

数据库第二次课程大作业

姓名：***

学号：***

一、分析与说明

1.1 SQL 关系数据库具有强大长久的生命力的原因

当今的企业和组织需要存储、管理和处理大量的数据，以支持他们的日常业务活动。SQL 关系数据库是一种长久而广泛使用的数据存储方式，其强大的生命力得益于以下原因：

- 📖 **数据结构清晰：**关系数据库采用二维表形式存储数据，这种结构非常清晰。每个表格都有自己的列（属性）和行（元组合），使得用户可以轻松地理解和查询数据。该结构还为开发人员和管理员提供了更好的控制和管理。这种结构化数据存储和管理能力使得 SQL 关系数据库成为了企业级应用和数据管理的首选。
- 📖 **数据完整性保障：**这种数据库具有严格的数据完整性保障机制，包括实体完整性约束、参照完整性约束、用户自定义完整性约束等。通过这些约束，数据库得以确保数据的正确性和一致性。这意味着即使在多个应用程序或用户访问同一个数据库时，数据也不会出现冲突。
- 📖 **数据安全可靠：**这类数据库提供多重认证授权机制，可以为不同用户或角色分配不同的权限。这种机制确保只有经过验证的用户能够访问敏感数据，并且可以防止对数据库进行未经授权的修改。它还提供了备份和恢复机制，能够帮助管理员在系统崩溃或其他事件中快速恢复数据，免遭数据丢失的困扰，极大方便了运维团队的工作。
- 📖 **巨大的生态系统和用户社区：**由于 SQL 关系数据库存在时间较长并且广泛使用，已经发展出了巨大的生态系统。有成千上万的应用程序和工具与之兼容，这意味着开发人员可以轻松接入数据库，无需从头开始重新编写应用程序或工具。而且，这些应用程序和工具在开源社区中得到了广泛推广和开发。同时，其拥有庞大的开发者和用户社区。这些人员共同协作，致力于提供更好的开发工具和技术支持，不断改进和优化数据库的性能和功能。为 SQL 关系型数据库的发展提供了良好的用户支持和技术支持。



图 1：新型数据库生态系统架构

良好的标准化支持：SQL 是一种标准化的语言，在各种数据库中都可以得到广泛的应用。这种标准化使得开发人员易于学习和使用 SQL，同时也增加了各种工具之间的互操作性。这样，就可以在不同的数据库之间移植代码和数据，从而实现更高效的开发。降低开发和维护成本。同时避免了锁定效应，防止对于特定厂商的数据库，用户无法轻松地该厂商迁移到其他的数据库平台。

技术成熟且应用广泛：SQL 关系数据库现有技术手段已经非常成熟，已经被广泛应用于各种领域并且被证明非常可靠，包括企业、教育、医疗保健、政府等，它们可以存储和管理各种类型的数据，如客户信息、财务记录、库存、订单、日志等。这些系统具有高度的稳定性、安全性和可扩展性，能够满足企业级应用的需求。

综上所述，SQL 关系数据库的强大和长久的生命力得益于其数据结构清晰、数据完整性保障、数据安全可靠、巨大的生态系统和标准化等特点。这些特点使得 SQL 关系数据库成为企业和组织存储和管理数据的首选方案，具有广泛的应用前景和可持续的发展潜力。相信，在未来，SQL 关系型数据库的发展前景会越来越广阔。

1.2 对我国发展数据库技术的借鉴意义

SQL 关系数据库作为一种经典的数据库技术，具有强大长久的生命力，其对中国发展数据库技术的具有广而深远的借鉴意义。通过了解 SQL 关系型数据库，我们可以从中提炼出以下几个值得借鉴之处，以及借鉴的意义价值：

- **加强推动数据库标准化，助于和国际接轨：**SQL 成为了数据库领域的事实标准，被广泛认可和接受。极大增强数据库的可移植性和集成度，降

低了开发和维护成本。中国企业也应当注重标准化建设，通过借鉴 SQL 关系型数据库的数据模型、参数接口等，加快国际化进程，推行适合中国现今技术发展的可行性标准。通过与国际同步和加强自身规范，得以让中国的数据库技术乃至其他关联产业走向世界。对于提高我国相关技术产业的知名度和国际地位大有裨益。

- **推动安全性可靠性保障，助于减少极端事件：** SQL 关系数据库提供了强大的安全和数备份机制。随着中国企业信息化程度的提高，对数据库的可靠性要求逐步增大，如何防止用户隐私数据泄露？如何做到故障快速容灾？如何提升系统总体稳定性？通过借鉴 SQL 关系型数据库的各项安全性措施（如提供用户权限控制、加密、日志记录等一系列安全机制），有助于推动我国数据库产业的互联网安全发展，减少数据外泄等极端事件的发生，有效地保护数据库中的数据不被恶意攻击者获取或篡改，提高我国数据库技术的可信度。



图 2：阿里巴巴团队的数据库安全解决方案架构

- **提高数据处理效率，助于推动产业发展：** SQL 关系数据库被设计成高效的数据存储和检索系统。它使用了优秀的算法和数据结构，使得其能

够在短时间内处理非常大的数据量。中国数据库技术企业需要注重数据管理和运营，尤其是在海量数据时代，速度的提升已经成为各个数据库厂商日益竞争的主要指标。借鉴 SQL 关系型数据库的数据结构、查询机制有助于提升数据库使用效率，带动辐射的产业（如人机对话、智能向导、风控系统）的速度提升与动能提升，对于我国产业协同发展有着深远的意义。

- **丰富生态环境，提升用户体验，助于提高知名度：**“金杯银杯，不如老百姓的口碑”，SQL 之所以能广为流传，很大程度上是核心用户的宣传推广以及生态工具的完善（如 pgAdmin 等数据库管理工具），这些工具可以成为学习和借鉴的对象，以此方便用户对数据库的使用。我国数据库要加强这方面的建设与推进，借鉴 SQL 关系型数据库的生态架构与用户服务，不断丰富生态环境提升用户体验，才能更好的走向世界，走进千家万户。

综上，SQL 关系数据库具有强大的生命力和广泛的应用价值，其经验和思想对于中国发展数据库技术具有重要的借鉴意义。为中国数据库技术的发展起到了积极的推动作用。

1.3 推崇技术创新与技术标准化对我国数据库软件发展的作用

技术创新和技术标准化是推动中国数据库软件发展的重要因素，对于提高中国数据库软件的竞争力和市场地位具有至关重要的作用。笔者通过一些例证来表明技术创新和技术标准化对于数据库软件的发展作用，具体如下——

- **推动国产数据库软件产品升级和优化：**技术创新可以推动中国数据库软件不断进行功能改进和性能优化，使之不断满足企业、用户的多方面要求。例如，在云计算时代，用户需要充分利用云资源来提高效率和降低成本。因此，现代数据库需要支持云计算环境下的自动化扩容、数据备份与恢复等功能，为用户提供更完善的服务，这就需要数据库相关企业不断技术创新，优化查询机制——其中，国产数据库厂商阿里云 RDS 通过技术创新实现了产品的优化，引入了一系列高可用、自动化运维的新技术，如极速实例、增量备份、MySQL 双机热备等，极大提高用户满意度，推动产业的升级发展。
- **提高国产数据库软件产品质量和稳定性：**技术标准化可以规范中国数据库软件的开发过程和产品质量，从而迫使企业提高用户的满意度和信任度，形成规范反扑效应，间接推动国产数据库技术的升级。例如，国家标准 GB/T 15579-2008《数据库系统安全性技术要求》对数据库安全性进

行了规范，要求数据库应该具有访问控制、加密保护、审计跟踪等多种安全保障机制，以确保数据的机密性、完整性和可用性。伴随着标准的提出，越来越多的国内数据库厂商不断优化自身产品，更新迭代，逐步提高了产品质量。

- **提高国产产品竞争力和市场地位：**技术创新可以提高中国数据库软件在技术领域的竞争力，可以让中国数据库软件在技术领域取得突破性进展，从而提高其在市场上的竞争力和地位，扩大自身品牌影响力。同时，符合国际标准的产品可以更容易地进入国际市场，扩大自身品牌影响力和市场份额。例如，在 NoSQL（非关系型数据库）的领域，MongoDB 以其高性能、高可扩展性、易部署的特点迅速崛起，并成为行业中的领导者之一。MongoDB 不仅技术上取得了突破性进展，还主动参与了 NoSQL 标准制定，并努力推动标准化工作，提高了自身品牌影响力和市场份额。中国数据库厂商可以以此为鉴，加快推动标准化与技术创新。
- **改善用户体验和增加用户黏性：**技术创新和技术标准化可以使中国数据库软件在功能、性能和易用性方面更加完善，从而增加用户体验和黏性。例如，近年来大数据分析需求不断增长，用户期望通过可视化工具方便地进行数据查询和分析。阿里云系列数据库通过对产品的升级优化技术创新，不断吸收用户，提高市场份额，在国内外数据库软件市场已经居于高位。

IDC 2021年中国关系型数据库市场份额
(传统部署+公有云模式)

