分转元等确定性工作放到语义定义，查询只使用规范字段。
- **公开面与回归：** QM 只公开当前字段，核对过滤、金额和客户集合是否一致。

本章 3 项查询及一项错误字段拒绝通过，见实操。下一章在这套已确认的实际数据旁，再接入一份 Excel 目标。

第 33 章 系统实际与 Excel 目标一起分析

实际账单在系统里，目标却由业务人员维护在 Excel 中。要回答“完成了多少”，先要确认两边说的是同一批客户、同一期间和同一种金额。

本章复用上一章的 83 张有效账单，接入教学目标文件，不混入第 31 章另设的追加与更正。目标为教学设定，双方金额均按教学元比较。

第一步：确认对齐规则，先不查询

发送：

> 检查实际与目标，确认客户、期间和单位，先不查询。

本次确认：

对齐项	约定
主体	五位客户编号精确匹配，不按姓名猜测
粒度	目标一行一客户一年一个版本；实际按客户汇总同年账单
期间与版本	2012 年，目标版本 v1
单位	双方均为教学元，准备时确认，不在查询结果里临时凑换算
缺失与零	没有目标、目标为零、没有匹配实际分别表示

目标唯一键是“客户编号、年份、版本”。同一个键重复，应先确认哪条有效，不能直接求和，也不能任取第一条。

第二步：准备共同分析副本

发送：

> 按确认规则准备实际与目标分析，校验并加载，先不计算完成率。

0.1.42 的直接跨源 JOIN 尚有已记录的校验缺口。本例通过系统 QM 提取有效账单，与解析后的目标文件放入同一个独立 SQLite，先完成数据准备，再交给引擎查询。

这条路线使用同库分析副本。提取时另查总数，确认 83 行取齐；超过返回上限时须补齐分页，不能用截断数据计算完成率。

本章 Namespace 为 `file-target-analysis`，源码包括：

模型	内容
实际 TM / QM	系统有效账单的分析副本，保持账单粒度
目标 TM / QM	公开客户、年份、版本及目标金额

两套模型校验、加载后，再实时获取查询说明。

第三步：先汇总实际，再关联目标

发送：

> 查看 2012 年 v1 完成情况，并检查未匹配客户。

Codex 按客户汇总实际，筛出目标的年份和版本，再以目标为左侧关联。沿用前面的 Compose 用法，完整请求中完成率表达式为：

```
actual_amount / NULLIF(target_amount, 0) AS achievement
```

实际值和目标值保留，便于核对。CLI 校验、预览并执行后，按百分比展示如下：

客户编号	目标	实际	完成率
00002	0.00	0.99	目标为零，不计算
00013	20.00	24.75	123.75%
00026	30.00	25.84	86.13%
00034	25.00	24.77	99.08%
00999	10.00	未匹配	不计算

00002 的目标为零，不能作分母；00999 未匹配实际，应先检查是否无交易、编号不一致或数据缺失。本例都保留空值，不能统一解释成 0%。

核对两侧范围

只以目标为左侧，看不到目标表以外的客户。本次系统实际有 **47 位**客户，目标表有 **5 位**：4 位匹配，43 位实际客户没有目标，另有 1 位目标客户未匹配实际。

因此又分别执行实际未匹配目标和目标未匹配实际。整体实际 477.53 不能直接除以这份局部目标 85；要算整体完成率，先明确共同范围和缺失处理。

本次还故意传入一个重复目标键，准备层在写入前拒绝，原分析副本保持不变。

本章 Foggy 知识点

- **有界提取与同库准备：** 复用系统 QM 的公开字段，确认结果取齐后再生成分析副本。
- **共同粒度与范围：** 分别限定实际期间、目标年份和版本，先汇总再关联。
- **比值与覆盖率：** 用 NULLIF 处理零分母，另外核对双向未匹配，避免局部目标冒充整体范围。

