



National University of Computer & Emerging Sciences, Karachi
Artificial Intelligence-FAST School of Computing Department
Fall 2024, Lab Manual - 07



Course Code- CL-2005	Course - Database Systems Lab
Instructor(s) -	Mehak Mazhar

Relational Modeling

SQL Data Modeler

Oracle SQL Developer Data Modelers a standalone, independent product, available for download from the Oracle Technology Network (OTN).SQL Developer Data Modeler runs on Windows, Linux and MacOS X.

Logical Models

The logical model in SQL Developer Data Modeler includes standard logical modeling facilities, such as drawing entities and relationships etc.

Relational Models

The SQL Developer Data Modeler relational model is an intermediate model between the logical model and the physical models. It supports relational design decisions independent of the constraints of the target physical platform(s). All many-to-many relationships and all super type/sub-types entity hierarchies are resolved during forward engineering (transformation) of the logical model, or part of it, to a relational model.

Physical Model

A physical data model defines all of the logical database components and services that are required to build a database or can be the layout of an existing database.

A physical data model consists of the table's structure, column names and values, foreign and primary keys and the relationships among the tables.

Data Modeling for a Small Database

We will use SQL Developer Data Modeler to create models for a simplified library database Entities Included will be-

- Books
- Patrons (people who have library cards)
- Transactions (checking a book out, returning a book, and so on).

We are using only a subset of the possible steps for the Top-Down Modeling approach.

1. Develop the Logical Model.
2. Develop the Relational Model.
3. Generate DDL.
4. Save the Design.

1. Develop the logical model

The logical model for the data base includes three entities-

Books (describes each book in the library),

Patrons (describes each person who has a library card), and

Transactions (describes each transaction involving a patron and a book).

Before we create the entities, we will create some domains that will make the entity creation (and later DDL generation) more meaningful and specific.

Adding Domains

In planning for our data needs, we have determined that several kinds of fields will occur in multiple kinds of records, and many fields can share a definition. For example, we have decided that-

- The first and last names of persons can be up to 25 characters each.
- Street address lines can be up to 40 characters.
- City names can be up to 25 characters.
- State codes (United States) are 2-character standard abbreviations.
- Zip codes (United States postal codes) can be up to 10 characters (nnnnn-nnnn).
- Book identifiers can be up to 20 characters.
- Other identifiers are numeric, with up to 7 digits (no decimal places). Titles (books, articles, and so on) can be up to 50 characters.
- These added domains will also be available after we exit Data Modeler and restart it later.

