数据库第三次课程大作业

姓名：* 学号：***

一、 分析与说明

作为数据库开发人员，在开发数据库应用系统时，遵从工程伦理、职业道德和国家法规是非常重要的。下面将结合实际的例证，从这三面浅析开发数据库应用系统的一些可能要点——

1.1 工程伦理角度

遵从工程伦理意味着我们需要在开发过程中注重系统的**可靠性、稳定性和安全性**，从而尽可能减少系统出现漏洞和故障的可能性，**需要开发人员承担起了开发安全可靠数据库的责任和担当**。

这需要开发人员对整个开发过程进行完善的计划、设计、实现和测试，并进行评审和验证，从而确保开发出的数据库应用系统达到高度的标准。

为此数据库开发人员需要做好以下几点：

1 部署访问控制与防火墙

数据库开发人员应对数据库的数据表进行一定的访问限制，部署相关防火墙系统。可从以下角度考虑：

- 直接访问控制：数据库开发人员需要采取有效的访问控制措施，只允许授权用户访问相应的数据，避免未经授权的用户或程序对数据进行访问和修改。
- 敏感数据加密：对于敏感数据，数据库开发人员可以采用数据加密技术，以避免数据在传输过程中被窃取或篡改。

2 进行安全审查和分析

在开发过程中需要进行适当的评估和风险控制尽可能消除开发风险。在开发过程中遵从规范的开发流程，严格按照规范流程进行，确保项目完整、安全。

数据库开发人员也需要在系统中引入**安全审计机制**，对系统和用户的操作进行记录和跟踪，及时发现和处理安全问题。

3 建立应急容灾机制

数据库开发人员需要制定合理的容灾机制，在出现意外情况（如撞库等）能够合理进行数据库安全防御。定期对重要数据进行备份，并建立数据恢复机制，以便在系统出现故障或攻击时可以及时恢复数据。

以下是有关阿里云数据库的一个正面例子——

阿里巴巴的数据库是国内领先的数据库团队。他们借鉴 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），通过制定一系列访问控制和密钥加密限制，强化鉴权措施，通过规范化流程设计出了一套数据库安全管理方案模式；建立容灾机制，成功的树立了我国数据库产业安全规范的标杆。

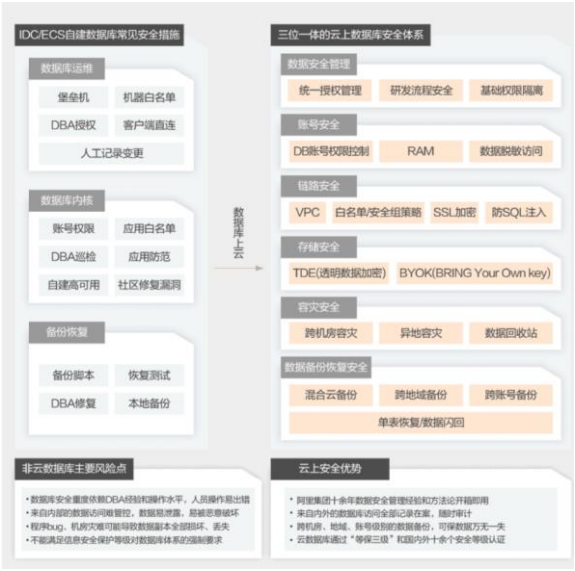

图 1：阿里团队数据库安全架构

阿里云数据库的数据库开发工程师遵守了工程伦理，从不同角度强化系统的可靠性、稳定性和安全性，承担起了开发安全可靠数据库的责任和担当。

1.2 职业道德角度

遵从职业道德则要求我们在处理用户数据时要严格保护用户的隐私，任何未经授权的行为都应该被禁止。**这就需要数据库开发者恪守道德底线，保持良好的职业操守。**

数据库开发者需要清楚确认用户数据使用的目的和方式，获得用户的明确授权，采取严密的安全措施，不给自己留后门。

因此，从职业道德角度来看，我们需要注重以下几个方面：

1 尊重用户隐私

作为数据库开发人员，我们需要始终尊重用户的隐私权，不得通过恶意手段获取、泄露或滥用用户的个人信息。我们需要了解相关法律，遵守规范，确保所处理的数据符合相关政策、行业标准及法律法规要求。

2 保证数据安全

保证数据安全对于数据库开发人员而言是非常关键的一点。我们需要严格控制数据访问权限采取有效的加密技术和安全防护措施来确保数据在存储、传输和使用过程中不会被未经授权的访问者获得；同时，对于敏感数据进行相应的鉴权

操作。不能故意为自己或他人留后门或漏洞。

以下是有关美国海关数据库的一个反面教材——

据《华盛顿邮报》2022 年 9 月 15 日报道，美国海关和边境保护局（CBP）的负责人透露：约 2700 名海关和边境保护局官员无需通行证即可访问海关信息数据库，其中记录了通关人员的各类信息（包含旅客设备中的照片、联系人、通话记录和信息等）。

俄勒冈州一位参议员罗恩·怀登（Ron Wyden）批评该数据库开发机构“允许不分青红皂白地搜查美国人的私人记录”，并呼吁加强隐私保护。

上述报道反映了一个重要问题，即海关信息数据库的开发机构存在数据安全和个人隐私保护问题，他没有保证数据安全也没有进行访问控制，导致敏感数据外泄。这是极其不符合职业道德的开发行为！作为信息系统开发人员，我们应该对此问题高度重视，积极履行职业道德和社会责任。

1.3 国家法规角度

遵从国家法规则要求数据库开发者**必须遵守《网络安全法》等相关法律法规**，并按照相关规定**建立健全的信息系统安全管理制度**。

开发机构应该严格遵守法律法规和标准，例如《个人信息保护法》、《网络安全法》等，明确数据用途，确保使用所收集的信息是合法的。

当然，对于数据库领域，我国也出台了一系列对于的指导文件。典型的例如国家标准 GB/T 15579-2008《数据库系统安全性技术要求》，他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。

同时，我们还需要**配合监管部门开展安全检查**，并及时报告和处理安全事件。

以下是有关滴滴公司的一个反面教材——

7 月 21 日，国家互联网信息办公室依据《网络安全法》《数据安全法》《个人信息保护法》《行政处罚法》等法律法规，对滴滴公司处人民币 80.26 亿元罚款。

经查明，滴滴公司共存在 16 项违法事实，大致有：违法收集用户手机相册中的截图信息、过度收集乘客人脸识别信息、以明文形式存储司机身份证号信息 5780.26 万条、未明确告知乘客情况下分析乘客出行意图信息、未准确、清晰说明用户设备信息。

这些违法事实体现出滴滴公司内部员工在设计数据库等环节时，未充分明晰法律条款的规定，擅自扩大收集范围或不进行加密处理。因此，作为数据库从业人员，更应该知法懂法遵守法律法规，不要在信息安全的法律边缘试探。

1.4　总结

综上所述，作为数据库开发人员，我们需要始终秉持职业道德和社会责任，不断提升自身素质和专业技能，努力开发出高质量、安全可靠的数据库应用系统，为用户和社会做出更多的贡献。