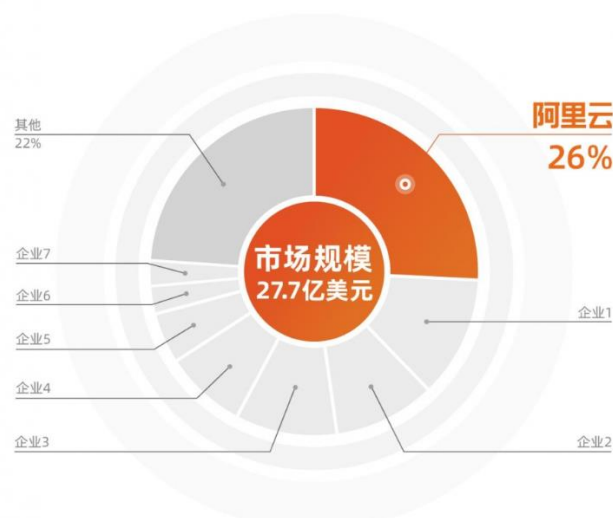


图 3：阿里云数据库中国市场份额（用户数量）居于高位

- **促进行业内交流和合作：**技术标准化可以促进行业内的交流和合作，推动中国数据库软件的共同发展。例如，数据库领域的 ISO/IEC JTC1/SC32 委员会致力于研究数据库系统的标准化问题，并尝试制定全球范围内通用的标准和规范。各家企业在制定技术标准时需要相互协商和合作，这种合作有助于推动整个行业的技术发展和进步，提高行业整体的水平和竞争力。当我国数据库厂商坚持与国际数据库标准接轨，那么一定可以走出国门，走向世界。

综上，技术创新与技术标准化对于国产数据库的发展大有裨益，它可以很好的推动我国数据库技术和产业的升级进步，更好的满足市场要求。相信在未来，国产数据库厂商在坚持数据库技术标准化以及数据库技术创新的前提下，能够更好的走向世界，甚至成为一个风向标。

二、 实践操作题

本次实践操作的目的是针对一个房产信息管理系统开发进行数据库编程。

2.1 创建数据库

本题需要创建数据库 EstateDB，SQL 语句为——

```
CREATE DATABASE EstateDB ;
```

程序运行截图如下：

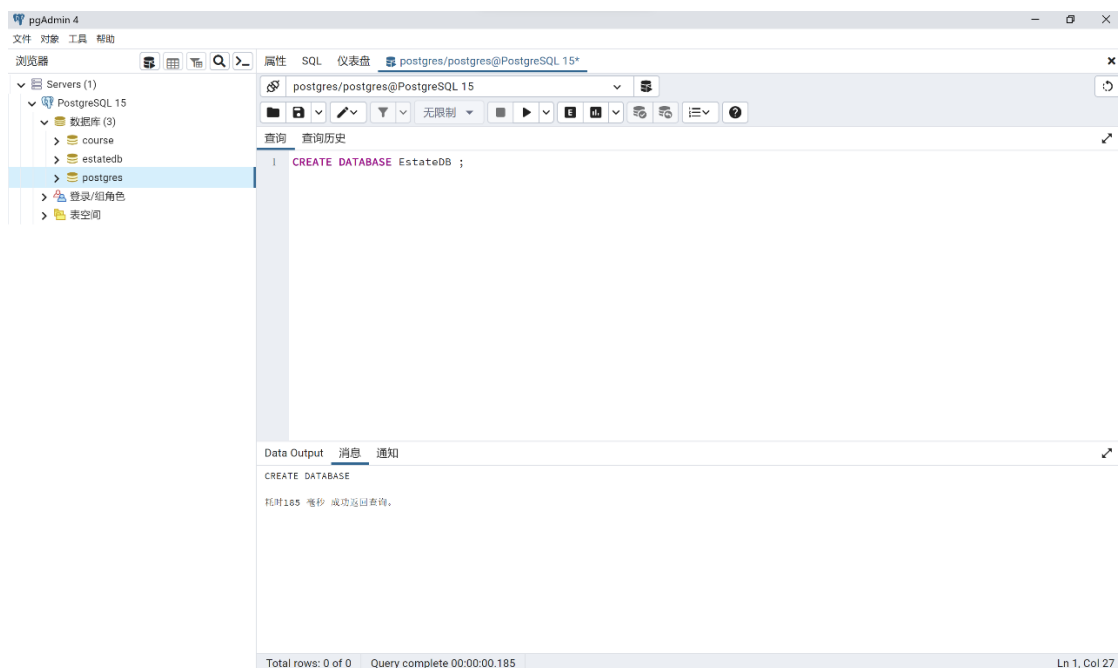


图 4：创建数据库

2.2 创建表并定义完整性约束

本题要求在数据库 EstateDB 中创建三个数据库表，并定义其完整性约束。
下面将分别展示三个表的创建——

❑ 业主表（Owner）

业主表的建表 SQL 语句为

```
CREATE TABLE Owner
(
    PersonID      Char(18)          PRIMARY Key,
    Name          Varchar(20)       NOT NULL,
    Gender        Char(2)           NOT NULL,
    Occupation     Varchar(20)       NOT NULL,
    Addr          Varchar(50)       NOT NULL,
    Tel           Varchar(11)       NOT NULL
);
```

程序运行截图为：

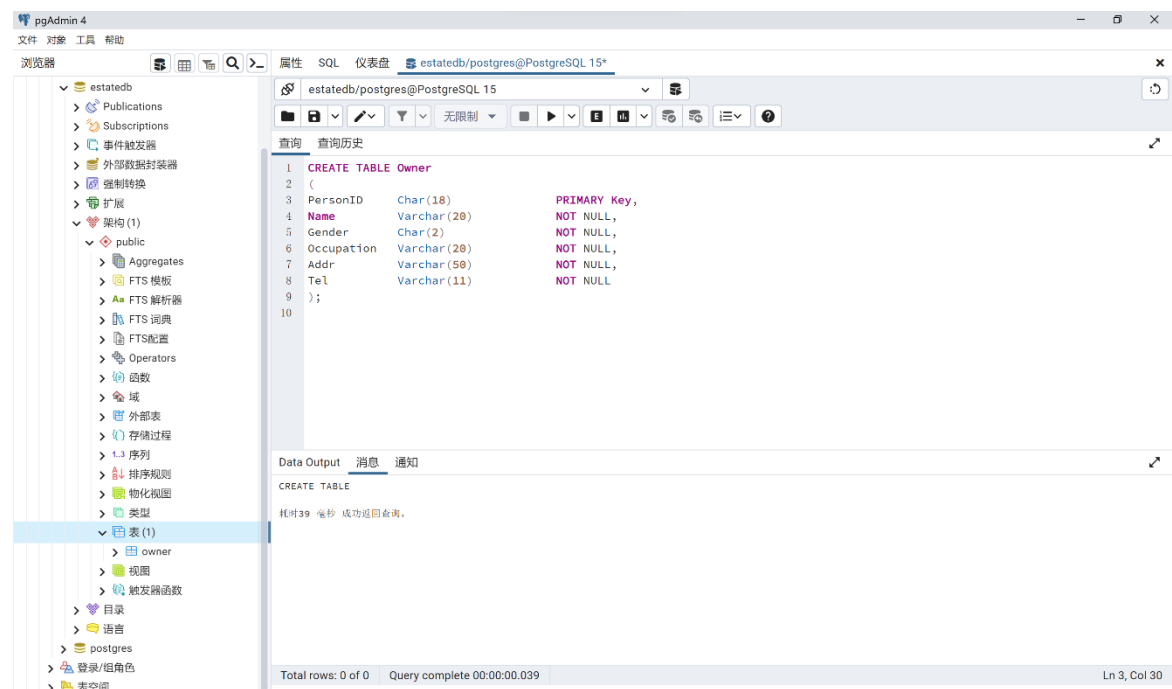


图 5：创建业主表

❑ 房产表（Estate）

房产表的建表语句为

```
CREATE TABLE Estate
(
    EstateID      Char(15)          PRIMARY Key,
    EstateName     Varchar(50)      NOT NULL,
```

```

EstateBuildName    Varchar(50)    NOT NULL,
EstateAddr         Varchar(60)    NOT NULL,
EstateCity         Varchar(60)    NOT NULL,
EstateType         Char(4)    NOT NULL CHECK(EstateType IN('
住宅','商铺','车位','别墅'))),
PropertyArea       Numeric(5,2)   NOT NULL,
UsableArea         Numeric(5,2)   NOT NULL,
CompleteDate       Date           NOT NULL,
YearLength         Int            NOT NULL DEFAULT 70,
Remark            Varchar(100)   NULL
);