Steps to add domains-

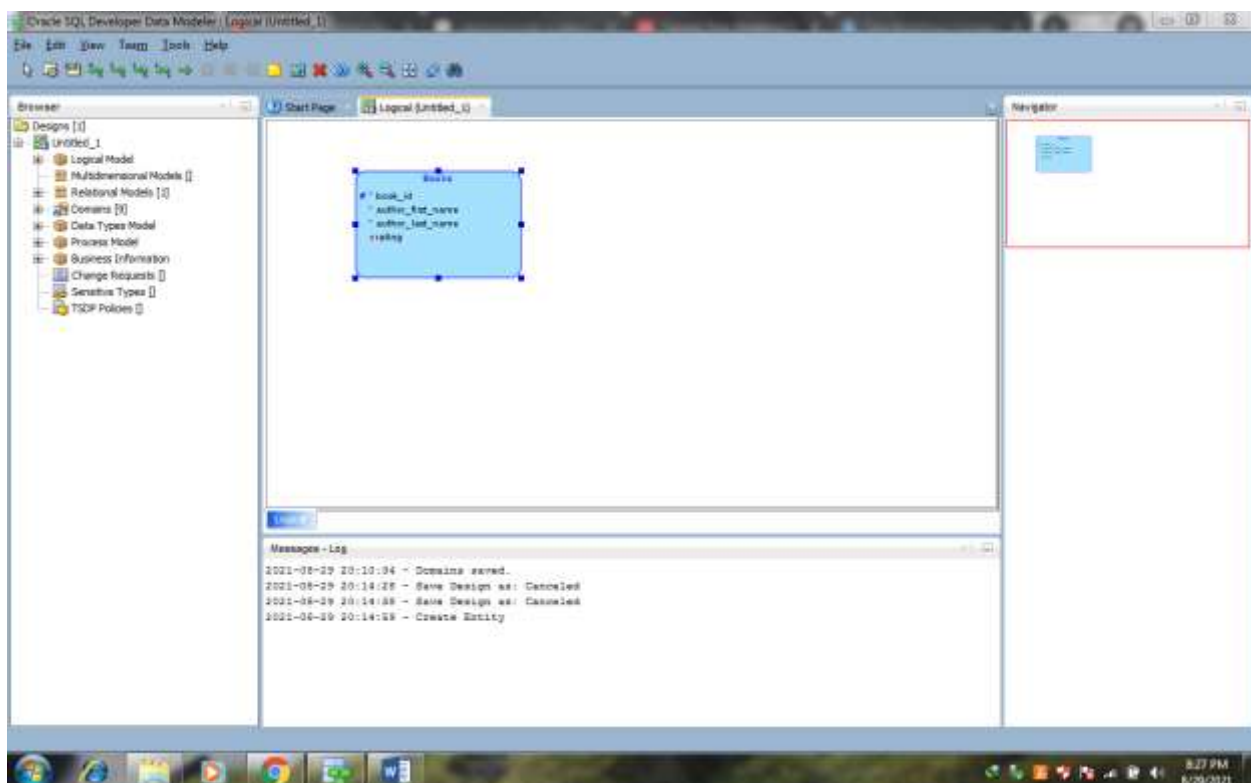
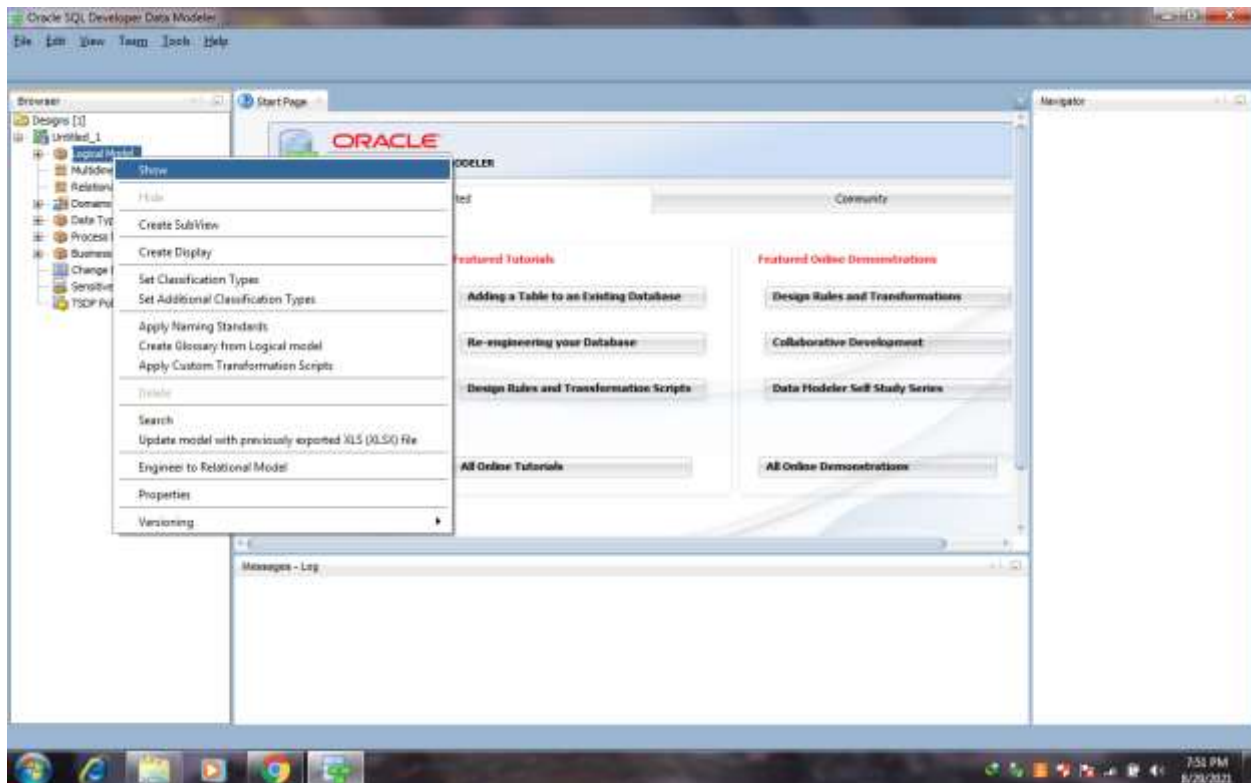
1. Click Tools, then Domains Administration.
2. In the Domains Administration dialog box, add domains with the following definitions. Click Add to start each definition, and click Apply after each definition.
3. When we have finished defining these domains, click Save. This creates a file named default domains.xml in the domains directory (folder) under the location where we installed Data Modeler.
4. Optionally, copy the default domains.xml file to a new location (not under the Data Modeler installation directory), and give it an appropriate name, such as library_domains.xml. We can then import domains from that file when we create other designs.
5. Click Close to close the dialog box.