二、　实践操作题

2.1　概念数据模型的设计

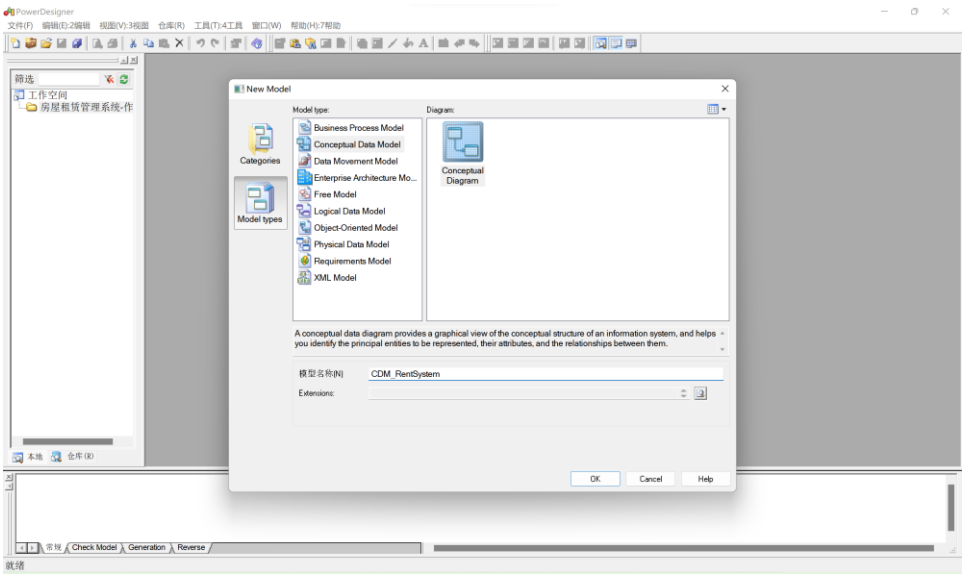

图 2：创建概念数据模型

首先，使用 Power Designer 软件创建一个概念数据模型。

在一个典型的房屋租赁管理系统中，主要有房东、租户、房产、合同这四类实体存在。

首先定义租户实体，设置身份证号为标识符，以及姓名、电话、性别属性：

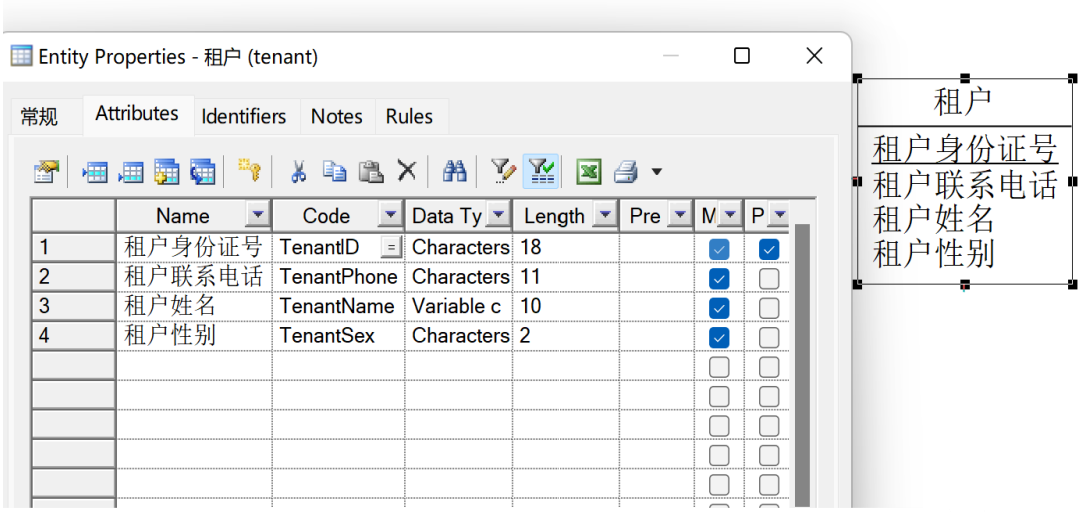

图 3：定义租户实体

同理，设计房东实体，标识符为房东身份证号，以及姓名、电话、性别属性：

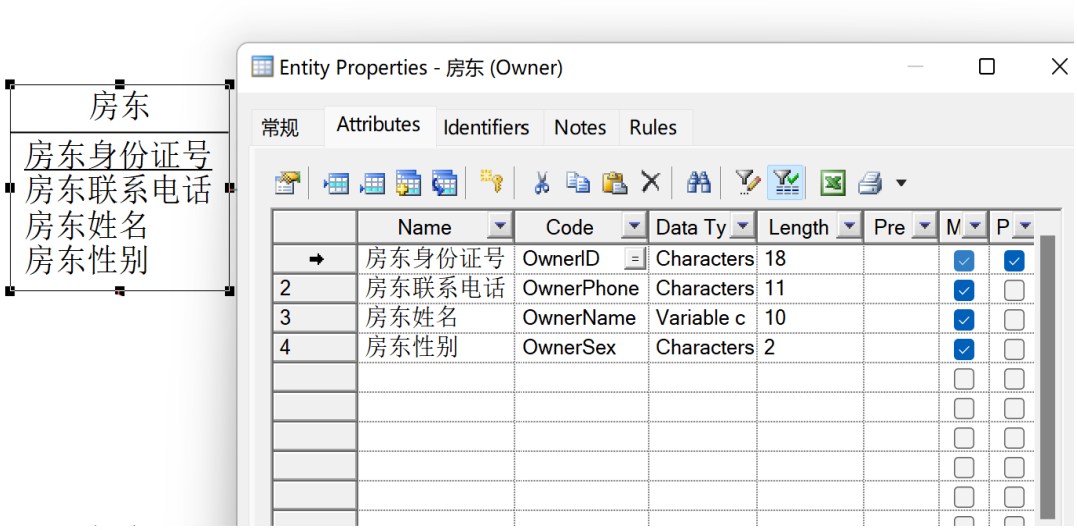

图 4:定义房东实体

定义房产实体，设置房产编号为标识符，截图参考下一页：

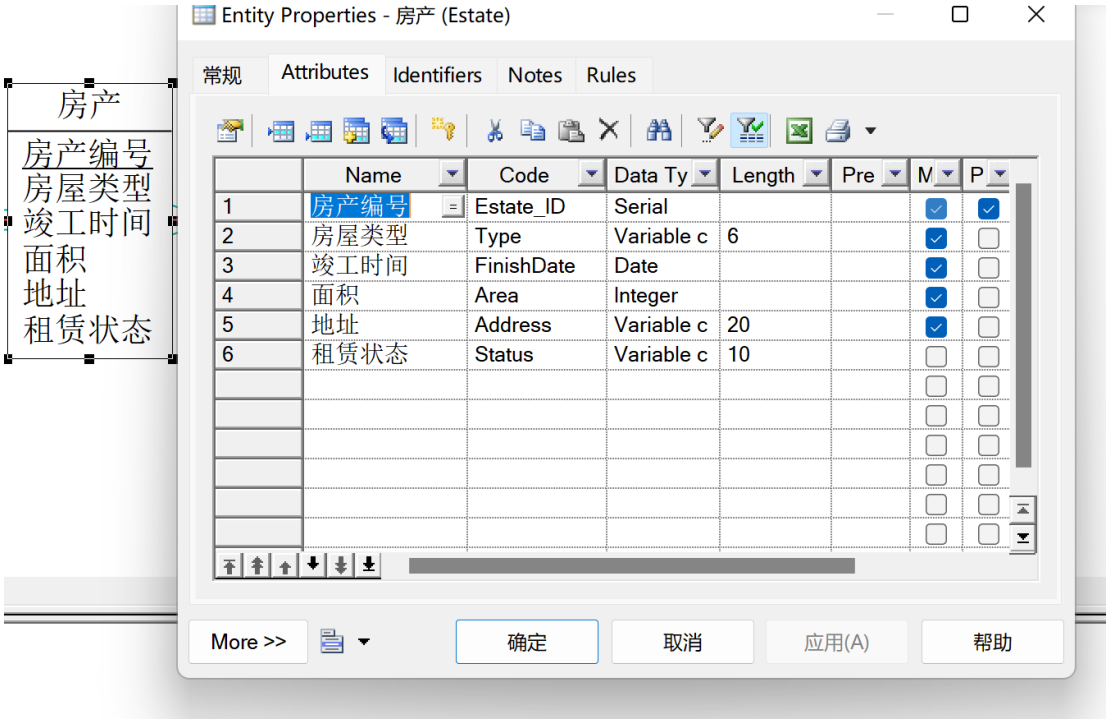

图 5：定义房产实体

定义租赁合同实体，以合同编号为标识符：

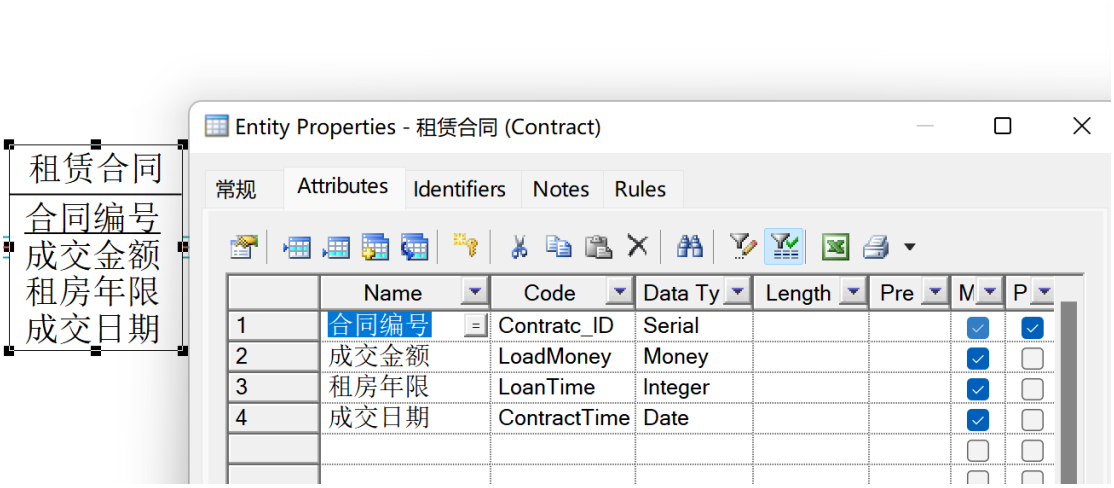

图 6：定义租赁合同实体

接下来为他们创建关系

房东持有（Own）房产，且一个房产只能对应一个房东，但是一个房东可以拥有多套房产，因此其为一对多的关系。

因此可以定义如下图的关系

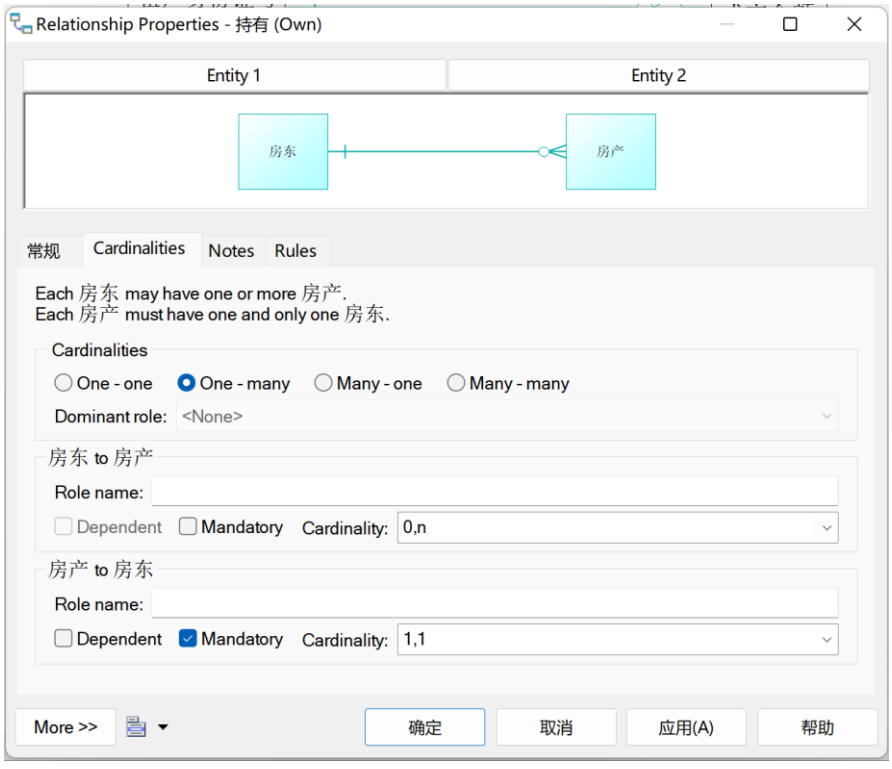

图 7：定义持有（Own）关系