```

程序运行截图为

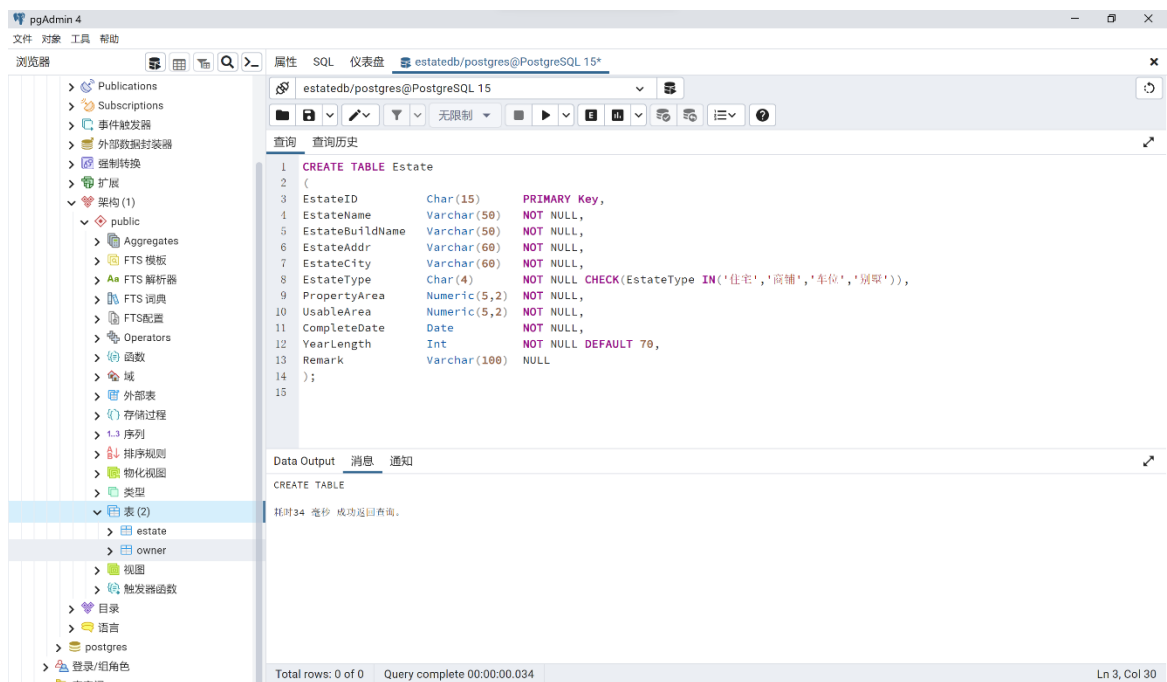


图 6：创建房产表

❑ 产权登记表（Registration）

产权登记表的建表语句为

```

CREATE TABLE Registration
(
    RegisterID      Int            PRIMARY Key,
    PersonID       Char(18)       NOT NULL,
    EstateID       Char(15)       NOT NULL,
    Price          Money          NOT NULL,
    PurchasedDate  Date           NOT NULL,
    DeliverDate    Date           NOT NULL,
    CONSTRAINT PersonID_FK FOREIGN Key(PersonID)

```



```
REFERENCES Owner(PersonID)
ON DELETE CASCADE,
CONSTRAINT EstateID_FK FOREIGN Key(EstateID)
REFERENCES Estate(EstateID)
ON DELETE CASCADE
);
```

程序运行截图为

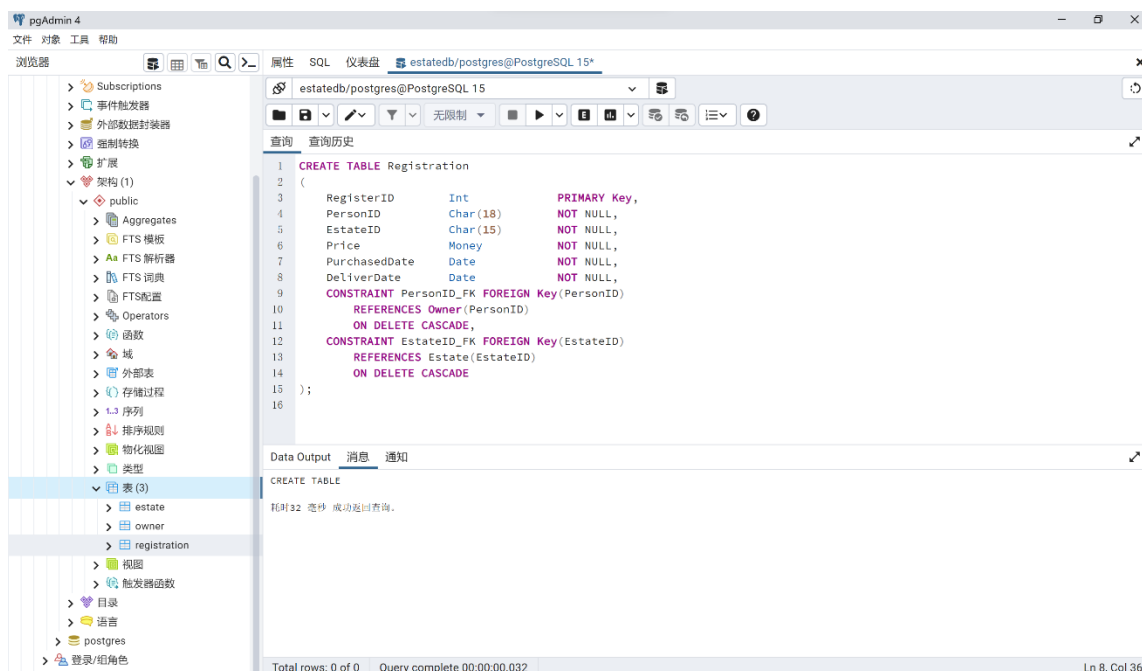


图 7：创建产权登记表

2.3 添加样本数据

本题要求编写并运行 SQL 语句，在上述三个数据库表中添加数据。借助虚假数据提供平台 Mock，笔者模拟了一系列虚拟的数据。下面是三个表的数据添加过程——

❑ 业主表（Owner）

对业主表的数据插入 SQL 语句如下，这里我们使用单条批量插入语句。

```
INSERT INTO Owner
VALUES
('360121199001011234', '张三', '男', '工程师', '江西省南昌市
东湖区八一大道 123 号', '13811112222'),
('110101198702040876', '李四', '女', '医生', '北京市海淀区西
三环北路 12 号院 5 号楼 1 单元 501 室', '13922223333'),
('440301198611205678', '王五', '男', '经理', '广东省深圳市罗
湖区田贝四路 567 号华润置地大厦 23 层 2303 室', '13733334444'),
```

```
( '310101199810145432', '赵六', '女', '教师', '上海市浦东新区张江镇碧波路 789 弄 56 号 405 室', '13644445555'),
( '500101198009263210', '钱七', '男', '律师', '重庆市渝北区翠云街道清波社区绿水小区 2 幢 3 单元 1203 室', '13555556666'),
( '610101200112030987', '孙八', '女', '医生', '陕西省西安市雁塔区高新南一路 88 号电子城 C 座 2201 室', '13466667777'),
( '320101199505024563', '周九', '男', '工人', '江苏省南京市鼓楼区中山路 399 号金鹰国际购物中心 10 层 1003 室', '13377778888');
```