Creating Entities

Creating the book Entity-

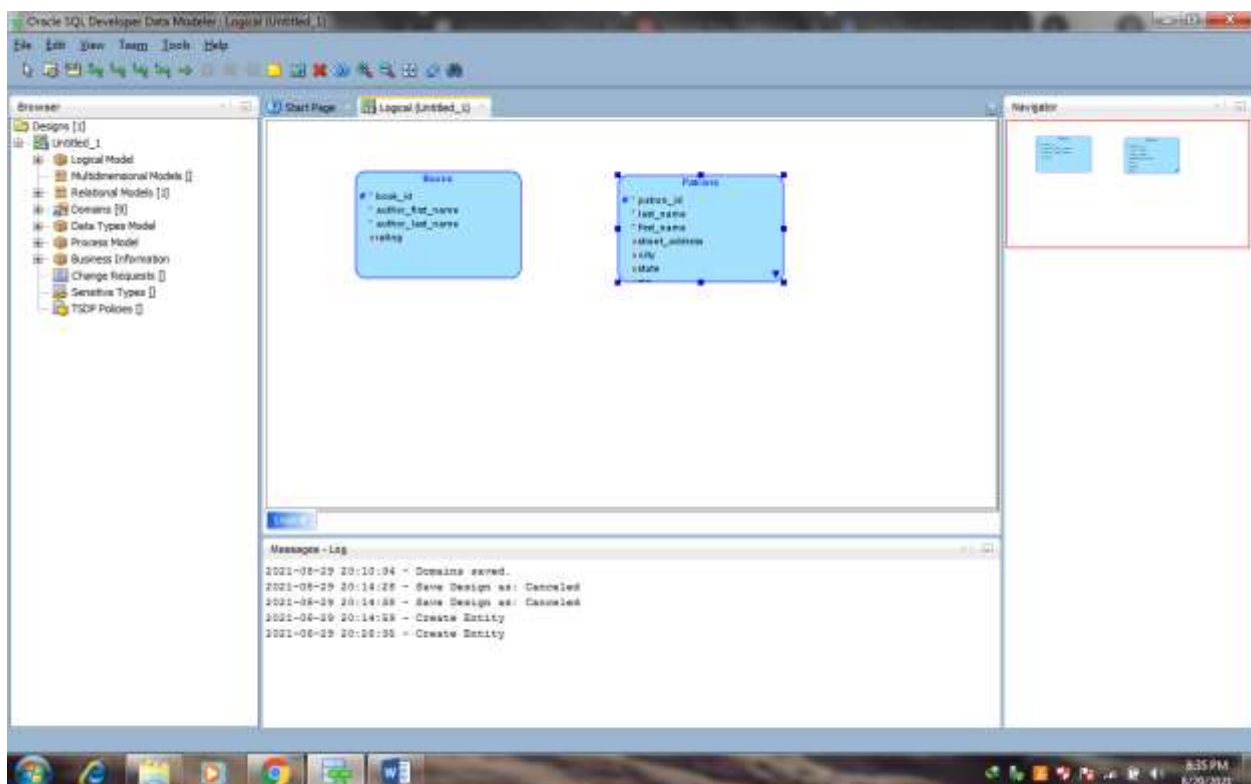
- Create the Books entity as follows-
- In the main area (right side) of the SQL Developer Data Modeler window, click the Logical tab.
- Click the new Entity icon.
- Click in the logical model pane in the main area; and in the Logical pane press, Diagonally drag, and release the mouse button to draw an entity box. The Entity Properties dialog box is displayed.
- Click General on the left, and specify as follows-
- Name- Books
- Click Attributes on the left, and use the Add (+) icon to add the following attributes, one at a time. (For data types, select from the Domain types except or Rating, which is a Logical type.)