对于租户和租赁合同的租赁关系（Rent），也是一对多的关系（一个租户在实际可以有多个租赁合同），但是一个合同只能有一个租户。即有如下关系：

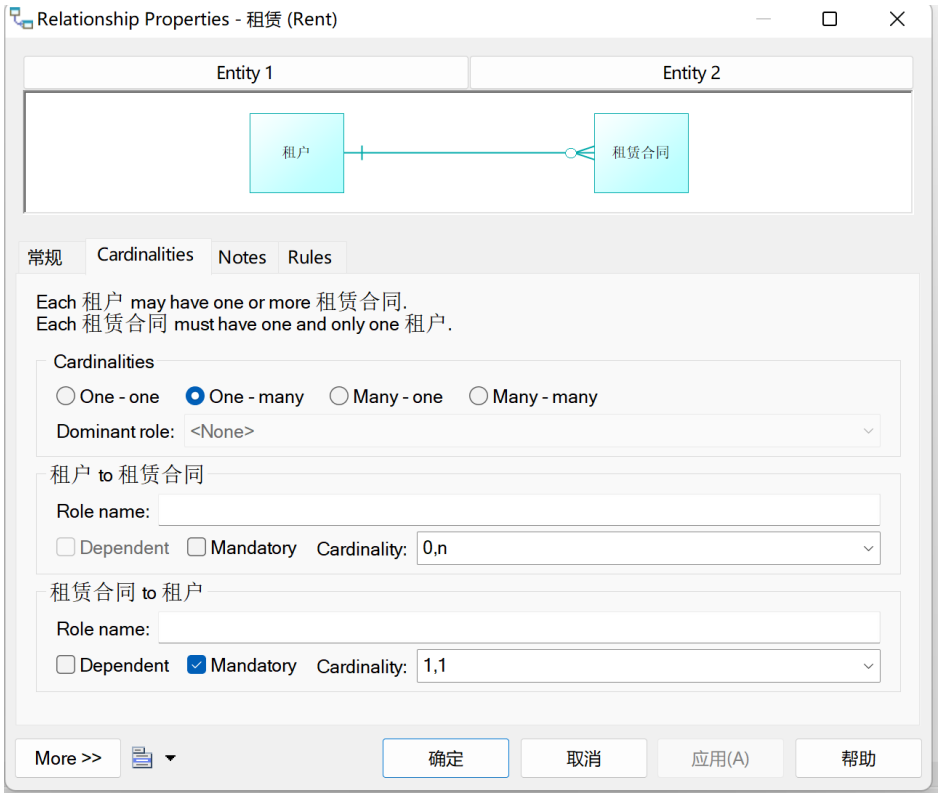

图 8：定义租赁（Rent）关系

对于租赁合同和房产，一个租赁合同会唯一的记录一个房产信息，但是一个房产可以被多个租赁合同记录（包括已经退租的），因此可以得到如下的关系：

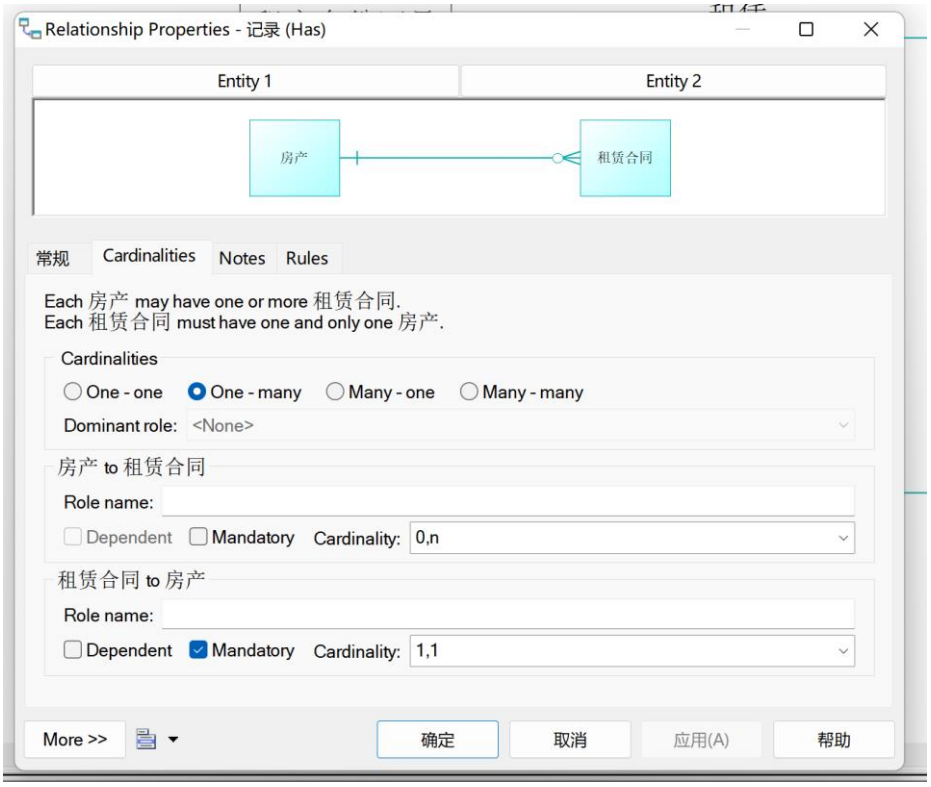

图 9：定义记录关系

同理，房东和租赁合同有签署关系。

最终我们得到了如下的概念数据模型——

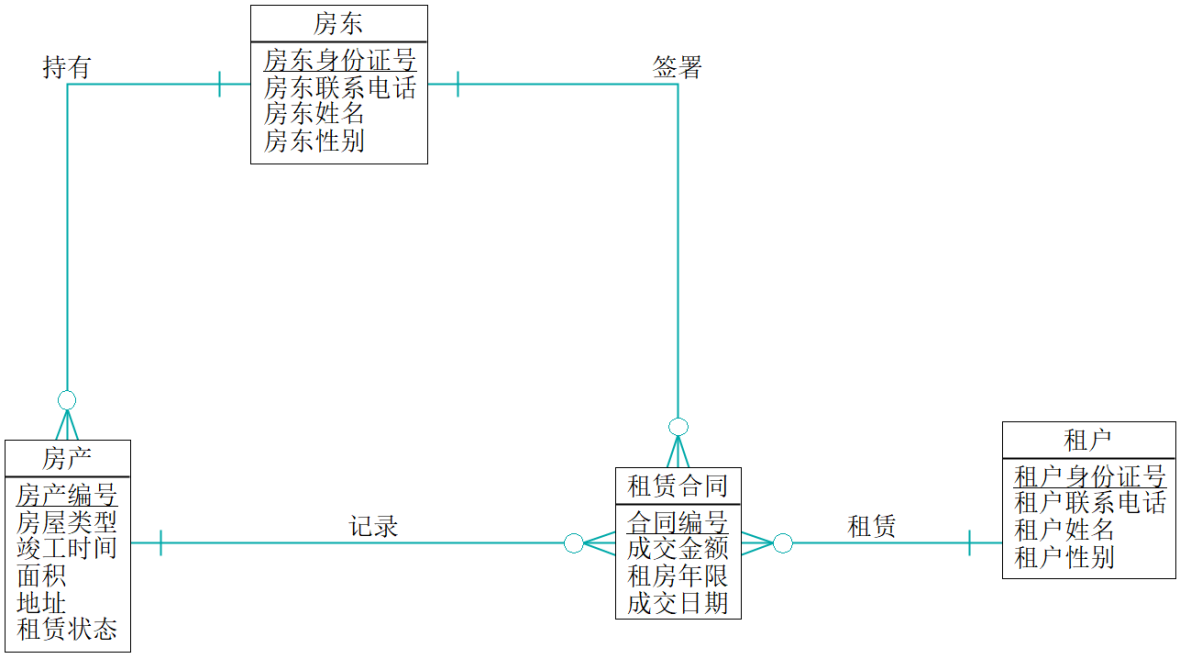

图 10：房屋租赁系统的概念数据模型（CDM）

2.2 逻辑数据模型的设计与规范化

接下来将概念数据模型转换为逻辑数据模型

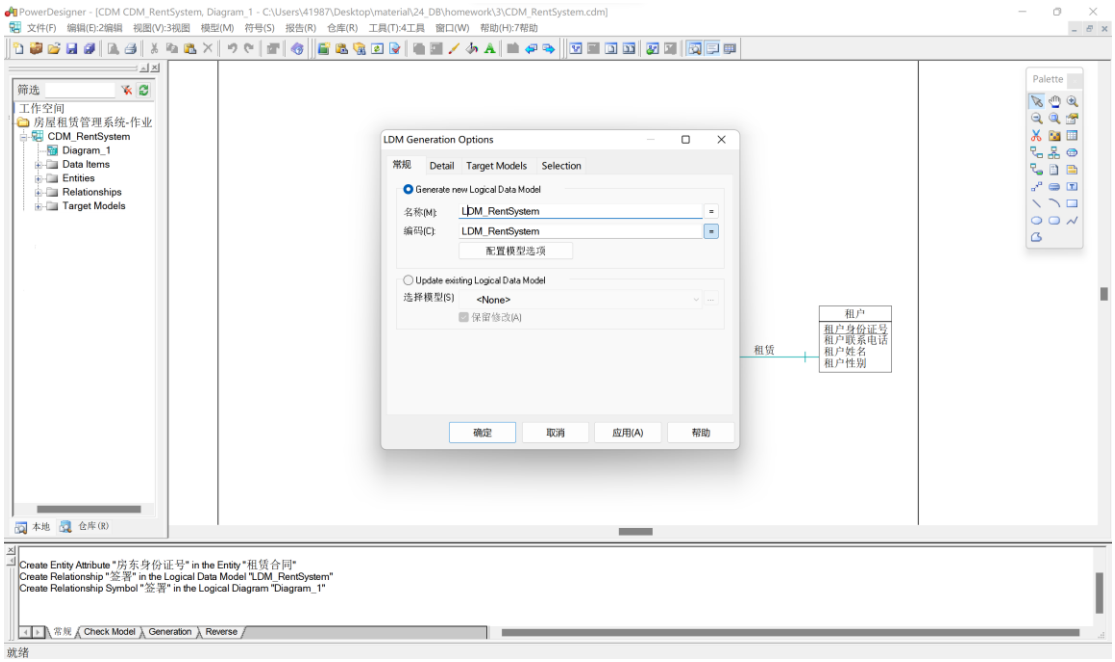

图 11：转换逻辑数据模型

转换得到的逻辑数据模型见下：

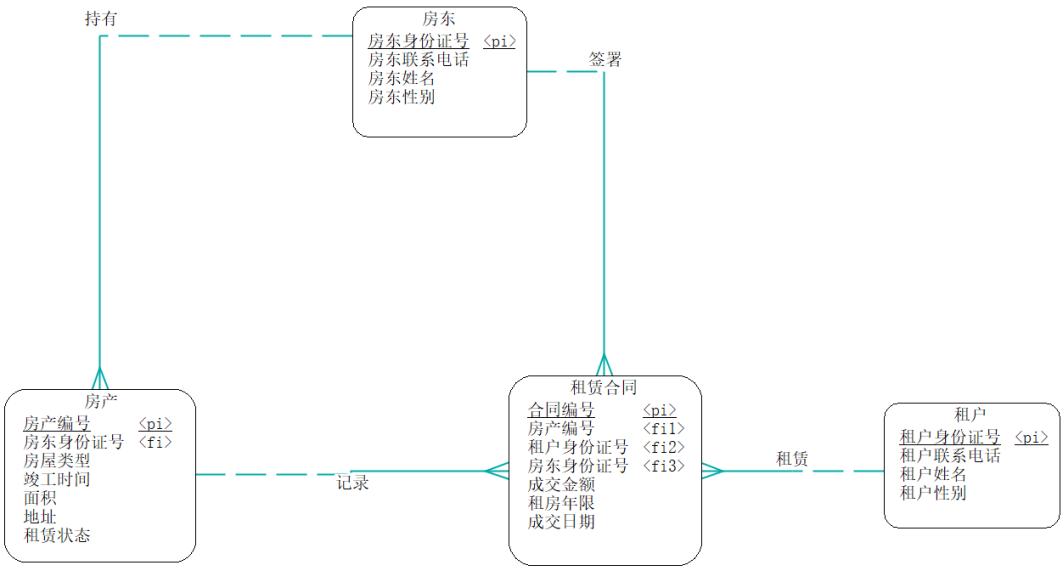

图 12：软件自动转换生成的逻辑数据模型（LMD）

接下来，我们对这个模型进行部分的修改，使之完成规范化设计，更加符合实际业务需求。

1. 首先，对每个实体，属性都不可再细分，其已经满足**第一范式**。
2. 对于每个实体，其内部也消除了关系中的属性部分函数依赖。实体中不存在复合主键，也就不存在某个非键属性单独依赖某个单一主键的特性。因此其**满足第二范式**。