程序运行截图为

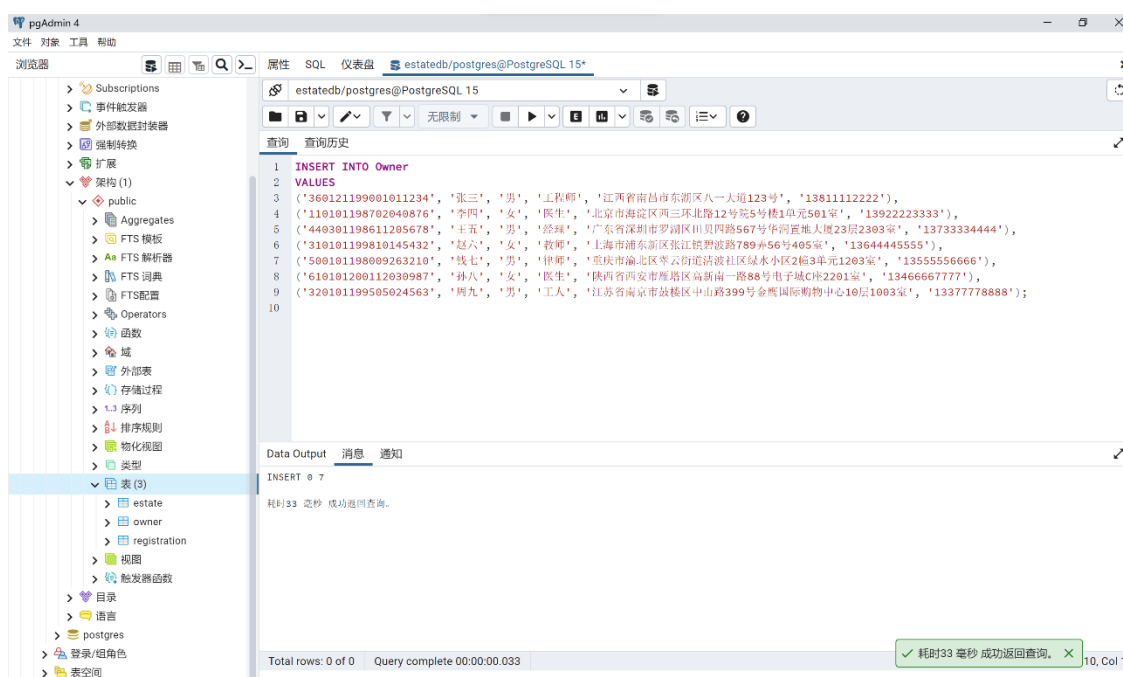


图 8：向业主表插入数据

□ 房产表（Estate）

对房产表的数据插入 SQL 语句如下，这里我们使用多条插入语句。

```
INSERT INTO Estate VALUES ('10001', '翠湖公园小区', '翠湖大厦', '南京市玄武区龙蟠中路 58 号', '南京市', '商铺', 72.50, 63.00, '2023-11-15', 95, null);
INSERT INTO Estate VALUES ('10002', '东方之门', '万达广场', '深圳市福田区红荔路 2018 号', '深圳市', '别墅', 320.00, 280.00, '2023-02-15', 99, '山景房');
INSERT INTO Estate VALUES ('10003', '天鹅湖花园', '金茂府', '上海市浦东区莫干山路 1 号', '上海市', '车位', 10.00, 9.00, '2025-05-10', 75, null);
INSERT INTO Estate VALUES ('10004', '世贸中心', '环球金融中心', '北京市朝阳区建国路 88 号', '北京市', '住宅', 80.00, 350.00, '2022-12-05', 80, null);
```

```

INSERT INTO Estate VALUES ('10005', '绿地中心', '绿地商场',
'上海市徐汇区龙华中路 666 号', '上海市', '住宅', 85.00, 72.00,
'2023-12-22', 60, null);

INSERT INTO Estate VALUES ('10006', '虹桥枢纽', '永旺梦乐城',
', '上海市长宁区遵义路 868 号', '上海市', '住宅', 92.00, 80.00,
'2024-01-01', 75, '地铁沿线');

INSERT INTO Estate VALUES ('10007', '漓江明珠广场', '漓江明珠商城',
'桂林市秀峰区解放中路 17 号', '桂林市', '商铺', 78.00,
69.00, '2023-12-25', 92, '旅游景点附近');

INSERT INTO Estate VALUES ('10008', '静安公寓', '鼎信大厦',
'上海市静安区南京西路 1188 号', '上海市', '住宅', 95.00, 80.00,
'2024-04-01', 70, '近地铁站');

INSERT INTO Estate VALUES ('10009', '浦东太阳城', '裕景花苑',
', '上海市浦东新区张杨路 1688 号', '上海市', '住宅', 125.00,
110.00, '2025-01-10', 95, null);

INSERT INTO Estate VALUES ('10010', '朝阳建国门', '希尔顿公寓',
'北京市朝阳区建国路 1 号', '北京市', '住宅', 85.00, 75.00,
'2023-12-15', 93, null);

INSERT INTO Estate VALUES ('10011', '海淀中关村', '清华科技园',
'北京市海淀区中关村大街 1 号', '北京市', '住宅', 88.50,
75.00, '2024-07-01', 75, null);

INSERT INTO Estate VALUES ('10012', '中山广场', '中山商城',
'广州市越秀区中山五路 38 号', '广州市', '商铺', 110.00, 100.00,
'2024-08-01', 80, null);

INSERT INTO Estate VALUES ('10013', '新首钢广场', '北京首钢社区',
'北京市朝阳区首钢北三路 18 号', '北京市', '住宅', 110.00,
100.00, '2024-08-01', 50, null);

INSERT INTO Estate VALUES ('10014', '太阳公馆', '成都市成华区',
'成都市成华区建设巷 18 号', '成都市', '住宅', 100.00, 90.00,
'2023-07-11', 80, null);

```

共计插入 14 条房产信息，程序运行截图见下一页。

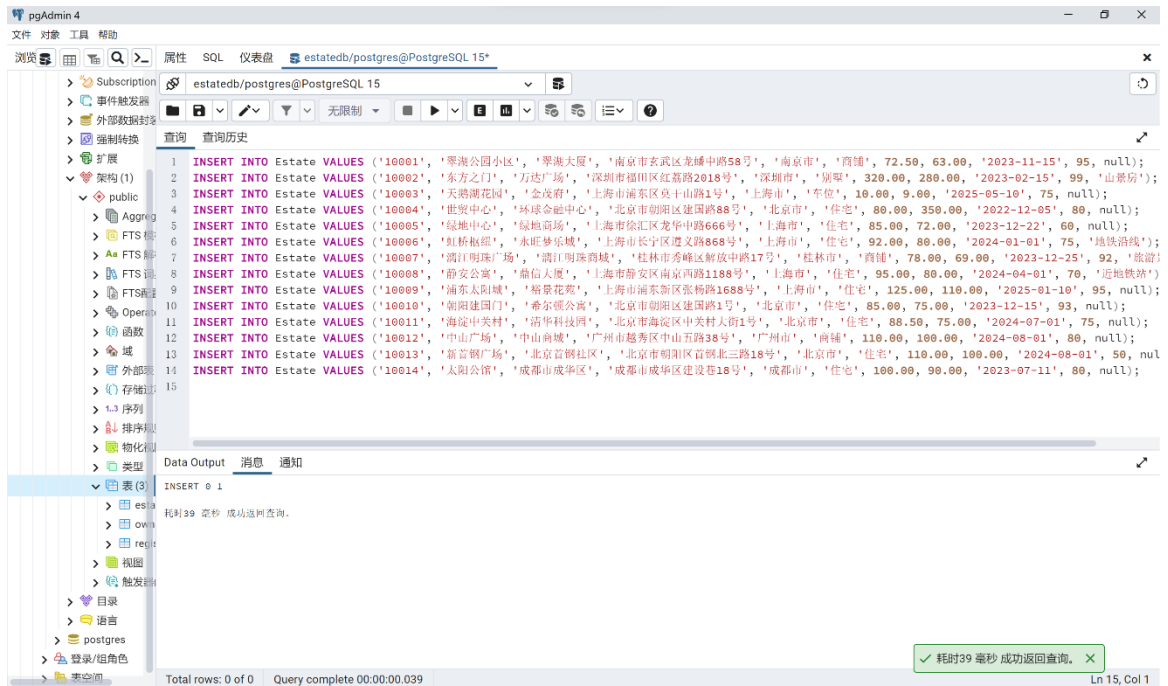


图 9：向房产表插入数据

□ 产权登记表（Registration）

对房产表的数据插入 SQL 语句如下，这里我们使用单条批量插入语句。

```
INSERT INTO Registration VALUES
(20001, '360121199001011234', '10006', 730000, '2023-11-16', '2024-01-15'),
(20002, '360121199001011234', '10008', 850000, '2023-04-02', '2024-06-01'),
(20003, '110101198702040876', '10010', 760000, '2023-12-16', '2024-02-15'),
(20004, '440301198611205678', '10007', 800000, '2023-12-26', '2024-02-25'),
(20005, '310101199810145432', '10009', 1125000, '2023-01-11', '2025-03-10'),
(20006, '500101198009263210', '10004', 3400000, '2022-12-06', '2023-02-05'),
(20007, '610101200112030987', '10002', 2720000, '2023-02-16', '2023-04-15'),
(20008, '320101199505024563', '10001', 555000, '2023-11-17', '2024-01-16'),
(20009, '360121199001011234', '10003', 4100000, '2023-05-11', '2025-07-10'),
(20010, '110101198702040876', '10011', 675000, '2023-07-02', '2024-09-01'),
(20011, '110101198702040876', '10013', 605000, '2023-11-13', '2024-08-11'),
```

```
(20012, '360121199001011234', '10014', 830000, '2023-11-16', '2024-01-15');
```