本章 6 项查询及重复目标拒绝已实际验证，见系列实操。下一章把提取、准备和核对串成可反复运行的脚本。

第 34 章 让文件分析持续复跑

下一版文件来了，提取、校验、加载和回归可以自动复跑；文件版本与口径是否可信，仍需要业务确认。

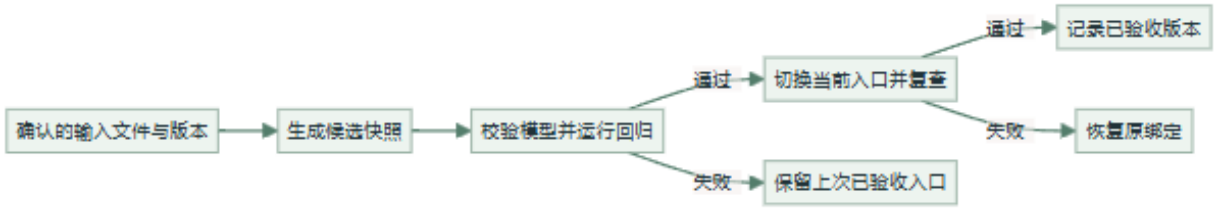
本章复用上一章的实际和目标，不增加新的分析问题。以独立的 `file-regression-current` 作为当前已验收入口，保留第 33 章的 `v1` 示例。

第一步：让 Codex 生成回归脚本

发送：

> 把接入和核对做成脚本，全部通过后再切换到新快照。

Codex 可以参考已有脚本、DSL 和验收结果，生成本例运行脚本。它做的是：



查询与建模流程

每轮生成独立数据快照与模型副本，在候选 Namespace 中验证，通过后切换当前数据源绑定，再复查。旧快照与报告保留。

第二步：固定该检查什么

本例的检查来自前几章已经确认的业务约定：

检查	本例要求
文件	客户编号类型、年份、版本、单位和目标唯一键正确
提取完整性	系统 QM 提取行数等于总数，不能因返回上限少取记录
模型	先校验、加载，再实时获取元数据
实际	83 张账单、477.53 元，与只读原始记录独立核对
目标与关联	目标合计、完成率、真实零和双向未匹配符合已审核基线
切换后	当前入口能读到刚验收的目标版本，再更新验收记录

期望值应来自审核过的输入和独立核对，不能让脚本先取结果，再把结果保存成“应该得到的答案”。正常业务变化需要审核新基线；数据错误不能靠 AI 自动改期望值放行。

第三步：换输入，再运行同一套检查

先使用 v1 目标运行。之后，本例明确批准一项教学变更：客户 00026 的目标从 30 改为 35 元，其他内容不变，版本为 v2。

发送：

> 改用已确认的 v2 目标文件，复跑并说明变化。

两轮均先在候选快照中校验、查询，再切换和复查，实际结果为：

指标	v1	v2
实际账单数	83	83
实际金额	477.53	477.53
目标合计	85.00	90.00
00026 完成率	86.13%	73.83%

变化来自目标更新，不能解释成实际业务下降。实际与目标、数据版本和统计期间应一起报告。

脚本记录输入摘要、提取请求、模型版本、数据快照、响应和验收结果，`latest-accepted.json` 指向通过全部检查的运行。

失败时，确认当前入口没有被带坏

本次又验证了两种失败：

- 把 v1 文件按 v2 版本提交，在输入校验时停止，没有切换。
- 在切换后故意模拟复查失败，脚本恢复前一数据源绑定，验收记录保持不变。

失败后再次查询，当前已验收目标合计仍为 90 元。这两项是脚本故障检查，不是新发现的引擎 BUG。

排查时先定位失败阶段：输入检查、数据准备、模型加载、查询核对，还是入口切换。

保留与清理，也要有边界

原始输入、报告和已验收快照按保留规则管理。候选 Namespace、Bundle、数据源及文件不会自动过期，清理前须确认当前入口没有引用，再通过公开管理入口清理注册和对应文件。

本章 Foggy 知识点

- **候选验证：** 在独立数据源与 Namespace 中校验模型、执行固定语义查询。
- **验收后切换：** 使用公开绑定和刷新入口，切换后再复查；失败保留或恢复上次入口。
- **版本与复跑：** 固定查询、独立基线和运行记录共同解释结果变化。

两轮成功流程各完成 8 项执行，输入拒绝和切换后恢复也已验证，完整范围见实操。文件系列至此覆盖了规范接入、整理、多文件合并、旧库映射、目标对齐和持续复跑。

第 35 章 用 SQL 表达语义查询

前面让 Codex 直接写 DSL。这次用 SQL 表达同一个问题，仍复用已有语义模型。业务用户也可以继续用自然语言提问。

本章的链路是：**SQL 表达查询意图 → Codex 转为 DSL → Foggy 校验并编译物理 SQL → 数据库计算。**
0.1.42 已核对的公开入口未发现原生语义 SQL 解析器，转换由 AI 完成，详见入口核对。