3. 对于每个实体，其内部也切断了关系中的属性传递函数依赖。每一个非键属性不能由其他属性单独确定。因此其也满足**第三范式**。
4. 他也满足**巴斯-科德范式（BCNF）**，即对于每个实体，所有函数依赖的决定因子都是候选键。
5. 对于**第四范式**，每个实体内部的取值唯一，不存在多值的情形，因此可以认为上述的逻辑数据模型满足第四范式。

综上所述，上述的逻辑数据模型已经满足规范化设计要求。

2.3 物理数据模型及索引、视图设计

接下来将逻辑数据模型转换为物理数据模型，点击工具、生成物理模型，出现如下弹窗

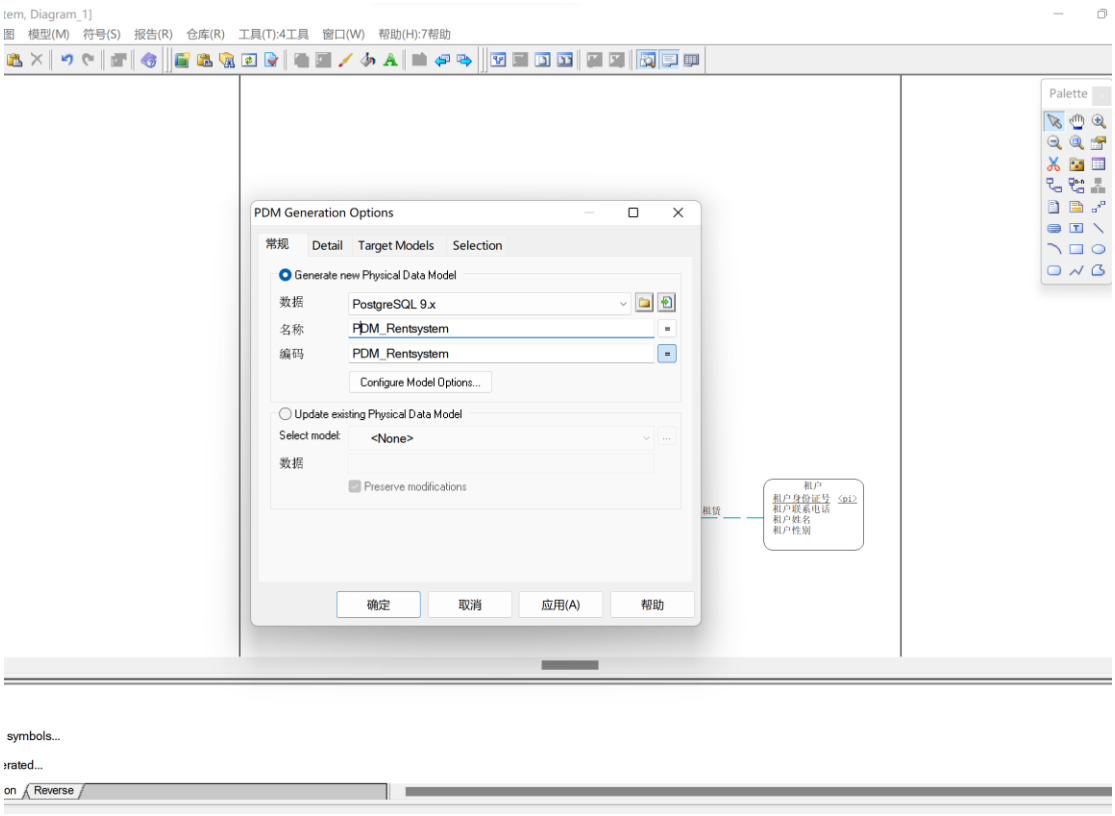

图 13：转换为物理数据模型

程序默认不展示数据类型和键类型，进入如下设置打开显示：（见下页图）

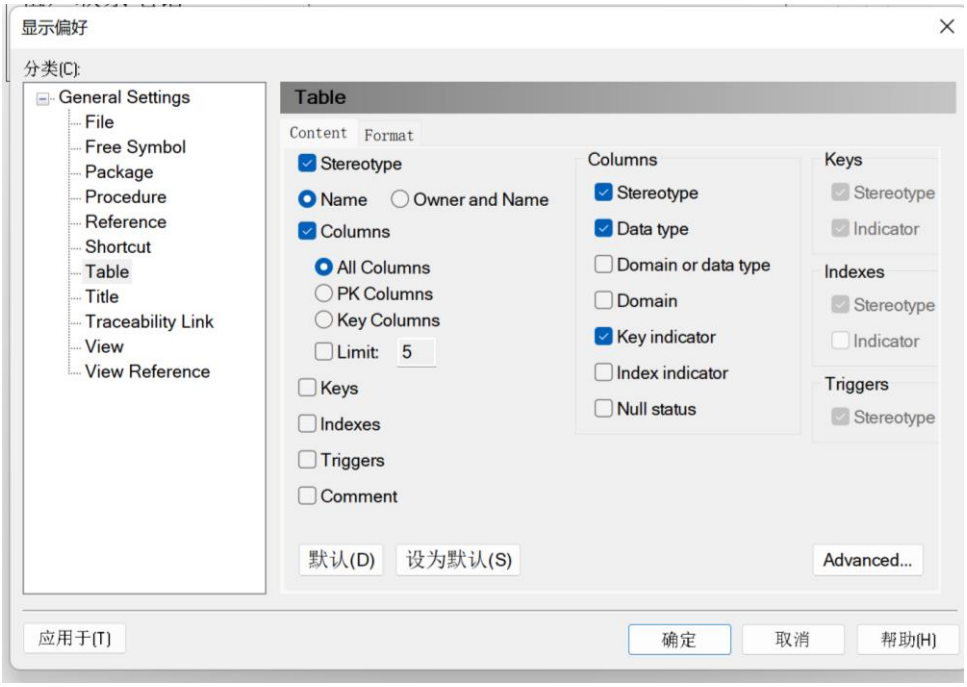

图 14：打开数据类型和键的显示

程序生成的物理数据模型如下图所示（表名和属性名已经转换为概念数据模型中预先设置的 Code）：

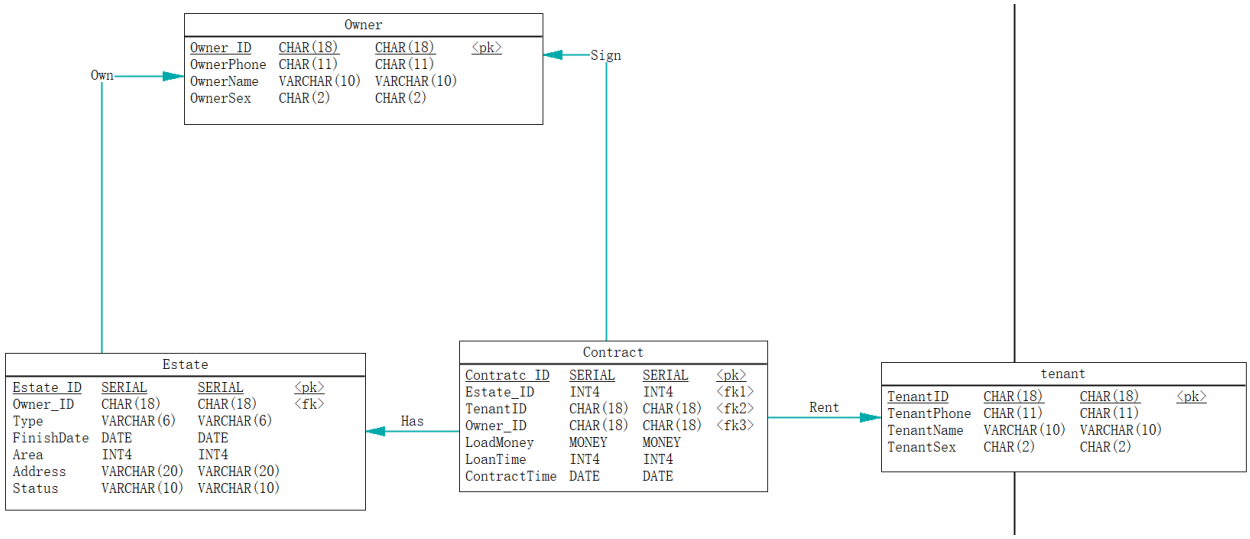

图 15：软件生成的物理数据模型（PDM）

下面考虑索引的设计：

首先，对于租户和房东，一般需要通过身份证号、电话、姓名进行快速的查询查找，因此可以为这三个进行索引的添加。值得说明的是，姓名可能重名，但是电话、身份证号不会重复，他们两个可以设为唯一索引。截图可见：