Name	Datatype	Other Information and Notes
book_id	Domain: Book Id	Primary UID (unique identifier). (The Dewey code or other book identifier.)
title	Domain: Title	M (mandatory, that is, must not be null).
author_last_name	Domain: Person Name	M (mandatory, that is, must not be null).
author_first_name	Domain: Person Name	25 characters maximum.
rating	Logical type: NUMERIC (Precision=2, Scale= 0)	(Librarian's personal rating of the book, from 1 (poor) to 10 (great).)



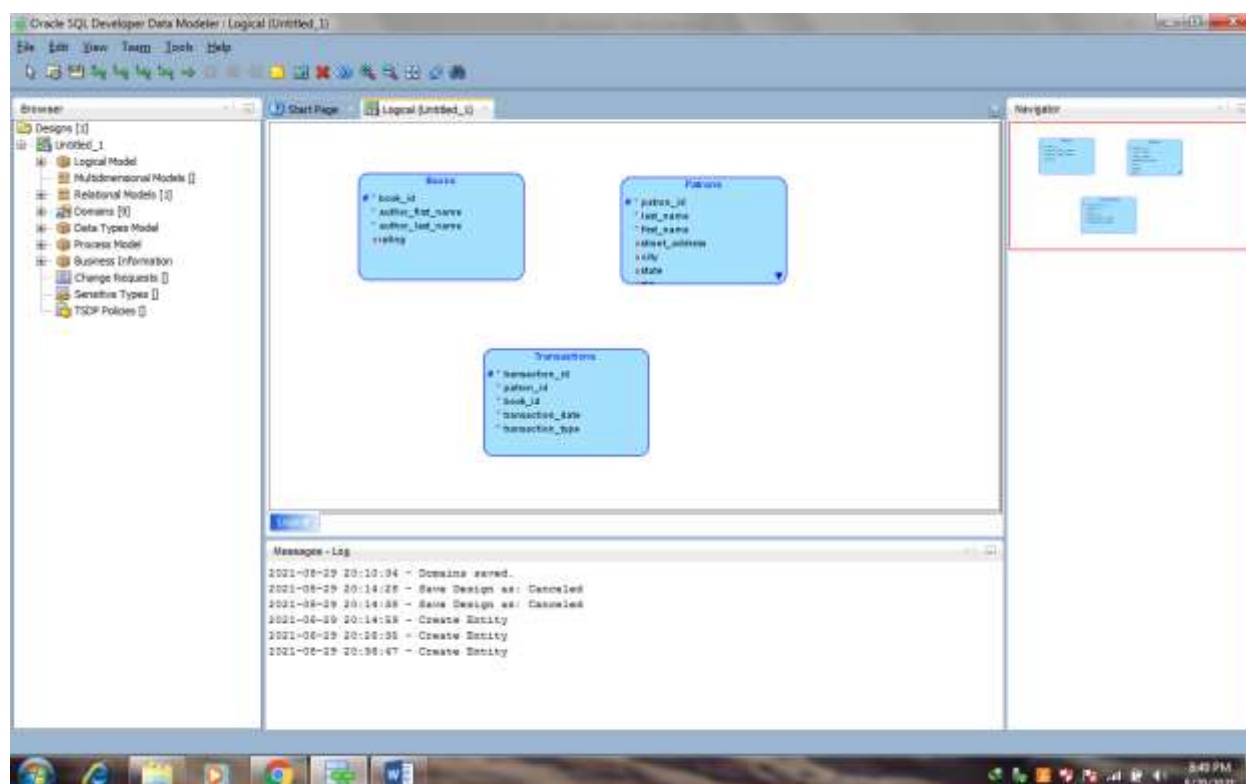
Creating the Patrons Entity

Attribute Name	Type	Other Information and Notes
patron_id	Domain: Numeric Id	Primary UID (unique identifier). (Unique patron ID number, also called the library card number.)
last_name	Domain: Person Name	M (mandatory, that is, must not be null). 25 characters maximum.
first_name	Domain: Person Name	(Patron's first name.)
street_address	Domain: Address Line	(Patron's street address.)
city	Domain: City	(City or town where the patron lives.)
state	Domain: State	(2-letter code for the state where the patron lives.)
zip	Domain: Zip	(Postal code where the patron lives.)
location	Domain: Address	Oracle Spatial geometry object representing the patron's geocoded address.



Creating the Transactions Entity

Attribute Name	Type	Other Information and Notes
transaction_id	Domain: Numeric Id	Primary UID (unique identifier). (Unique transaction ID number)
transaction_date	Logical type: Datetime	M (mandatory, that is, must not be null). Date and time of the transaction.
transaction_type	Domain: Numeric Id	M (mandatory, that is, must not be null). (Numeric code indicating the type of transaction, such as 1 for checking out a book.)