第一步：读取已有模型，先不查询

复用第32章的 `file-legacy` Namespace 与 `LegacyBillQueryModel`，仍使用文件教学服务。本章不新建模型，也不重新导入数据。

发送：

> 读取 file-legacy 的账单模型，确认字段和单位，先不查询。

Codex 实时读取查询说明，确认：

公开字段	本例含义
invoiceId	账单编号
customerCode	五位客户编码，保留前导零
invoiceDate	账单日期
billingCountry	账单地址国家
invoiceAmount	已换算为教学元的账单金额，可求和

语义模型已经限定有效账单，作废和草稿不在查询范围内。Markdown 由引擎根据已加载代码实时生成，无须提前维护。

第二步：用 SQL 表达客户排名

仍然查询“2012年金额最高的三位客户”。把下面的 SQL 一起交给 Codex：

> 把这段 SQL 转为 DSL 查询，再核对全年账单数和金额。

```
SELECT customerCode, SUM(invoiceAmount) AS amount
FROM LegacyBillQueryModel
WHERE invoiceDate >= '2012-01-01'
  AND invoiceDate < '2013-01-01'
GROUP BY customerCode
ORDER BY amount DESC, customerCode ASC
LIMIT 3;
```

FROM 后面是 QM 标识，字段来自刚读取的查询说明。这里把 QM 看作一张“虚拟业务表”，不需要在数据库中创建同名表或视图。

Codex 在当前会话将它转换为：

```
{
  "columns": ["customerCode", "sum(invoiceAmount) as amount"],
  "slice": [{"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}],
  "groupBy": ["customerCode"],
  "orderBy": ["-amount", "customerCode"],
  "limit": 3
}
```

对应关系如下：

SQL	本次转换
FROM	CLI 的模型参数 LegacyBillQueryModel
SELECT	columns，包含求和与结果别名
WHERE	slice，两个日期条件合成左闭右开区间
GROUP BY	按 customerCode 分组
ORDER BY	金额降序，同额时客户编码升序
LIMIT	返回上限 limit

CLI 提交的是这份 JSON，QM 名作为模型参数单独传入。sql query 则直接探查物理数据库，本次用 QM 名调用时返回了 no such table: LegacyBillQueryModel。

第三步：看规则有没有继续生效

Codex 通过 CLI 先提交 query validate，通过后再提交同一份 DSL 执行。Foggy 按语义模型编译，数据库计算后返回：

客户编码	金额（教学元）
00026	25.84
00034	24.77
00013	24.75

另查总量仍为 **83张、477.53教学元**，客户排名与只读原始账单逐项核对一致。

查看引擎返回的物理SQL，其模型展开部分仍包含：

```
FROM (
  SELECT b.id0 AS invoice_id, m.customer_code,
         b.d0 AS invoice_date, b.n0 AS billing_country,
```

```
        b.v0 / 100.0 AS invoice_amount
  FROM legacy_bill b
  JOIN legacy_customer_map m ON b.c0 = m.old_code
  WHERE b.s0 = 'V'
) t1
```

这段规则来自第32章的 TM，Foggy 在编译时将其展开。前面的 SQL 引用业务字段，不再重复状态条件或除以100；按作废、草稿编号查询也返回空集。

如果追问“只保留年度金额超过25的客户”，条件应是 HAVING SUM(invoiceAmount) > 25。转换时给上面的 DSL 增加 having，以下只展示新增属性：

```
{
  "having": [{"field": "amount", "op": ">", "value": 25}]
}
```

沿用上述日期与分组后，本次只返回 00026、25.84。把条件放入 slice，会变成先筛单张账单，已经换了问题。

转换后仍要检查什么

AI 可能看懂 SQL，却在转换时漏掉条件、用错别名或筛选阶段。引擎校验能检查请求是否合法，不能证明 AI 完整保留了原意。

例如要求查询旧字段 v1，Codex 应根据元数据说明它未公开。本次另将含 v1 的错误 DSL 送入校验，得到 FIELD_NOT_FOUND，没有执行。不能为了完成 SQL 请求而改查物理表。