程序运行截图为

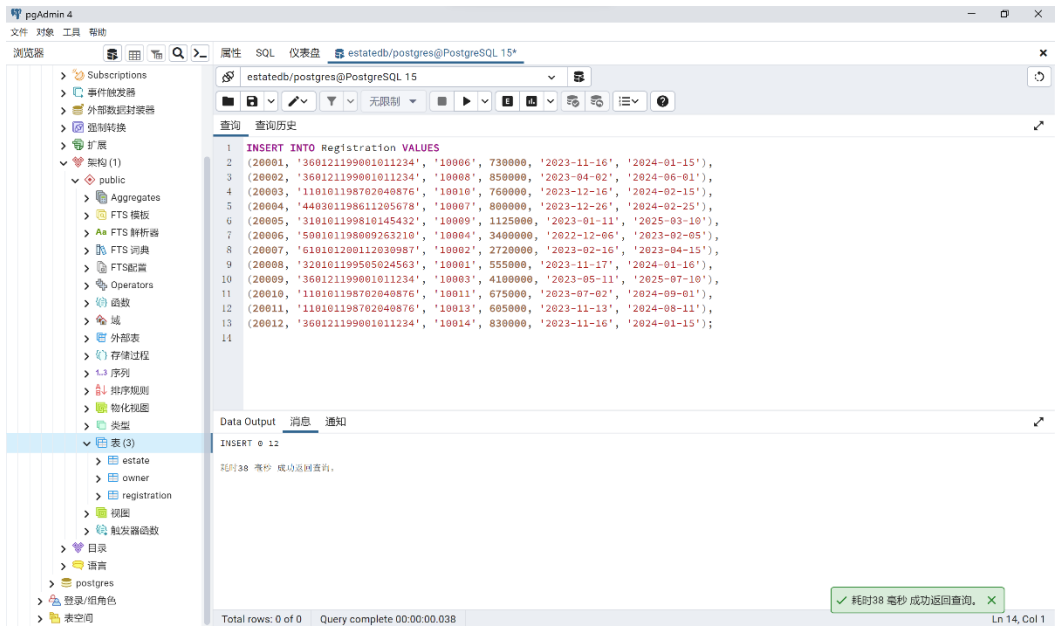


图 10：向产权登记表插入信息

2.4 查询类别为“商铺”的房产信息

查询类别为商铺的 SQL 语句为

```
SELECT * from Estate
WHERE EstateType='商铺'
```

查询到 3 条结果，与预期结果相符合：

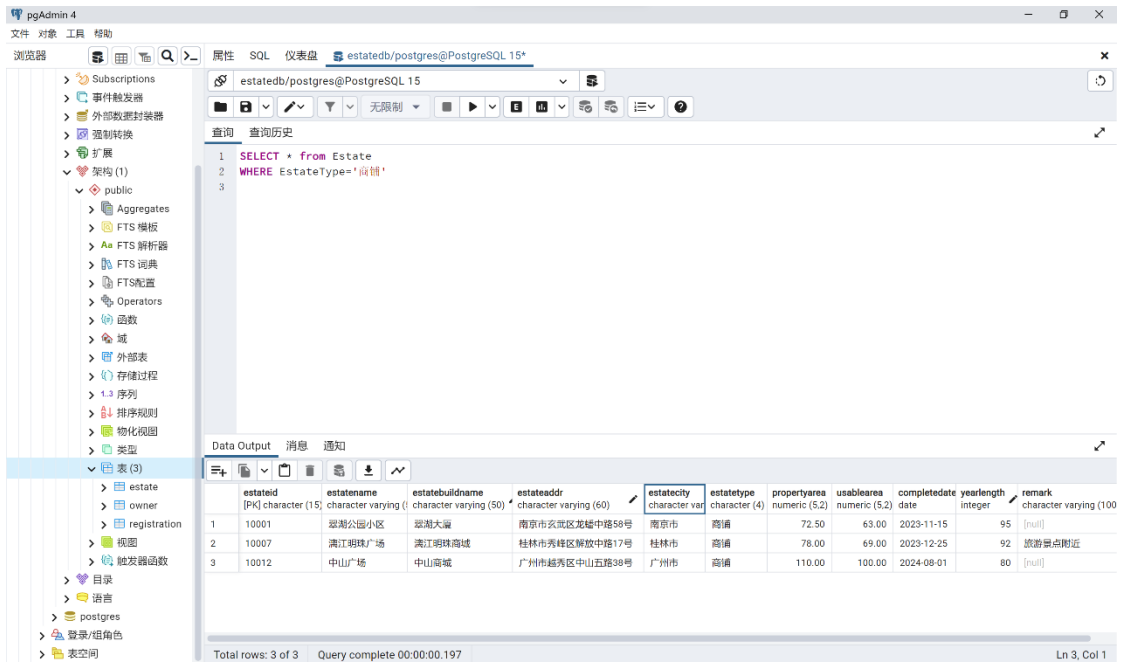


图 11：查询类别为商铺的房产信息

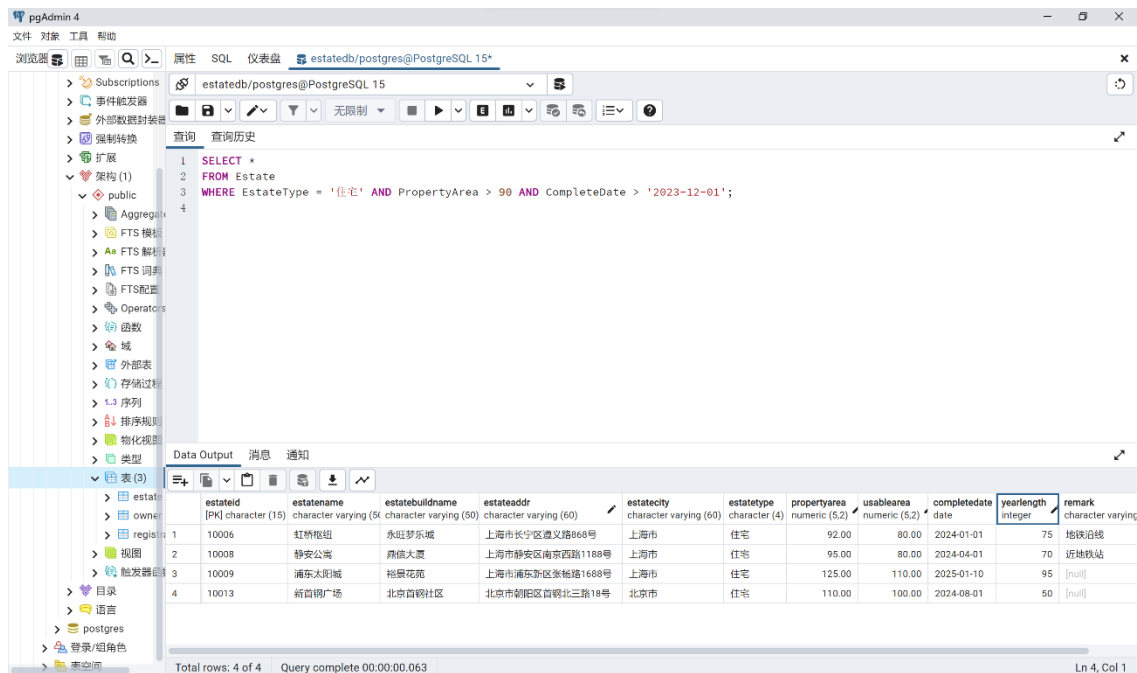
2.5 查询查询竣工日期为 2023 年 12 月 1 日后，产权面积 90 平米以上的“住宅”的房产信息

本题的 SQL 数据查询语句为

```
SELECT *
FROM Estate
WHERE EstateType = '住宅' AND PropertyArea > 90 AND
CompleteDate > '2023-12-01';
```

查询到的结果共计 4 条，与预期符合。

程序运行截图如下图：



The screenshot shows the pgAdmin 4 interface. The SQL query is displayed in the 'Query' tab, and the results are shown in the 'Data Output' tab. The query filters for residential properties (住宅) with a property area greater than 90 and a completion date after 2023-12-01. The results table contains 4 rows of data.

	estateid	estatename	estatebuildname	estateaddr	estategcity	estatetype	propertyarea	usablearea	completedate	yearlength	remark
1	10006	虹桥悦庭	永旺梦乐城	上海市长宁区遵义路868号	上海市	住宅	92.00	80.00	2024-01-01	75	地铁沿线
2	10008	静安公寓	鼎信大厦	上海市静安区南京西路1188号	上海市	住宅	95.00	80.00	2024-04-01	70	近地铁站
3	10009	浦东太阳城	裕景花园	上海市浦东新区张杨路1688号	上海市	住宅	125.00	110.00	2025-01-10	95	[null]
4	10013	新首钢广场	北京首钢社区	北京市朝阳区首钢北三路18号	北京市	住宅	110.00	100.00	2024-08-01	50	[null]

图 12：查询竣工日期为 2023 年 12 月 1 日后，产权面积 90 平米以上的“住宅”的房产信息

2.6 查询个人在各地购买住宅 2 套以上的业主基本信息

本题的 SQL 数据查询语句为

```
SELECT A.PersonID AS 身份证号 , A.Name AS 姓名 , A.Gender
AS 性别,A.Occupation AS 职业, A.Addr AS 身份地址, A.Tel AS 电
话, COUNT(B.EstateID) AS 房产数
FROM Owner AS A JOIN Registration AS B ON
A.PersonID=B.PersonID JOIN Estate AS C ON
B.EstateID=C.EstateID
WHERE C.EstateType='住宅'
GROUP BY A.PersonID
HAVING COUNT(*)>2
```

我们使用多表关联的第一种写法来书写。值得说明的是，为了更好的展示用户的房产信息，我们将用户的房产数单列在查询结果最后一列。（对应 SQL 语句中的“COUNT(B.EstateID) AS 房产数”这一列）

程序运行结果找到了 2 位拥有两套以上房产的人——张三和李四，与预期结果相符合。

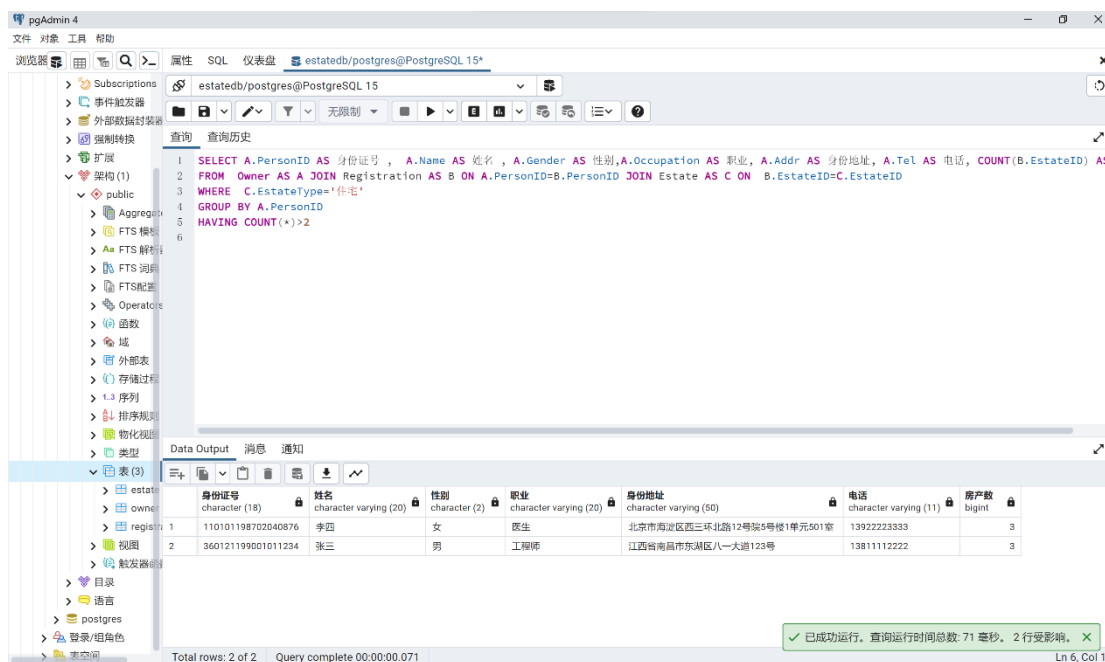


图 13：查询个人在各地购买住宅 2 套以上的业主基本信息

2.7 查询个人在特定城市购买住宅 2 套以上的业主基本信息

这里我们以查询北京市的业主基本信息为例子做演示。

本题的 SQL 数据查询语句为

```
SELECT A.PersonID AS 身份证号 , A.Name AS 姓名 , A.Gender AS 性别,A.Occupation AS 职业, A.Addr AS 身份地址, A.Tel AS 电话, COUNT(B.EstateID) AS 房产数
FROM Owner AS A JOIN Registration AS B ON
A.PersonID=B.PersonID JOIN Estate AS C ON
B.EstateID=C.EstateID
WHERE C.EstateCity='北京市' AND C.EstateType='住宅'
GROUP BY A.PersonID
HAVING COUNT(*)>2
```