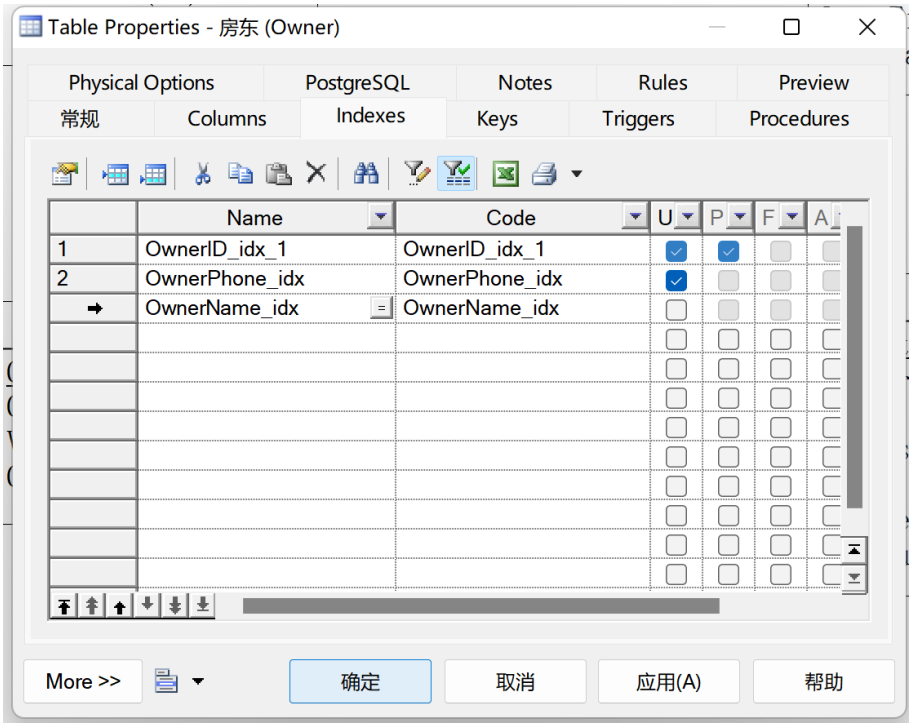

图 16：房东表的索引设计

同理，可以对租户的索引设计如下：

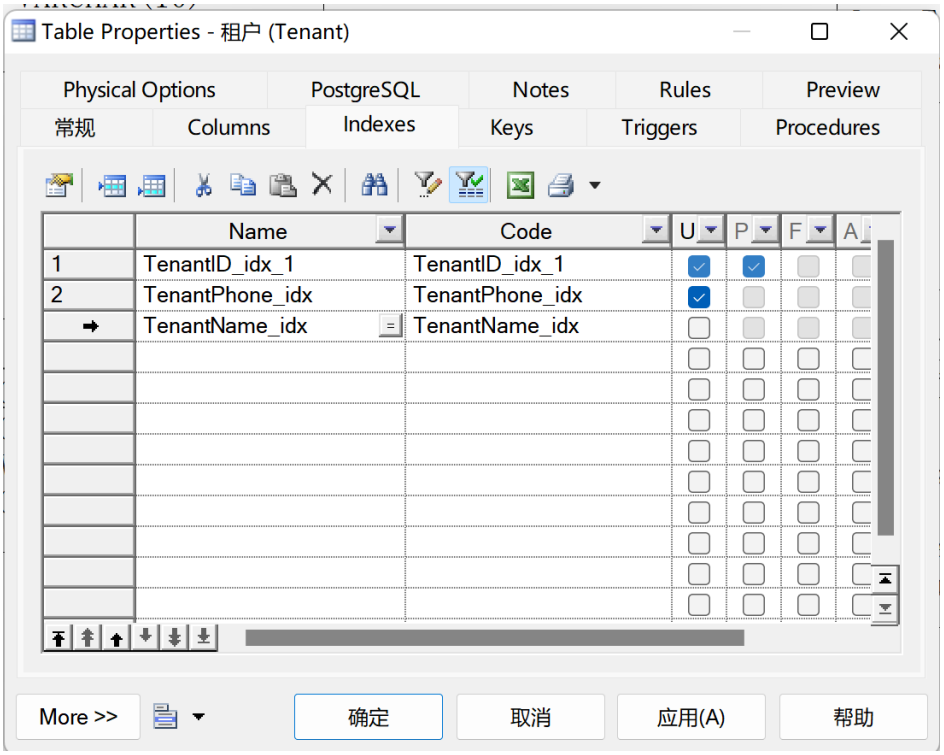

图 17：租户表的索引设计

对于租赁合同表，我们为主键（合同编号）以及两个外键（房产编号、租户身份证号）设置索引，这些是在实际业务中非常常用的查询条目，因此应当设计索引。截图见下页：

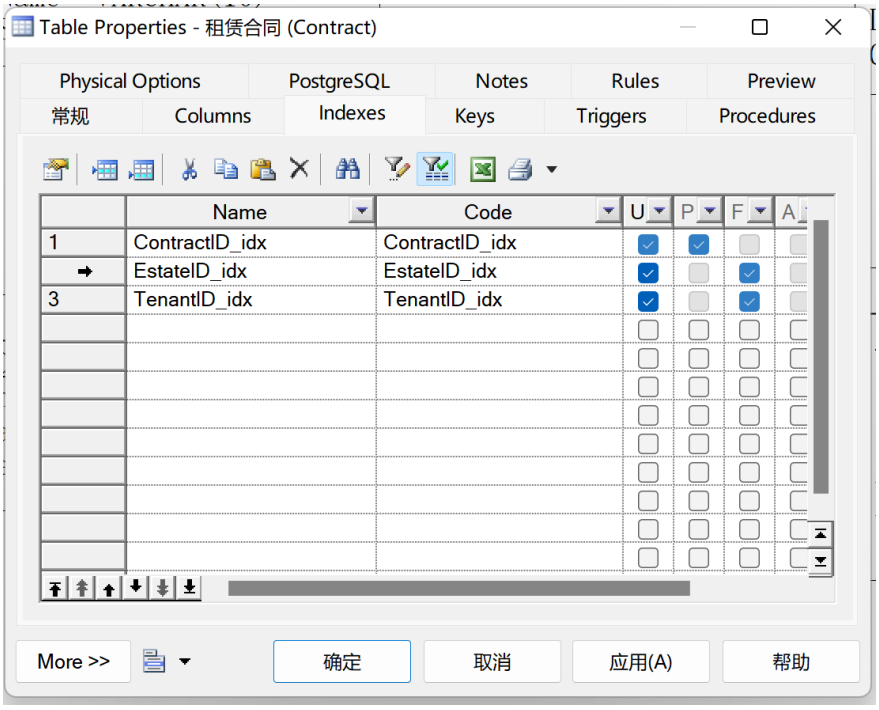

图 18：租赁合同表的索引设计

同样的，需要为房产表的主键（房产编号）和外键（房东身份证号）设置索引，这样可以快速定位房东的房屋。

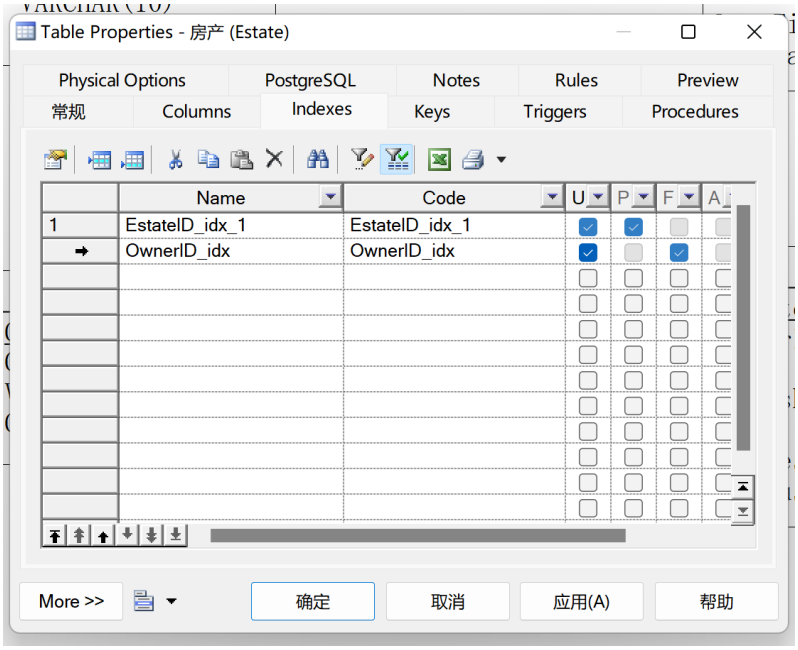

图 19：房产表的索引设计

下面为系统设计视图，视图可以很方便的为用户提供查询的方法，他与原本的数据相“隔离”，使得用户无法查看无权查看的内容。

我为系统设计了三张视图，分别为——匹配信息视图、合同详细信息视图、房产详细信息视图。

以房产详细信息视图为例子，了解一个房产，不仅需要知道其基本信息（如面积、类型）必要时，还需要提供户主的信息以供进一步了解。在一些租房 APP 上，常常附带了房主的电话号码以便与他们联系。因此可以设置如下的查询视图：