本章 Foggy 知识点

- **语义查询视角：** SQL 使用 QM 标识与公开字段，仍需实时元数据。
- **转换与执行分工：** AI 转换、CLI 提交、Foggy 校验与编译，数据库计算。
- **核对原意：** 除数值外，还要检查日期、分组、筛选阶段与返回范围。

本章4项查询、1项字段拒绝已实测，源码与响应见实操记录。下一章将转换前后的含义与约束整理成可复跑的核对案例。

第 36 章 SQL 转成 DSL 后，怎样核对查询含义

上一章已经跑通 SQL 表达、AI 转 DSL、引擎执行的链路。这次检查：请求能执行，是否还在回答原来的问题？

仍使用 file-legacy 的 LegacyBillQueryModel，不新建模型。当前路线由 Codex 转换，Foggy 校验和编译，数据库计算。

第一步：先确定怎样算对

把本章 SQL 文件交给 Codex，发送：

> 整理这些 SQL 的含义和核对项，先不查询。

Codex 实时读取模型元数据，确认字段与单位。核对时固定同一份数据、模型版本和调用身份；本例仍按2012年有效账单、教学元统计，空集案例另行指定客户或期间。

排名和 HAVING 复用上一章已确认的请求，新增案例的预期值从只读原始账单独立计算。不能让 AI 同时写两份请求，结果相同就算互相证明。

第二步：转换后查询，再核对计算含义

清单确认后发送：

> 转成 DSL 查询，并与已确认结果核对。

同一份 DSL 先校验，再执行。本次主要看下面两种情况。

平均金额，分子和分母都要保留

SQL 表达如下：

```
SELECT SUM(invoiceAmount) AS amount, COUNT(invoiceId) AS bills,
       SUM(invoiceAmount) / NULLIF(COUNT(invoiceId), 0) AS averageAmount
FROM LegacyBillQueryModel
WHERE invoiceDate >= '2012-01-01' AND invoiceDate < '2013-01-01'
LIMIT 1;
```

这个 QM 没有固定的均额指标。Codex 按第10章的方法，在本次请求中定义临时计算字段：

```
{
  "columns": ["sum(invoiceAmount) as amount", "count(invoiceId) as bills", "averageAmount"],
  "calculatedFields": [{"name": "averageAmount", "expression": "sum(invoiceAmount) / nullif(count(invoiceId), 0)"}],
  "slice": [{"field": "invoiceDate", "op": "[)", "value": ["2012-01-01", "2013-01-01"]}],
  "limit": 1
}
```

averageAmount 是这里声明的临时名称，不是给 QM 新增字段。本次返回金额477.53、账单数83、均额5.753373…，展示为 5.75教学元。

再按账单国家分组，20组金额、张数和均额也逐项一致。如果把20个国家的均额直接平均，会得到约5.84；整体均额仍是5.75。公式看着像“求平均”，转换前后的计算范围也必须相同。

校验通过，筛选阶段仍可能写错

上一章要求“客户年度合计超过25”，正确转换是 having。本次故意构造一个错例：去掉 having，改为在 slice 中加入单张金额条件。以下只展示这个错误属性：

```
{
  "slice": [{"field": "invoiceAmount", "op": ">", "value": 25}]
}
```

完整错例仍保留2012年日期、客户分组和排序，结果却不同：

请求含义	校验	实际结果
按客户汇总后，合计超过25	通过	00026，25.84
先筛单张金额超过25，再按客户汇总	通过	空集

因为本期没有单张金额超过25的账单。这是为教学故意改错的请求，不是本次观察到的 LLM 生成失败；引擎正确执行了收到的条件，却无从知道它偏离了输入 SQL。

所以要先比较字段、日期、分组、筛选阶段和排序，再比结果。只看“执行成功”，不够。

空集、同额与错误别名也要检查

这些边界通常不需要复杂 SQL，却容易在转换或回答时被忽略。本次又验证了：

案例	实际结果与解释
查客户 00999 的账单明细	返回空数组，表示当前范围没有匹配记录
汇总没有账单的1990年	返回 bills: 0；金额与均额为 NULL，当前默认响应省略这两个键，不能说均额为0
同额客户排序	00013、00051 都是24.75；按客户编码升序，前三名取 00013。47位客户完整顺序已核对
选列别名是 amount，排序却写 totalAmount	DSL校验返回 FIELD_NOT_FOUND，没有执行