我们使用多表关联的第二种写法——加入 JOIN 语句来书写。同样，值得说明的是，为了更好的展示用户的房产信息，我们将用户的房产数单列在查询结果最后一列。（对应 SQL 语句中的“COUNT(B.EstateID) AS 房产数”

这一列)

程序运行结果找到了 1 位在北京拥有两套以上房产的人——李四。上一题查询结果的张三并未在北京市购得 2 套以上房屋，因此不在表中，与预期结果相符合。

程序运行截图如下：

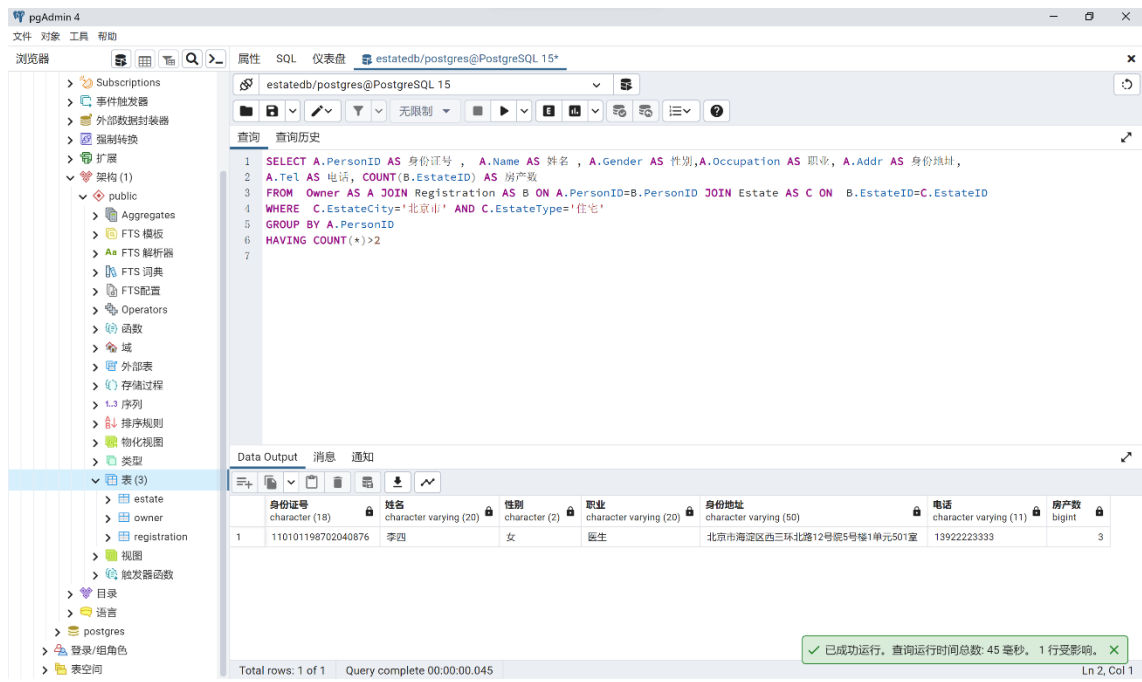


图 14：查询个人在特定城市(北京市)购买住宅 2 套以上的业主基本信息

2.8 统计 2023 年度某城市的各类房产销售面积

这里我们以查询上海市的各类房产销售面积为例子做演示。

本题的 SQL 数据查询语句为

```
SELECT A.EstateType AS 房产类型, SUM(A.PropertyArea) AS 销售  
面积  
FROM Estate AS A, Registration AS B  
WHERE A.EstateID=B.EstateID AND A.EstateCity='上海市'  
AND B.PurchasedDate BETWEEN '2023-01-01' AND '2023-12-31'  
GROUP BY 房产类型;
```

根据预期可知，张三 2023 年在上海市买了 2 套房子，赵六买了 1 套，共计 $92+95+125=312$ 平米。另外张三在 2023 年还买了一个上海市 10 平米的车位。

程序运行结果见下一页，符合预期。

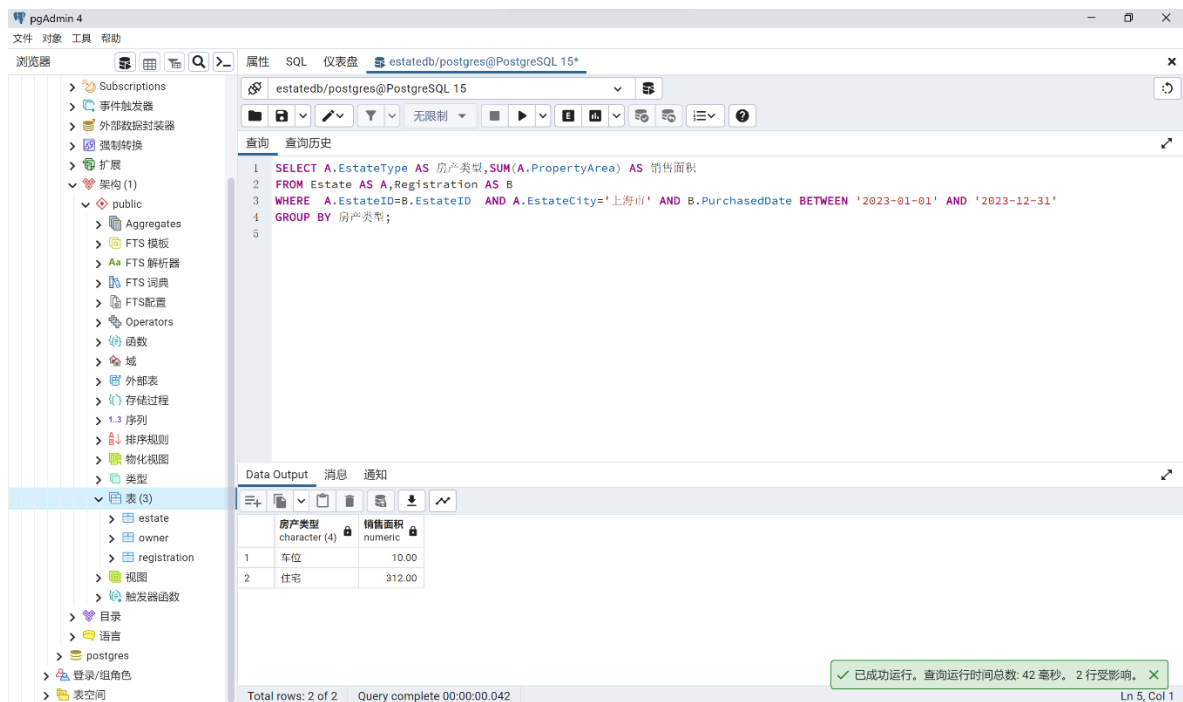


图 15：2023 年度某城市(上海市)的各类房产销售面积

2.9 通过视图查询指定身份证号下，该业主的购置房产信息

首先，我们创建一个 Usr_view 的视图，用于查询张三（身份证号 PersonID 为 360121199001011234）的购置房产信息。本题的创建视图 SQL 创建语句为

```

CREATE VIEW Usr_view AS
SELECT R.RegisterID AS 登记编号,E.EstateName AS 房产名称,E.EstateType AS 房产类型,E.PropertyArea AS 产权面积,R.Price AS 购买金额,
      R.PurchasedDate AS 购买日期,E.EstateBuildName AS 房产楼盘,E.Estatecity AS 房产城市
FROM Registration AS R,Estate AS E
WHERE E.EstateID=R.EstateID AND R.PersonID
='360121199001011234'
ORDER BY 购买日期 DESC;