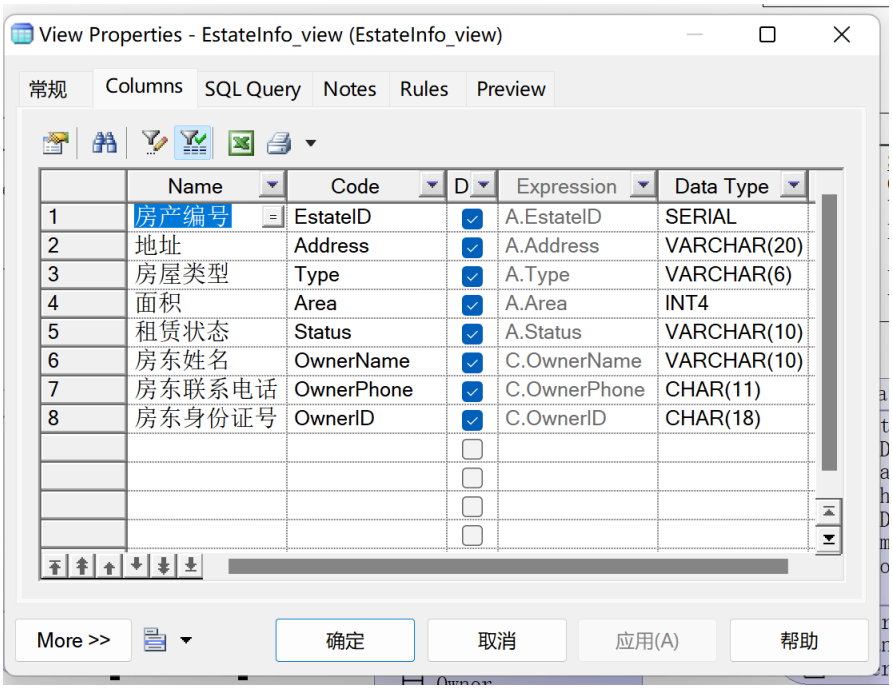

图 20：房产信息视图

同理，当查询一个合同的条目时，其房产的基本信息也需要纳入合同条款的内容中，因此，合同详细信息视图可以为他们做一个统一：

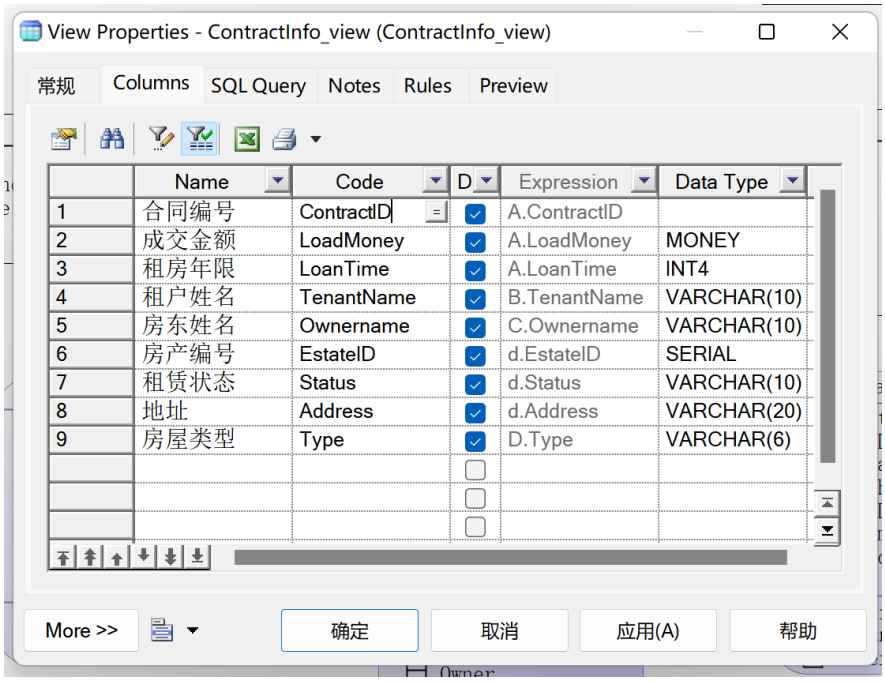

图 21：合同详细信息视图

当交易双方达成一致或有分歧时，匹配信息视图可以快速找到某个合同下交易双方的基本信息。简化业务查询需求：

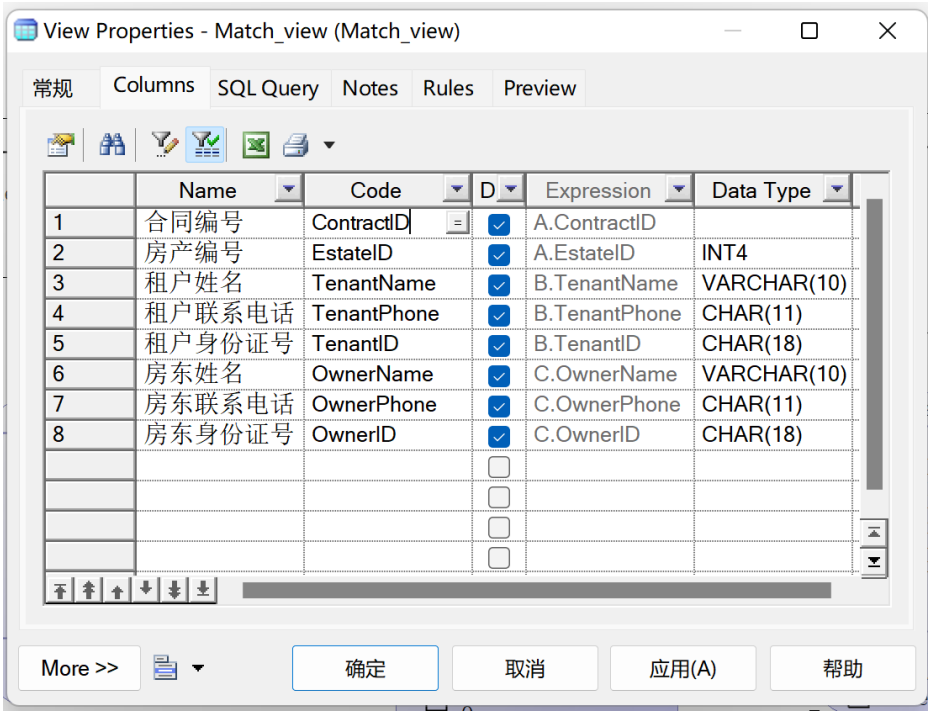

图 22：匹配信息视图

综上，可以设计如下三张视图

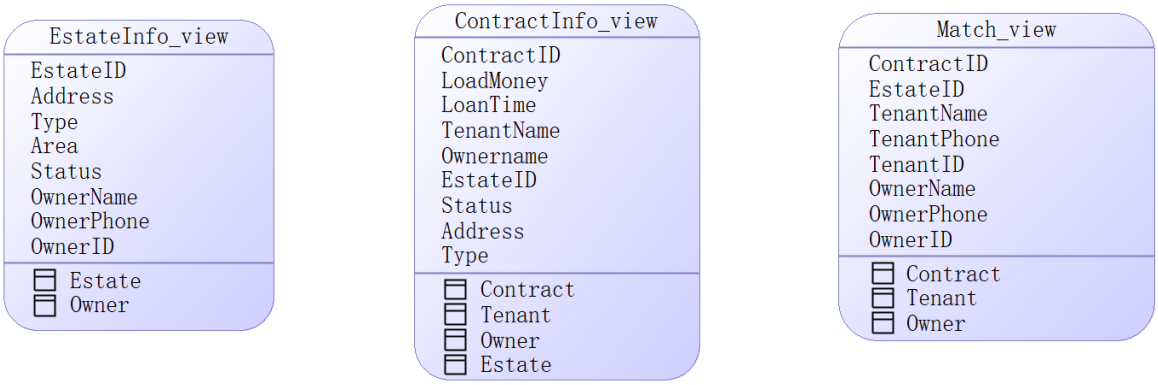

图 23：视图总览

综上，可以得到如下一页图所示的物理数据模型：

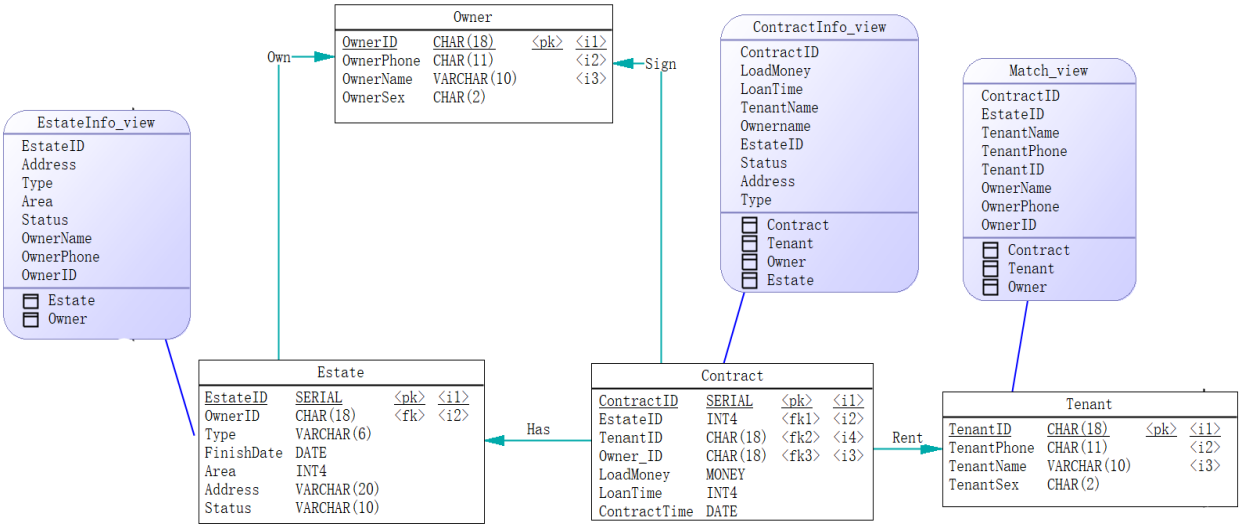

图 24：修改后的物理数据模型（PDM）

其中蓝底方框为视图，白底方框为表，表中形如<i1>的标示为索引。

2.4 脚本转换

使用 Power Designer 进行数据库脚本生成，相关截图见下一页。

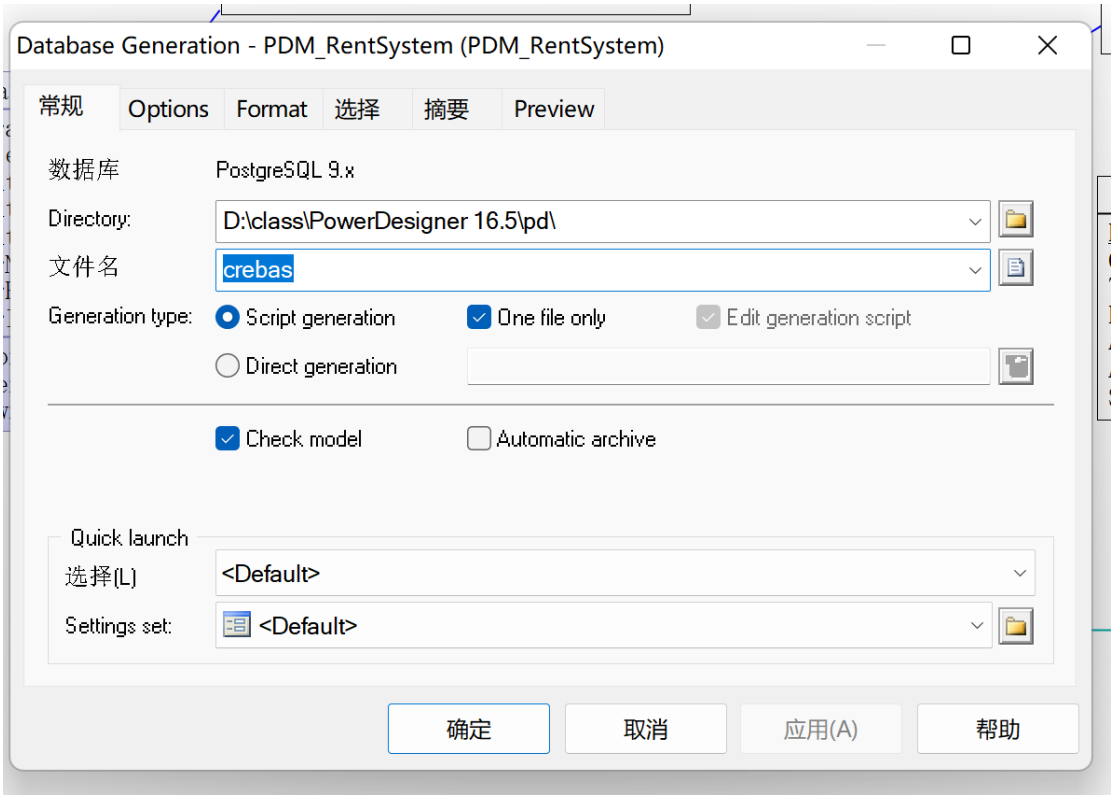

图 25：生成脚本程序