别名错误由引擎校验发现；把条件放错阶段，则要靠原意与结果核对。AI 不能仅凭名称相近就替换字段，也不能把含义不明的 WHERE amount > 25 自动改成 HAVING：这里的 amount 是聚合结果，应先确认用户要筛单张还是年度合计。

第三步：留下可复跑的核对

确认含义和结果后发送：

> 把这些已确认的用例加入回归，以后修改后复跑。

沿用已有回归方法，保存输入 SQL、转换后的 DSL、独立预期和实际响应。本章的核对脚本重放已保存的请求；它不会重新让 AI 理解 SQL 或生成 DSL。输入含义变了，预期值也需要重新审核。

本章 Foggy 知识点

- **计算字段：** 聚合表达式、分组范围与零分母处理必须保留。
- **校验边界：** 字段与别名错误可被拒绝，合法请求仍可能表达错问题。
- **语义回归：** 同时核对请求含义、数值、空值和顺序，保留独立基准。

本章7项正常查询、1项教学错例执行及1项校验拒绝均已验证，见实操记录。本系列核对的是保存的转换案例，不据此判断 SQL 或 DSL 哪种更准确；性能与持续运行留待下一系列规划。

第 37 章 查询慢在哪里：先看执行解释

查询结果正确后，才轮到性能。先判断时间花在哪里，再考虑减少扫描、缓存或预聚合，不能看到第二次更快就宣布优化成功。

本系列使用独立的 chinook-performance Namespace，沿用0.1.42文件教学服务。数据是2012年83张账单的教学副本，金额477.53教学元；数据很小，用来验证执行行为，不用于证明加速倍数。

第一步：准备独立副本

发送：

> 准备独立的账单性能练习项目，保留原数据，校验并加载模型。先不查询。

Codex通过CLI检查连接与表结构，准备源表TM和QM。模型一行一张账单，公开国家、日期、金额与张数；完整准备和复跑代码见实操记录。

第二步：先查，再解释这条查询

发送：

> 按国家统计账单金额和张数，再解释它怎样计算。

Codex实时读取元数据，先校验、执行下面的请求：

```
{
  "columns": ["country$id", "invoiceAmount", "invoiceCount"],
  "groupBy": ["country$id"],
  "orderBy": ["country$id"],
  "limit": 100
}
```

本次返回20个国家，合计83张、477.53。再通过 query explain 查看这条请求的解释与物理SQL。金额的 SUM、张数的 COUNT、国家分组和排序已经进入数据库查询；不是取回全部账单后让LLM逐条算。

解释返回 basis=RECOMPILED，表示重新编译这份请求。它有助于理解计划，不能当作此前执行的完整历史轨迹，也不等于数据库真实耗时分析。

从哪里判断还能优化

观察	下一步
返回大量无用明细	在含义不变的前提下，只选需要的字段或直接聚合
日期范围很大	先确认业务需要，再检查过滤、数据库索引和计划
高频重复的同一查询	检查是否有结果缓存及可靠的命中证据
大量重复聚合	评估预聚合的粒度、刷新成本与正确性

下推只说明某项计算交给了数据库，不保证它已经足够快。索引是否合适，需要数据库计划与真实数据量支撑；不能为了让查询更快而缩小用户要求的范围。

本章 Foggy 知识点

- **查询解释：** 针对具体问题调用 `query explain`，区分重新编译与历史执行证据。
- **计算下推：** 核对过滤、聚合和排序在何处执行。
- **先正确再优化：** 优化前后使用同一口径与独立答案，不用小样例推导性能倍数。

本章国家汇总与解释已验证。下一章先看重复查询与数据更新。

第 38 章 重复查询与数据更新：怎样判断缓存

同一个问题第二次回答更快，原因可能是连接已经建立、模型已经加载，或数据库自己的缓存。仅凭耗时，判断不了是否命中Foggy的查询结果缓存。

继续使用上一章的独立副本和 PerfSourceQueryModel。

先看证据，再谈命中

发送：

> 重复上一条国家汇总，检查结果和缓存证据。

两次国家汇总均为20组、83张、477.53。当前解释中的L1/L2执行信息为 NOT_EVALUATED，没有可据此确认的结果缓存命中记录。标准包核对也未发现查询结果缓存模块，不能把资料中其他宿主的缓存配置直接套进来。