```

本题程序运行截图见下一页

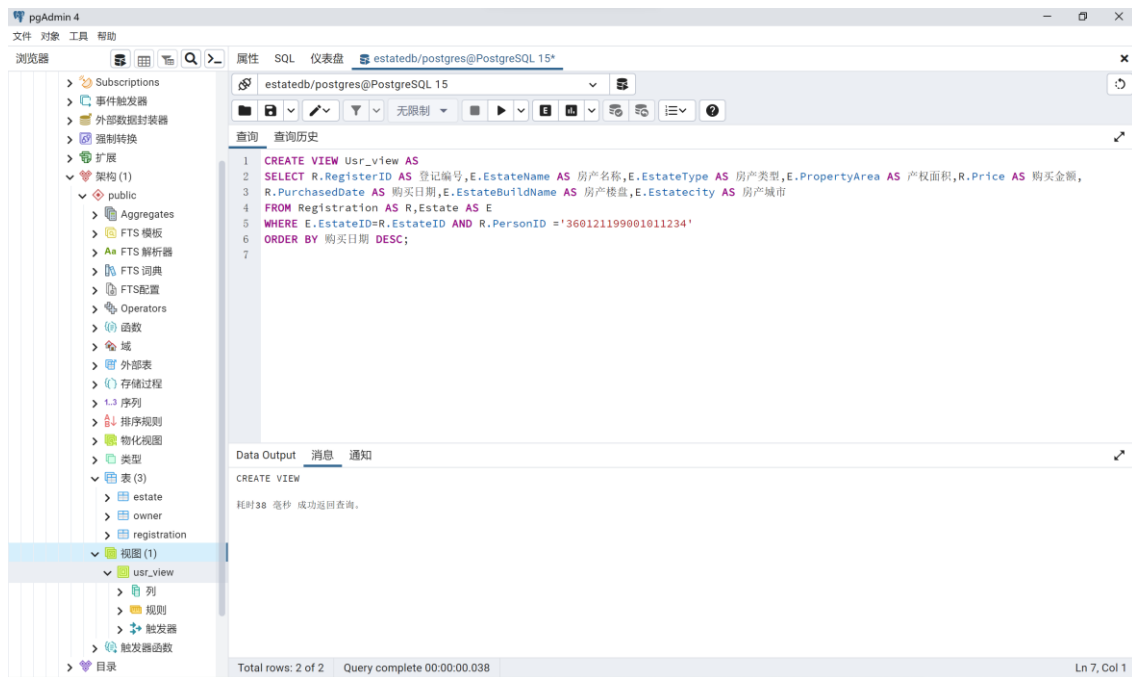


图 16: 创建 Usr_view 视图

随后使用

```
SELECT * FROM Usr_view;
```

查询到张三购置的四套房产信息

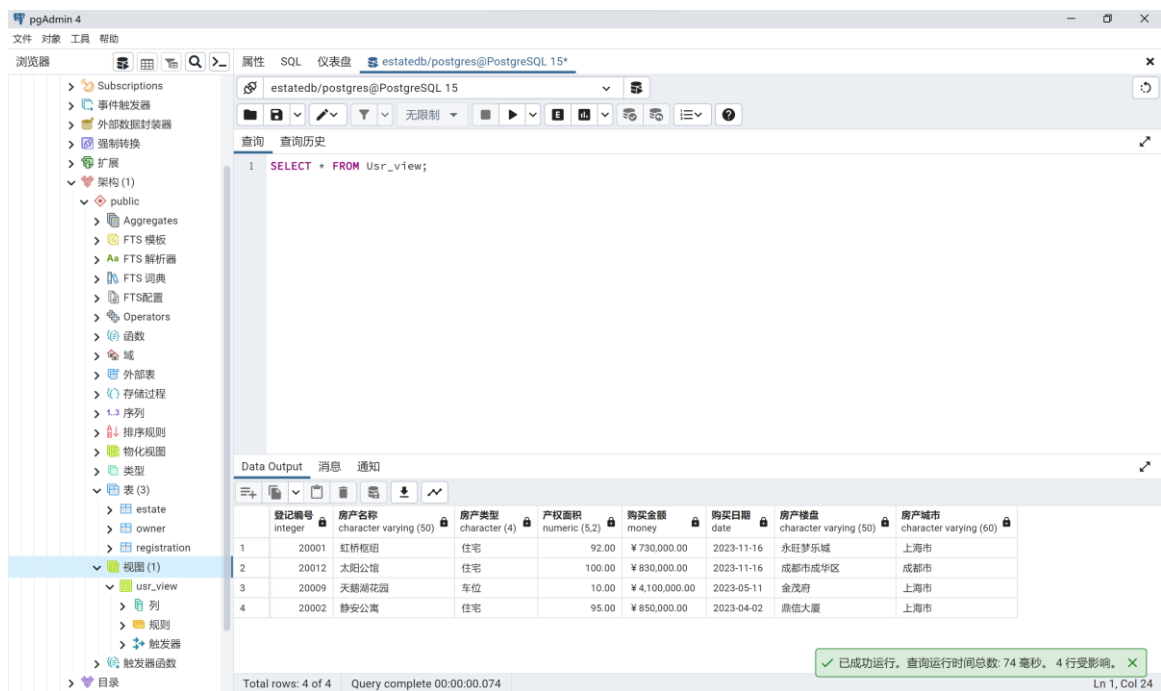


图 17: 通过视图查询指定身份证号下, 该业主的购置房产信息

2.10 分组统计 2023 年度各城市的住宅销售套数与总销售金额

首先，我们创建一个 Total_view 的视图，用于查询 2023 年度各城市的住宅销售套数与总销售金额。本题的创建视图 SQL 创建语句为

```
CREATE VIEW Total_view AS
    SELECT E.EstateCity AS 城市, COUNT(*) AS 住宅销售套数 ,
    SUM(R.Price) AS 销售金额
    FROM Estate AS E , Registration AS R
    WHERE R.EstateID=E.EstateID AND R.PurchasedDate
    BETWEEN '2023-1-1' AND '2023-12-30' AND E.EstateType='住宅'
    GROUP BY E.EstateCity;
```

程序运行结果为：

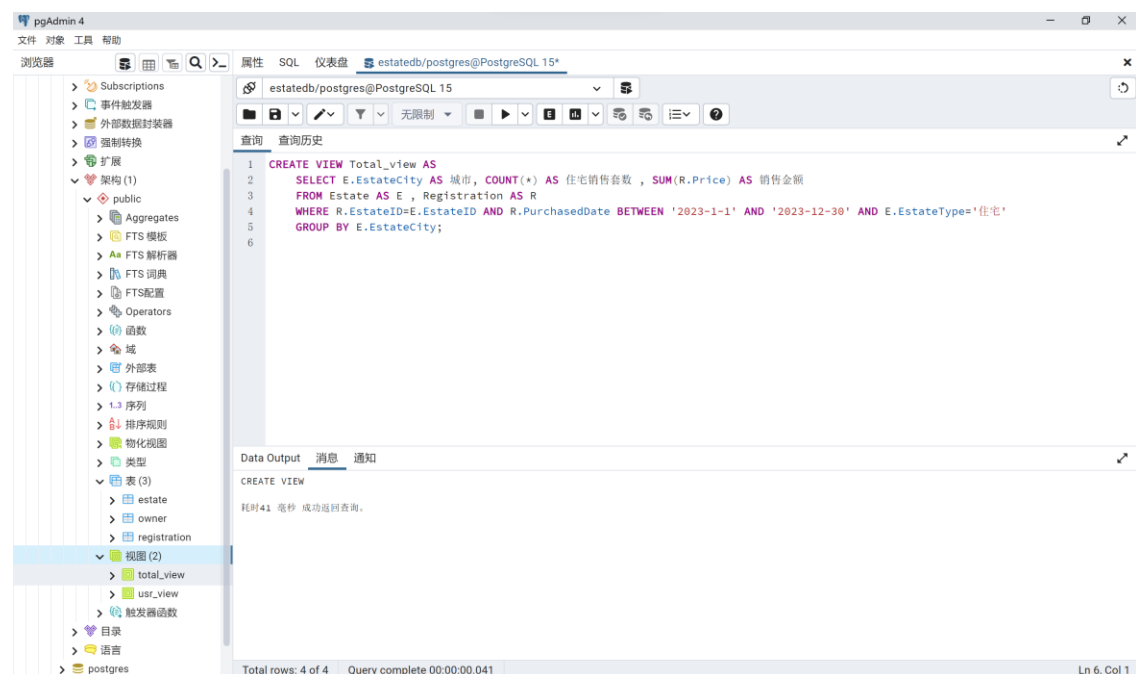


图 18：创建 Total_view 视图

随后进行查询

```
SELECT * FROM Total_view;
```

通过原始数据，我们预期在成都、上海、北京分别有 1、3、3 套房产在 2023 年售卖。其他城市售卖的房产要么不符合房产类型为住宅（比如南京市的为商铺），要么不符合购买日期在 2023 年内，因此其余城市结果为 0。

得到的程序查询结果见下一页，结果完全符合预期。

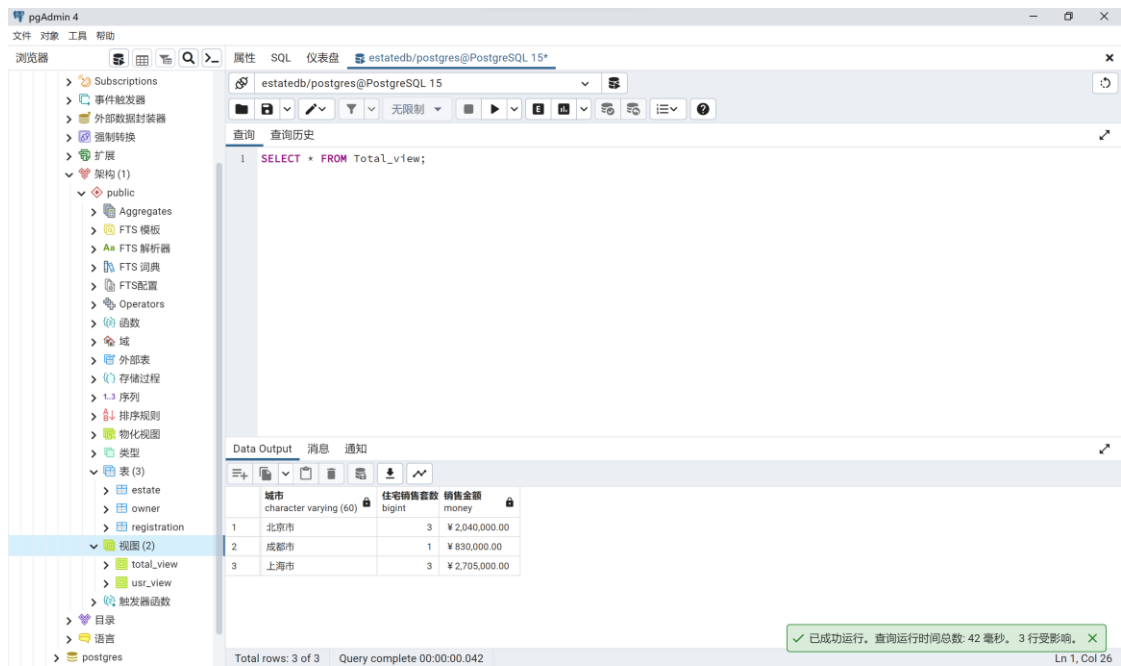


图 19：分组统计 2023 年度各城市的住宅销售套数与总销售金额

三、 挑战性问题研究

3.1 三种查询方案的执行代价

首先表明笔者的结论，个人认为 Q3 的查询代价最小。因为它先进行了选取，然后进行了一次耗时较少的关系的连接操作（ ∞ ），下面将对其进行具体分析。

3.1.1 Q1~Q3 的语句如何实现业务要求

Q1~Q3 是四种等价的关系运算表达式，它们可以解决查询“软件学院”的教师名单的问题。在研究四种查询方案的代价之前，我们首先需要了解这四种方案是如何查询到需求的信息的——