Creating Relations between Entities

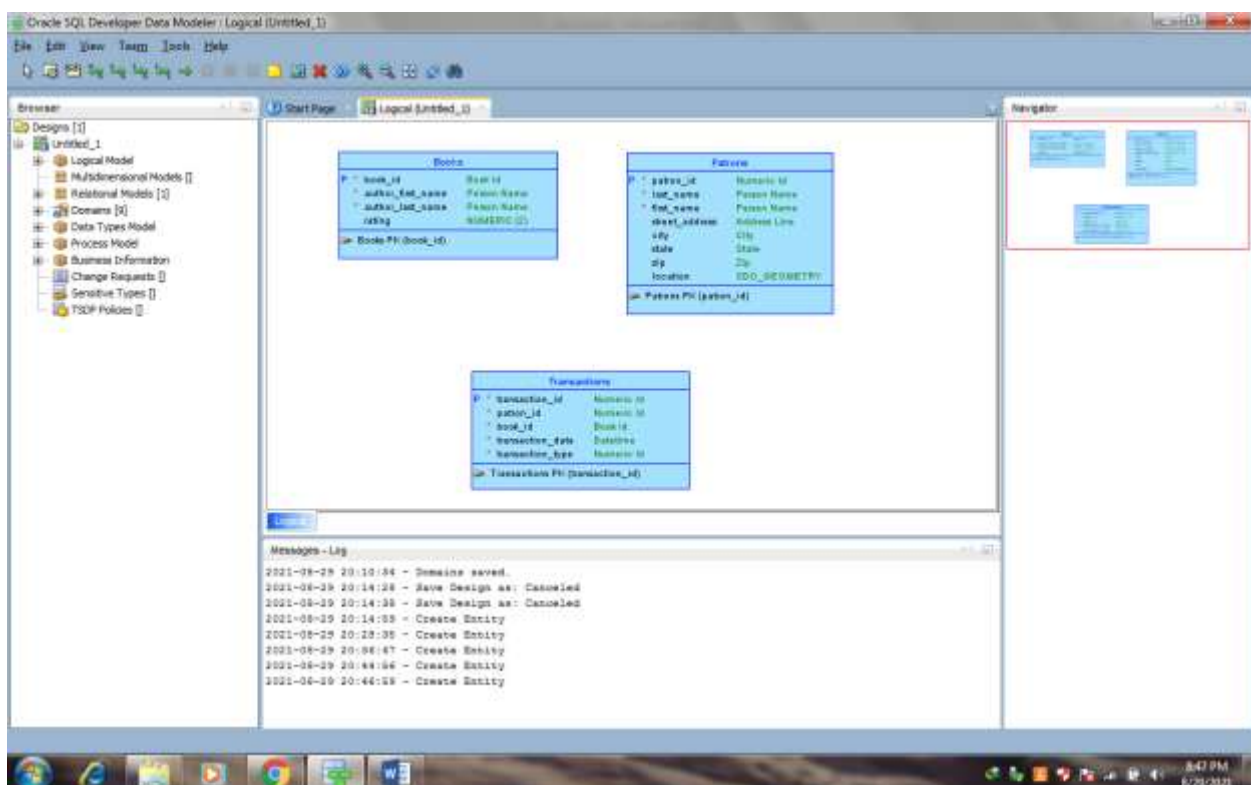
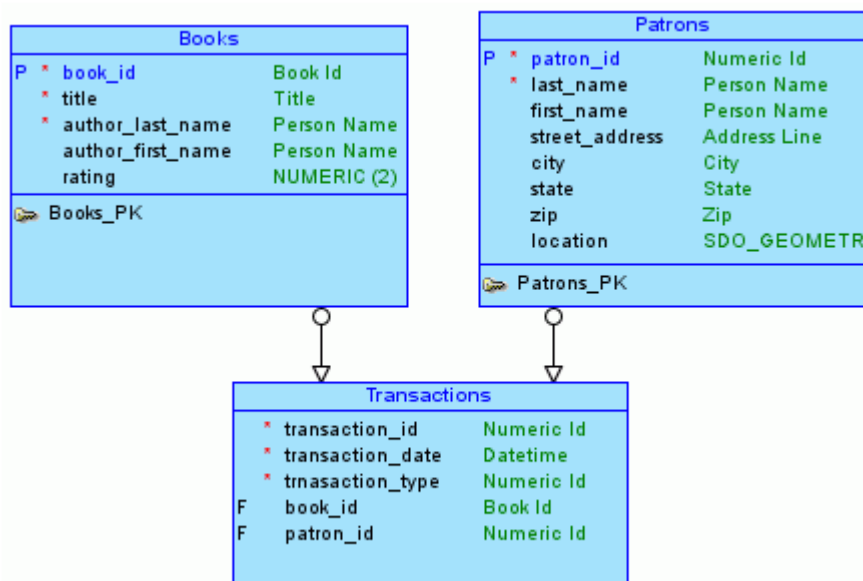
Relations show the relationships between entities- one-to-many, many-to-one, or many-to-many. The following relationships exist between the entities-