这里的L1/L2名称也要看具体实现：不能只凭“一级、二级”推断它是本地内存、Redis或某一种查询键。配置和命中证据应一起核对。

在副本中检查数据新鲜度

发送：

> 在练习副本中把一张账单增加1元，再查同一个汇总，验证后恢复。

本例只修改独立教学表，不动原始Chinook库。仍调用同一个源表QM，结果为：

状态	汇总金额（教学元）
修改前	477.53
副本账单250增加1元	478.53
恢复后	477.53

本次源查询能读到修改后的数据，没有观察到旧结果阻断更新。这不证明所有部署都没有缓存，更不证明数据修改会自动清除任何宿主的缓存。

启用缓存前，要确认哪些事

问题	为什么要确认
允许结果落后多久	缓存过期时间需要符合业务新鲜度要求
查询键包含什么	字段、条件、分页、模型版本和身份范围都可能影响结果
数据和模型更新后怎么办	数据重载、模型刷新与缓存失效是不同动作
怎么证明命中	使用实际执行中的命中标记或统计，而不是只看“变快了”

具体启用与清理方式由公开接口和宿主配置决定。本章没有给当前DSL添加未经支持的缓存开关，也没有调用旧资料中的私有清理接口。

本章 Foggy 知识点

- **证据边界：** 重复结果、执行解释和实际缓存命中是不同结论。
- **数据新鲜度：** 用独立副本验证更新后结果，保留恢复步骤。
- **维护分工：** 数据、模型、缓存分别维护，不能互相代替。

下一章的预聚合会实际保存汇总数据，其新鲜度同样需要检查。

第 39 章 预聚合：先汇总，再保持同一口径查询

如果经常按国家汇总一张很大的账单表，可以先保存国家级汇总，再查询这些结果。它节省了重复扫描的机会，也增加了刷新和维护责任。

本章仍用83张账单演示正确性，不据此判断预聚合一定更快。

第一步：准备汇总表与映射，先不查询

发送：

> 为独立账单副本准备国家汇总和预聚合模型，校验并加载，先不查询。

本例由外部准备代码生成 perf_country_totals，20行分别保存国家、金额合计和账单数。TM声明让引擎知道这些列能替代哪些计算；声明本身不会创建或自动刷新数据表。

下面是 PerfAggregateModel.tm 中的 preAggregations 属性，其他定义见完整源码：

```
preAggregations: [{
  name: 'country_totals', tableName: 'perf_country_totals',
  dimensions: ['country'],
  dimensionProperties: { country: ['id'] },
  dimensionPropertyColumnNames: { country: { id: 'billing_country' } },
  measures: [
    { name: 'invoiceAmount', aggregation: 'sum', columnName: 'amount_sum' },
    { name: 'invoiceCount', aggregation: 'count', columnName: 'bill_count' }
  ],
  enabled: true
}]
```

本例显式声明国家ID及物理列映射。最初缺少这部分，解释返回 PREAGG_DIMENSION_PROPERTY_NOT_MATERIALIZED，保留源查询；补齐后才选中预聚合。不能仅凭配置了表名就认定已经命中。

第二步：用相同业务字段查询

模型校验、加载后实时读取说明，再发送：

> 用预聚合入口按国家统计金额和张数，与源查询核对。

查询字段仍是 country$id、invoiceAmount、invoiceCount，请求无须指定汇总表。源表入口与预聚合入口的20组结果逐项一致，均合计83张、477.53。

本次重新编译的解释选择 country_totals，路由为 PREAGG_ROLLUP。执行响应里的调试SQL仍保留源查询文本，因此不能仅凭这一段调试SQL断言最终路由；本例还用下面的数据对照验证了执行行为。

明细请求需要账单编号，国家汇总无法提供，实际返回仍来自源表。预聚合没有覆盖的粒度与字段，不能凭空还原。

第三步：检查刷新前后

发送：

> 修改副本中的一张账单，对比刷新前后的源查询和预聚合结果，最后恢复。

本次把账单250增加1教学元：

状态	源查询	预聚合查询
修改前	477.53	477.53
只更新账单，未刷新汇总表	478.53	477.53
外部准备代码重建汇总后	478.53	478.53