使用 VSCode 软件打开脚本，可以看到已经生成好的脚本

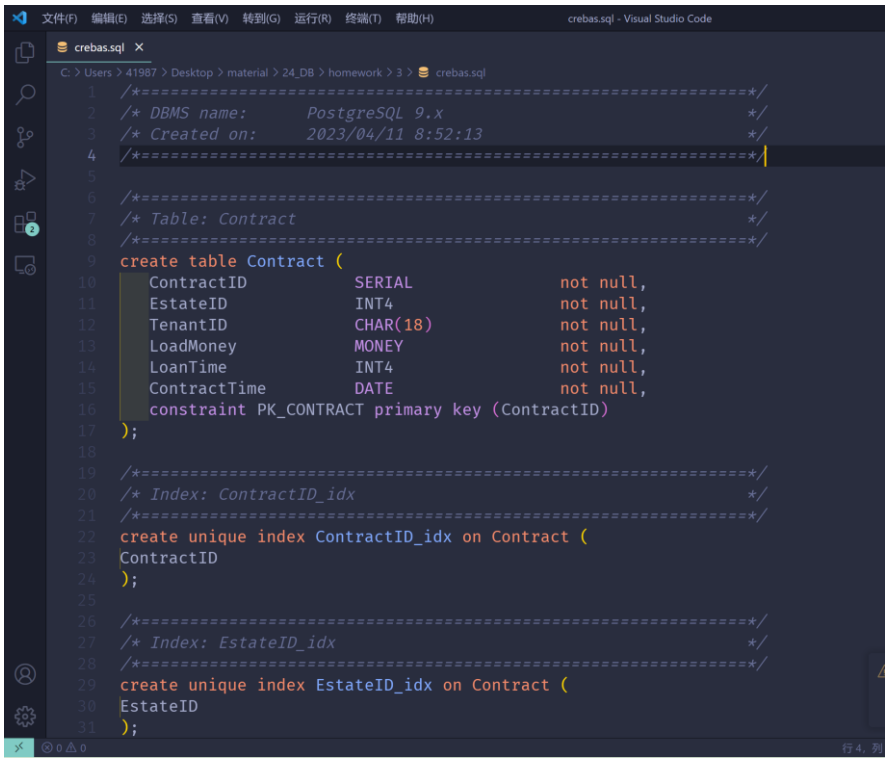

图 26：脚本程序截图（部分）

生成的完整代码因为内容太多置于文章末尾的附录中。

2.5 脚本执行

下面将脚本程序运行至 PostgreSQL 数据库中，在 pgAdmin 中打开此文件。

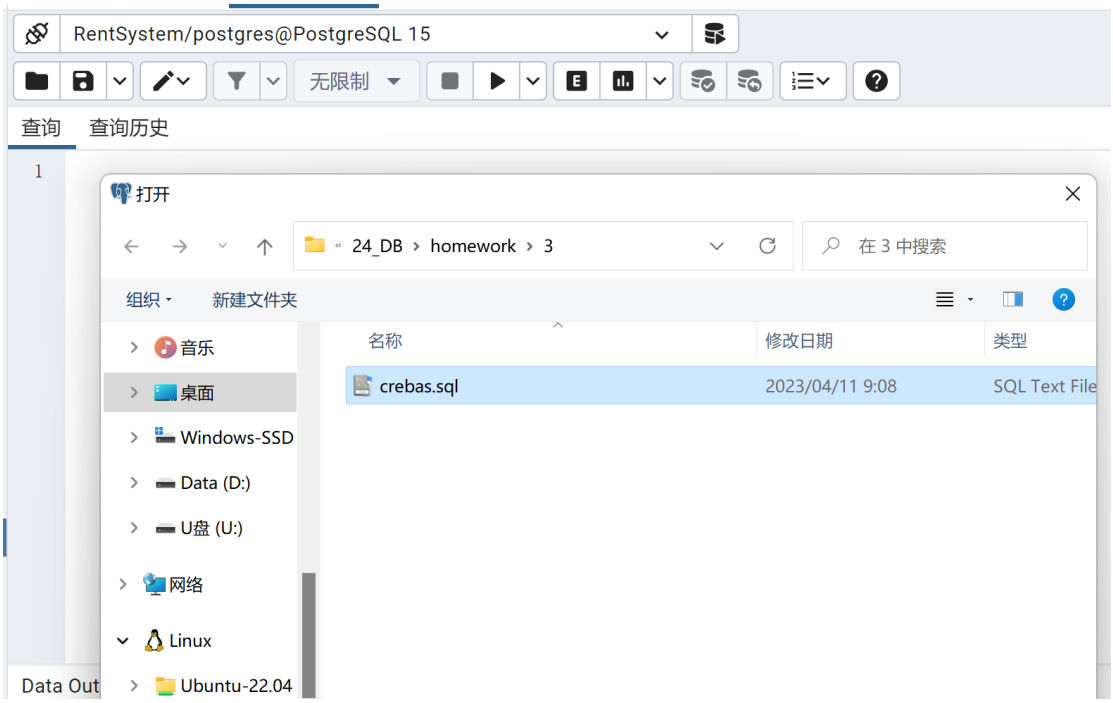

图 27：打开脚本文件导入到 pgAdmin

运行程序，可以看到无报错运行正常

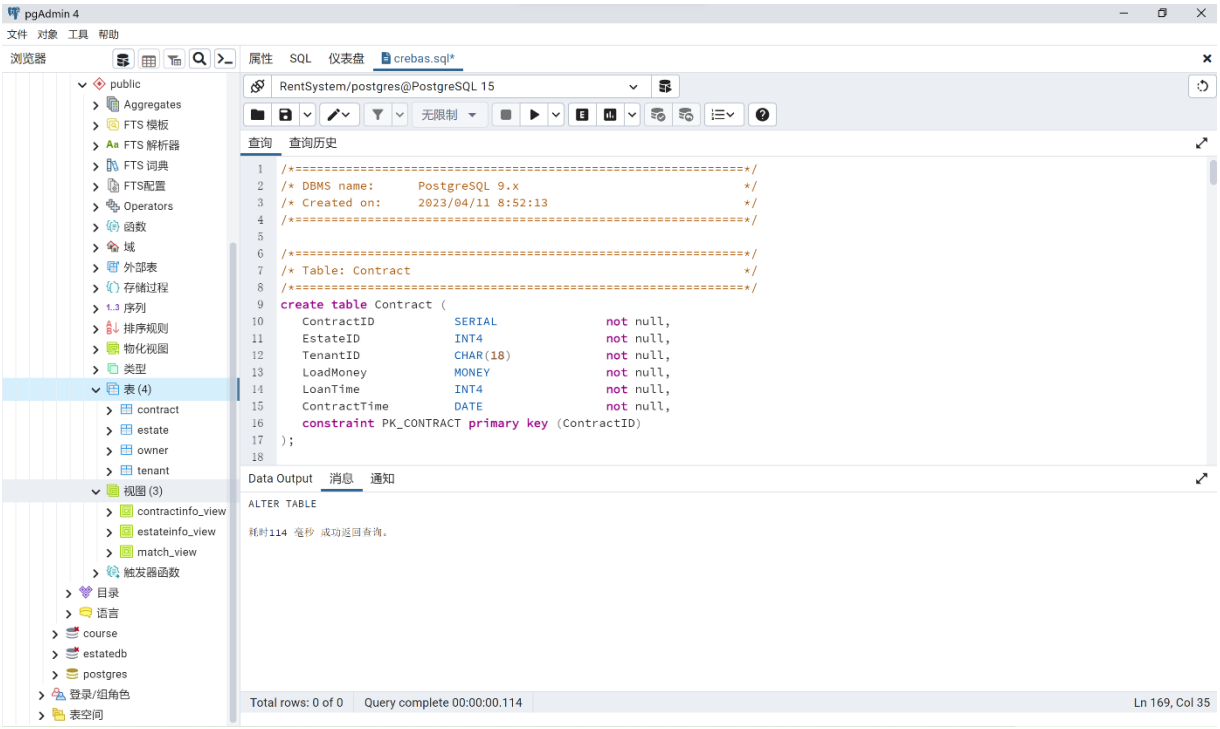

图 28：执行脚本程序

在上图左侧可以看到我们的四张表和 3 个视图均正确的创建，表明脚本程序与预期一致。

三、 挑战性问题研讨

电商系统是较为常见的大型系统工程，他对高并发访问、高性能存取、可扩展应用的需求较高。针对电商系统的以上特点，可以考虑采用以下数据库设计与开发方案：

1 数据库的读写分离

在高并发访问场景下，读操作是远远大于写操作的。因此，可以采用数据库的读写分离技术将读和写分开处理，以提高数据库并发能力。其中，写操作的数据将存放在主数据库中，而读操作的数据则会存放在多个从数据库中。这样，通过横向扩展，数据库的性能可以得到有效提升。