- **Q1:** 该查询使用了笛卡尔积（ \times ）、筛选（ σ ）和投影（ Π ）三种关系运算符。其中， $\text{College} \times \text{Teacher}$ 表示对两个表进行笛卡尔积操作得到一个新的虚拟表，然后通过选择条件 σ 来保留符合条件的记录，最后通过投影操作 Π 来选取指定的属性列输出查询结果。
- **Q2:** 该查询使用了外连接（ ∞ ）、筛选（ σ ）和投影（ Π ）三种关系运算符。其中， $\text{College} \infty \text{Teacher}$ 表示对两个表进行自然连接操作得到一个新的虚拟表，并保留左表（即 College）中的所有记录，然后通过选择条件 σ 来保留符合条件的记录，最后通过投影操作 Π 来选取指定的属性列输出查询结果。

- **Q3:** 该查询使用了外连接 (\bowtie)、筛选 (σ) 和投影 (Π) 3 种关系运算符。其中, $\text{Teacher} \bowtie (\sigma_{\text{A.CollegeName}='软件学院'}(\text{College}))$ 表示对 College 表进行选择操作得到一个子表, 然后与 Teacher 表进行外连接, 最后通过投影操作 Π 来选取指定的属性列输出查询结果。

从上述说明可以直观的看出, 四者的主要区别在于笛卡尔积和连接的使用上面。对于投影操作, 四者的使用完全等价, 故后续不讨论投影的影响。

3.1.2 Q1~Q3 的语句具体查询代价计算

输入输出与中间数据访存代价是最主要的影响查询代价因素, 一般来讲, 读取表, 读写中间文件 (当用到笛卡尔积/连接时需要产生中间文件), 选取, 投影是主要影响查询代价的四个基本内容。

首先我们给出基本运算的查询代价公式。

对于选择运算 (σ), 我们采用**全局选择代价**来代替其查询代价, 代价为需遍历总条数 N , 即:

$$O(\sigma) = N$$

对于投影运算 (Π), 查询代价为遍历的条目 $N \times \text{投影的列数 } m$, 即

$$O(\pi) = N \times m$$

已知学院表 College 中已有 20 个学院信息, 教师表 Teacher 中已有 4000 名教师信息。假设一共有 400 条数据符合查询条件。

首先, 数据库需要读取教师表和学院表, 将表格信息读入到内存中。对四类语句他们的查询代价相同, 读教师表的查询代价均为 4000, 读学院表的查询代价均为 20。

Q1: 使用了笛卡尔积运算, 用**笛卡尔积**连接后的元组为 $4000 \times 20 = 80000$ 条, 每块能装 1 个元组。那么写中间文件需要 80000 的查询代价; 随后**选取**这 80000 中符合要求的条目, 那么全局搜索代价为 80000; 再将 400 个元组遍历挑选 3 个属性进行**投影**, 需要查询代价 $400 \times 3 = 1200$

Q2: 使用了连接运算, 首先对两个表进行**连接**, 因为每个老师只对应一个学院, 那么只需遍历老师, 代价为 4000; 然后进行**选取**操作, 全局选择代价为 4000; 最后进行**投影**操作, 需要遍历筛选出来的 400 条记录, 需要挑选出 3 个属性, 因此代价为 $400 \times 3 = 1,200$ 。

Q3: 在执行查询 Q3 的过程中, 首先需要对 College 表做**选取** (σ) 操作, 需要遍历 20 条记录, 因此全局选择代价为 20。最终选定了一条软件学院的数据; 然后进行等值**连接操作** (\bowtie), 因为教师记录有 4000 条, 因此需要遍历 4000 次,

连接的代价为 4000;最后进行**投影(Π)**操作，需要遍历筛选出来的 400 条记录，需要挑选出 3 个属性，因此代价为 $400 \times 3 = 1200$ 。

表 1：四条语句的查询代价计算

语句	读教师表 到内存	读学院表 到内存	读写 中间文件	选择	投影	总代价
Q1	4000	20	80000	80000	1200	165220
Q2			4000	4000		13220
Q3			4000	20		9240

综上计算可以得出，Q3 的查询代价最小，Q1 的查询代价最高。

3.1.3 Q1~Q3 的语句查询代价的差异原因

我们发现，Q1 的查询代价与其他几组相差一个数量级的差距。

那么，为什么使用笛卡尔积会降低查询效率呢，下面，对笛卡尔积和连接的查询效率差异原因做分析——

📖 **笛卡尔积：**笛卡尔积操作后，新表的行数等于左右两个表的行数相乘的乘积，因此会导致关系的行数急剧增加。假设左表有 m 行记录，右表有 n 行记录，那么笛卡尔积操作后得到的新表就有 $m * n$ 行记录。每生成一行记录就需要对数据库进行一次访问，从而导致访存操作的次数大大增加，影响查询效率。

📖 **连接：**连接操作根据连接方式和连接条件来决定查询的代价。具体来说，内连接和等值连接的查询代价相对较低，仅仅需要生成 n 行记录，因此它们需要比较少的访存操作。

因此，Q1 选择笛卡尔积进行查询是非常不明智的。它因为生成较大的中间文件大大降低了查询效率

而 Q2、Q3 的主要差别在于选择和连接的顺序上。

Q2 是先连接；Q3 是先选择。连接生成的中间文件较大，因此再对其进行选择，查询代价也比较大（需要遍历的条目多）。Q3 先选择学院，从 20 个学院筛选后学院条目只剩下 1 个，与 4000 条教师进行连接。Q3 虽然连接成本与 Q2 相同但是选择成本极低，因此在本题的大样本大数据中 Q3 更占优势。

3.2 当增加索引后

索引是数据的目录，它可以极大的减少查询所带来的时间损耗。在建立索引的过程中，通常会采用二叉搜索树/B+树等**树状结构**来实现索引的快速查找，这样的数据结构的查找时间复杂度通常为 $O(\log_2 n)$ 。因此，当使用索引进行查询时，我们通常可以将查找代价的时间复杂度视为 $O(\log_2 n)$ ，其中 n 表示元组数量。

那么 Q3 的执行过程变为——首先需要对 College 表做**选取**(σ)操作，可以使用索引直接找到“软件学院”，因此时间代价缩减为 $O(\log_2 20) \approx O(1)$ 。最终选定了一条软件学院的数据；然后进行等值**连接操作**(\Join)，教师记录有 4000 条，可以根据定义的外键索引快速定位连接，连接的代价为 $O(\log_2 4000) \approx O(12)$ ；最后进行**投影**(Π)操作，需要遍历筛选出来的 400 条记录，需要挑选出 3 个属性，因此代价为 $O(400 \times 3) = O(1200)$ ，加上读表到内存(4000+20)等操作最终可以计算出查询代价为 5233。

可以看到，增加索引后可以大大减小查询时间，查询代价近乎减小 40%。因此合理利用索引对于查询的优化非常显著。

3.3 结论

综上所述，我们可以得出结论——

- ① 使用笛卡尔积会大大降低查询速度，应减少笛卡尔积的使用。
- ② 应尽量使得被连接的表数据量少。更建议先进行选择再进行连接。
- ③ 在大数据的样本中，尽量使得选择操作成本低（比如先对部分数据少的表进行选择），可以节省很多时间。
- ④ 合理的使用索引会在一定程度上提高查询效率，特别是选择的条件恰好处于有索引的属性之中时。

可以借助如上方法加快对数据查询的效率，减少查询等待。同时，还可以不断优化查询方法，调整索引结构，对于未来大数据、大样本的数据库技术有着非常重要的借鉴意义和参考价值。

四、 学习收获与心得体会

通过本次大作业与实践研究，收获颇多。

首先，通过了解 SQL 关系型数据库的强大生命力背后的原因，了解到这类数据库背后广阔的应用价值。它的诸多闪光点值得我们国产数据库进行借鉴发掘，为此可以不断提升我国国产数据库的实力与国际水平，进而可以更好的开展国际合作。进而探讨标准化和技术创新对数据库发展的意义，我深刻了解到标准化和

创新对于技术发展不可磨灭的推动作用和重要意义，我们应该坚持技术标准化，不断推陈出新。

随后，动手实践在 PostgreSQL 中实践操作数据库的创建、表创建删除、查找、视图创建等各项操作，我才明白“纸上得来终觉浅，绝知此事要躬行”。书本课堂上的内容需要更多的实践才能熟能生巧。

最后，通过挑战性学习，了解到不同的查询机制对查询代价的不同影响，我们应该更多的使用先选择再连接，尽量不笛卡尔积的原则进行查询，也了解到索引这一数据目录的重要作用。通过研讨，使得我加深了对关系运算的理解与认识，加强了对索引内在数据结构的理解。