这次没有依靠模型刷新重算汇总，而是实际更新了汇总表。验证后源表和汇总表都恢复为477.53。原始Chinook数据始终未改。

需要实时口径时，可以先使用已核对的源查询入口，或把汇总刷新与验收纳入发布流程。本例不会自动补齐这次变更，不能只看“预聚合已启用”。

什么值得提前汇总

先选常用且口径稳定的维度组合。粒度越细，覆盖的问题可能越多，存储和维护成本也越高；按月汇总后不能再回答某一天的明细。

金额可以再求和，账单数上卷时应累加已保存的计数。平均值要保留金额合计与账单数后重新相除，不能直接平均各组均额。自动刷新、增量、水位补偿及复杂AVG路由，仍须在目标宿主中单独验证，本章只演示已测试的SUM/COUNT路径。

本章 Foggy 知识点

- **预聚合定义：** 在TM声明维度、指标和物理列，QM继续公开业务字段。
- **匹配与回退：** 先核对解释与实际结果，覆盖不了的明细返回源查询。
- **刷新责任：** 数据表的更新由准备或刷新链路完成，模型声明不能保证新鲜度。

完整源表/预聚合模型、请求及首次回退记录见实操。

第 40 章 分页与持续运行：把结果交付完整

查询成功不等于已经取齐数据。问“前20张账单”和“全部账单”是两个需求，返回上限也不是参与汇总的记录数。

本章继续使用独立性能副本，保留同一数据和模型版本。

分页要有稳定顺序

发送：

> 分页取齐全部账单，检查重复、遗漏和总金额。

Codex实时获取模型字段，选择账单ID作为唯一排序键，逐页先校验再查询：

```
{
  "columns": ["invoiceId", "invoiceAmount"],
  "orderBy": ["invoiceId"],
  "start": 0,
  "limit": 20,
  "returnTotal": true
}
```

本次将 `start` 依次设为0、20、40、60、80，实际返回 **20、20、20、20、3行**，每页报告总数83。合并后83个账单编号无重复、无遗漏，金额477.53，与原始记录一致。

同额排名还要追加唯一键排序，否则页边界可能不稳定。分页期间如果源数据变化，简单的位移分页仍可能漏行或重行；本例使用固定快照，不能扩大成实时更新库的一致性保证。

几种“限制”要分清

限制	实际含义
DSL的 limit	限制返回行数，不代表只计算这些明细
returnTotal	请求总数；核对它与明细/分组的统计对象是否相同
CLI等待超时	客户端等不到响应，不自动证明数据库任务已停止
宿主或数据库资源策略	控制可接受的扫描、执行时间和并发，需要部署侧配置与验证

本章没有通过故意制造长查询验证取消能力，也没有把CLI等待时间写成数据库执行上限。取数规模超过约定时，应先明确用途、范围和资源预算，不能无限重试或悄悄截断。

更新后，还要检查什么

发送：

> 把模型、数据和汇总更新后的检查整理成日常运行清单。

可以沿用第16、34章的回归方法，不再建另一套框架：

1. 记录数据快照、模型版本、当前绑定和统计截止时间。
2. 数据更新后核对数量与金额；模型变更后先校验、加载，再获取实时元数据。
3. 若有缓存或预聚合，分别确认失效和刷新，不把模型刷新当作数据刷新。
4. 用同一组题目检查正常结果、空集、错误字段和调用身份范围。
5. 失败时报告实际阶段和范围；切换入口前后核对，保留已验收版本。

这些检查解决技术链路的可重复性。指标是否合理、数据是否完整、业务解释是否有依据，仍需要人与业务系统协作。

本章 Foggy 知识点

- **结果边界：** 区分返回上限、总数与计算范围，明确分页顺序。
- **运行边界：** 区分客户端等待与数据库执行控制。
- **持续核对：** 数据、模型、缓存、汇总各自更新，用已审核题库回归。

本系列最终12项正常执行、2次解释已完成，见实操记录。第一阶段至此覆盖概念、AI与CLI实践、高级分析和运行维护；在线问数、FDP模型维护及Dashboard/报表留待后续阶段。