当然，也可以考虑合理的创建视图，通过视图查询，减小对原始数据的访问。可以加快访问速度，同时提高查询性能

例如：在实际业务中，商品搜索的频次较高，但是对其实时性要求较低（即新商品上架不一定要求立刻展现）。我们可以对一些较为常用的商品属性条目进行视图创建，在查询时只需检索视图即可；也可以为商品数据创建多个相同的数据库，其中设置一个唯一主库用于写入操作，其余多个为从库，专用于读写操作。用户查询时仅需访问从库数据进行查询，不会对主库进行干扰，提升查询效率。主从库之间设置定时器进行数据同步。

2 数据库的分库分表分区

为了应对高性能存取以及可扩展的需求，在商品销售数据库设计时可以采用分库分表技术，将数据划分到不同的数据库和表中。这样可以降低单一数据库的压力，提高查询性能。同时，分库分表也可以增加数据库的数据安全性和可靠性，避免数据丢失。

例如：我们将商品数据拆分成多个表，例如按照不同的类别或者按照不同的地区进行划分。每个表单独处理自己的数据，这样可以有效地减少单个表的负载，提高整个系统的性能。

也可以考虑将用户数据和订单数据分别存储在不同的库中。用户数据通常比较少更新，因此可以使用主从复制的方式来将用户数据同步到从库中。订单数据则需要支持高并发访问和高可用性，我们可以采用分布式数据库的方式，将订单数据分散到多个服务器上，以提高系统的可扩展性和容错性。

3 使用缓存技术

在电商系统中，商品信息等数据相对稳定，可以采用缓存技术来提高访问速度。将这类数据缓存到内存中，可以避免频繁访问数据库，减轻数据库负担，同

时提高系统的响应速度和稳定性。

常常采用分布式数据库技术进行内容分布式分发，也可以结合计算机网络的CDN 技术进行数据报文缓存。

进一步，可以使用分层缓存策略。将缓存数据按照不同的优先级进行分类。例如，可以将最常访问的数据放在高速缓存中，访问较少的数据放在低速缓存中。可以更加提高性能需求。

例如：将商品信息、用户信息和购物车信息等经常被访问的数据存储到内存缓存中，从而减少数据库的访问次数。同时，我们可以使用分布式缓存技术，将缓存分散到多个服务器上，以提高系统的可扩展性。

4　优化数据库索引

针对商品销售数据库设计，可以根据实际需求进行索引优化，提高查询性能。可以基于查询的频率和业务逻辑的复杂度来决定创建哪些索引以及优化的方式。

当然，过度索引会导致查询性能下降，并且会增加维护的成本。因此，应该只在需要的列上创建索引，避免创建不必要的索引，减轻系统负担。

例如：在实际业务中，需要经常查询订单列表并按照不同的条件进行排序和筛选。以按照用户 ID 和订单状态筛选订单为例，此时可以在订单表中创建一个用户 ID 和订单状态的联合索引。

5　数据库集群化处理

由于电商系统需要处理大量数据，单个数据库的访问压力会非常大。因此，可以采用数据库集群化的方式来提升数据库的性能和可扩展性。

数据库集群化是一种将多个数据库服务器组成一个集群，提高数据库性能和可扩展性的技术。在数据库集群中，数据通常被分散存储在多个节点，节点之间通过网络来相互通信和传递数据。因此当涉及到处理请求时，每个节点只需要处理自己对应的部分请求，其余请求进行转发，从而减轻了单个节点的访问压力。

若某个节点故障或者出现问题，其他节点仍然可以继续提供服务。此外，数据库集群还可以通过负载均衡技术来分配请求，使得所有节点的负载大致相等，从而更好地利用系统资源，提升整个系统的性能。这为电商系统的开发提供了强有力的支持。

一个非常实际的例证即阿里巴巴旗下的天猫集团也使用了上述四点类似的数据库设计原理。从网络的公开信息可以了解到，他们将数据库进行垂直分区，使用了全局缓存服务和分布式缓存，同时临近大促会对整体索引结构进行优化。

这样使得他们在每年“双十一”可以抵挡数以亿计的并发请求，可以见的天猫这一电商平台数据库设计与开发的精妙之处。天猫的例子给电商平台的高并发、高性能、可扩展需求的数据库设计开发提供了很好的标杆。

四、 心得体会与感悟

通过本次大作业与实践研究，收获颇多。

首先，我们通过对工程伦理、道德规范、法律问题的了解，探究了数据库从业人员的相关需求和规范。同时告诫我们，在未来的开发过程中下，一定要遵循这些规范制度，不要擅作主张、越过红线。

同时，也了解到数据库安全技术对于隐私数据保护和敏感数据防护的重要性。一个良好的数据库系统一定离不开数据库安全系统的加持与辅助。

随后，通过实践设计了房屋租赁管理系统。感悟到“凡事预则立”的道理，在数据库开发之前，一个好的设计方案是必不可少的。设计可以加快数据库系统的成型，规避后续的一些逻辑问题与漏洞，极大的提高系统开发效率。在实现数据库系统时，不能“鲁莽地”上手编写 SQL 语句，而是应让设计先行。与此同时，也实践了学会了用 Power Designer 对数据库进行建模，

最后，通过对一个电商平台的需求设计，了解到传统电商平台对于高并发、高性能、高扩展需求的初步解决方案。看到了天猫等电商平台对电商平台数据库设计的标杆作用。这些思路为现代化电商平台数据库开发提供了很好的借鉴意义。

附：Power Designer 生成的脚本程序原代码

因内容太多，所以放在了此处文章末尾

```
/*==============================================================*/
/* DBMS name:      PostgreSQL 9.x                               */
/* Created on:     2023/04/8 8:52:13                            */
/*==============================================================*/


/*==============================================================*/
/* Table: Contract                                              */
/*==============================================================*/
create table Contract (
   ContractID           SERIAL               not null,
   EstateID             INT4                 not null,
   TenantID             CHAR(18)             not null,
   LoadMoney            MONEY                not null,
   LoanTime             INT4                 not null,
```

```
   ContractTime        DATE                not null,
   constraint PK_CONTRACT primary key (ContractID)
);

/*==============================================================*/
/* Index: ContractID_idx                                        */
/*==============================================================*/
create unique index ContractID_idx on Contract (
ContractID
);

/*==============================================================*/
/* Index: EstateID_idx                                          */
/*==============================================================*/
create unique index EstateID_idx on Contract (
EstateID
);

/*==============================================================*/
/* Index: TenantID_idx                                          */
/*==============================================================*/
create unique index TenantID_idx on Contract (
TenantID
);

/*==============================================================*/
/* Table: Estate                                                */
/*==============================================================*/
create table Estate (
   EstateID            SERIAL              not null,
   OwnerID             CHAR(18)            not null,
   Type                VARCHAR(6)          not null,
   FinishDate          DATE                not null,
   Area                INT4                not null,
   Address             VARCHAR(20)         not null,
   Status              VARCHAR(10)         null,
   constraint PK_ESTATE primary key (EstateID)
);

/*==============================================================*/
/* Index: EstateID_idx_1                                        */
/*==============================================================*/
create unique index EstateID_idx_1 on Estate (
EstateID
```

```
);

/*==============================================================*/
/* Index: OwnerID_idx                                           */
/*==============================================================*/
create unique index OwnerID_idx on Estate (
OwnerID
);

/*==============================================================*/
/* Table: Owner                                                 */
/*==============================================================*/
create table Owner (
   OwnerID              CHAR(18)            not null,
   OwnerPhone           CHAR(11)            not null,
   OwnerName            VARCHAR(10)         not null,
   OwnerSex             CHAR(2)             not null,
   constraint PK_OWNER primary key (OwnerID)
);

/*==============================================================*/
/* Index: OwnerID_idx_1                                         */
/*==============================================================*/
create unique index OwnerID_idx_1 on Owner (OwnerID);

/*==============================================================*/
/* Index: OwnerPhone_idx                                        */
/*==============================================================*/
create unique index OwnerPhone_idx on Owner (OwnerPhone);

/*==============================================================*/
/* Index: OwnerName_idx                                         */
/*==============================================================*/
create  index OwnerName_idx on Owner (OwnerName);

/*==============================================================*/
/* Table: Tenant                                                */
/*==============================================================*/
create table Tenant (
   TenantID             CHAR(18)            not null,
   TenantPhone          CHAR(11)            not null,
   TenantName           VARCHAR(10)         not null,
   TenantSex            CHAR(2)             not null,
   constraint PK_TENANT primary key (TenantID)
```

```
);

/*==============================================================*/
/* Index: TenantID_idx_1                                        */
/*==============================================================*/
create unique index TenantID_idx_1 on Tenant (TenantID);

/*==============================================================*/
/* Index: TenantPhone_idx                                       */
/*==============================================================*/
create unique index TenantPhone_idx on Tenant (TenantPhone);

/*==============================================================*/
/* Index: TenantName_idx                                        */
/*==============================================================*/
create  index TenantName_idx on Tenant (TenantName);

/*==============================================================*/
/* View: ContractInfo_view                                      */
/*==============================================================*/
create or replace view ContractInfo_view as
select A.ContractID,A.LoadMoney,A.LoanTime, B.TenantName,
C.Ownername,D.EstateID,D.Status,D.Address,D.Type
from Contract as A,Tenant as B,Owner as C,Estate as D;

/*==============================================================*/
/* View: EstateInfo_view                                        */
/*==============================================================*/
create or replace view EstateInfo_view as
select
A.EstateID,A.Address,A.Type,A.Area,A.Status,
C.OwnerName,C.OwnerPhone,C.OwnerID
from Estate as A,Owner as C;

/*==============================================================*/
/* View: Match_view                                            */
/*==============================================================*/
create or replace view Match_view as
select
A.ContractID,A.EstateID,
B.TenantName,B.TenantPhone,B.TenantID,
C.OwnerName,C.OwnerPhone,C.OwnerID
from Contract as A,Tenant as B,Owner as C;
```

```
alter table Contract
   add constraint FK_CONTRACT_HAS_ESTATE foreign key (EstateID)
      references Estate (EstateID)
      on delete restrict on update restrict;

alter table Contract
   add constraint FK_CONTRACT_RENT_TENANT foreign key (TenantID)
      references Tenant (TenantID)
      on delete restrict on update restrict;

alter table Estate
   add constraint FK_ESTATE_OWN_OWNER foreign key (OwnerID)
      references Owner (OwnerID)
      on delete restrict on update restrict;
```