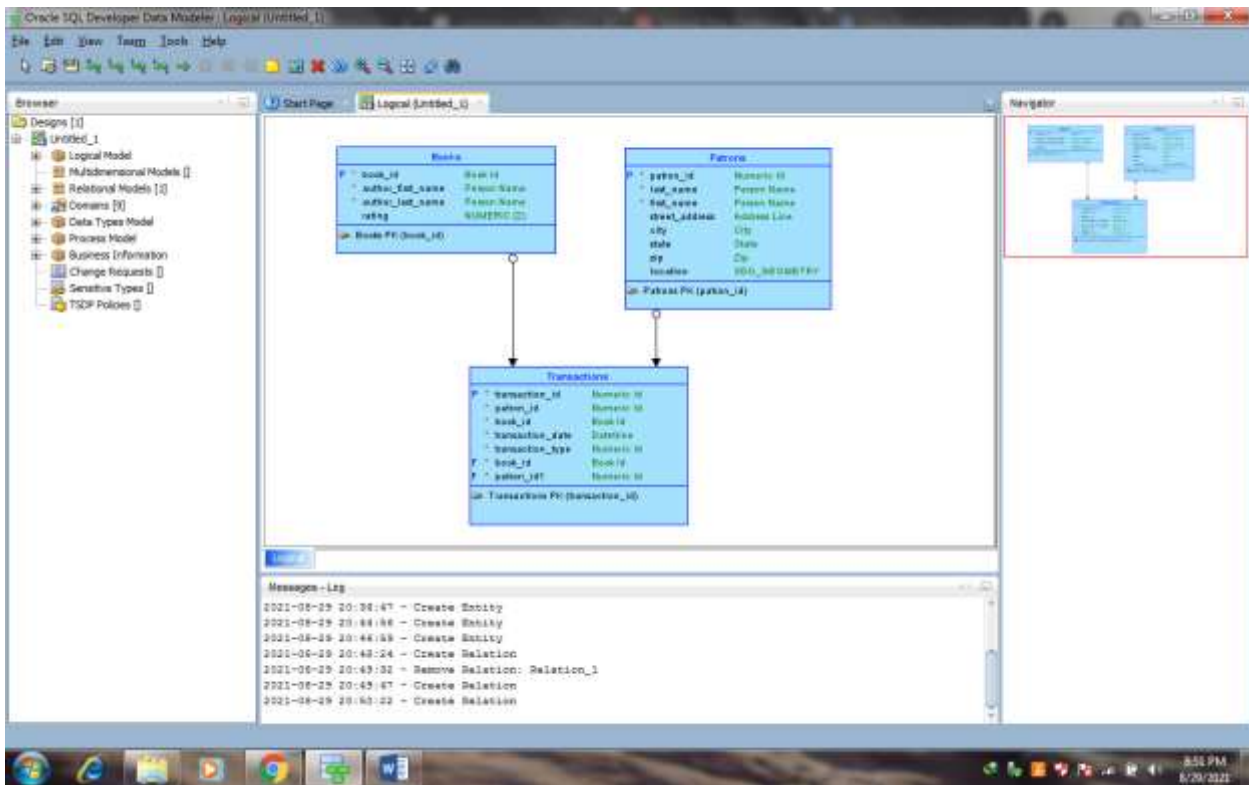
Books and Transactions- one-to-many. Each book can be involved in multiple sequential transactions. Each book can have zero or one active checkout transactions; a book that is checked out cannot be checked out again until after it has been returned.

Patrons and Transactions- one-to-many. Each patron can be involved in multiple sequential and simultaneous transactions. Each patron can check out one or many books in a visit to the library, and can have multiple active check out transactions reflecting several visits; each patron can also return checked out books at any time.

Steps to Create Relationships-

1. In the logical model pane in the main area, arrange the entity boxes as follows- Books on the left, Patrons on the right, and Transactions either between Books and Patrons or under them and in the middle. (If the pointer is still cross-hairs, click the Select icon at the top left to change the pointer to an arrow.)
2. Click the New 1- N Relation icon.
3. Click first in the Books box, then in the Transactions box. A line with an arrowhead is drawn from Books to Transactions.
4. Click the New 1- N Relation icon.
5. Click first in the Patrons box, then in the Transactions box. A line with an arrowhead is drawn from Patrons to Transactions.
6. Optionally, double-click a line (or right-click a line and select Properties) and view the Relation Properties information.



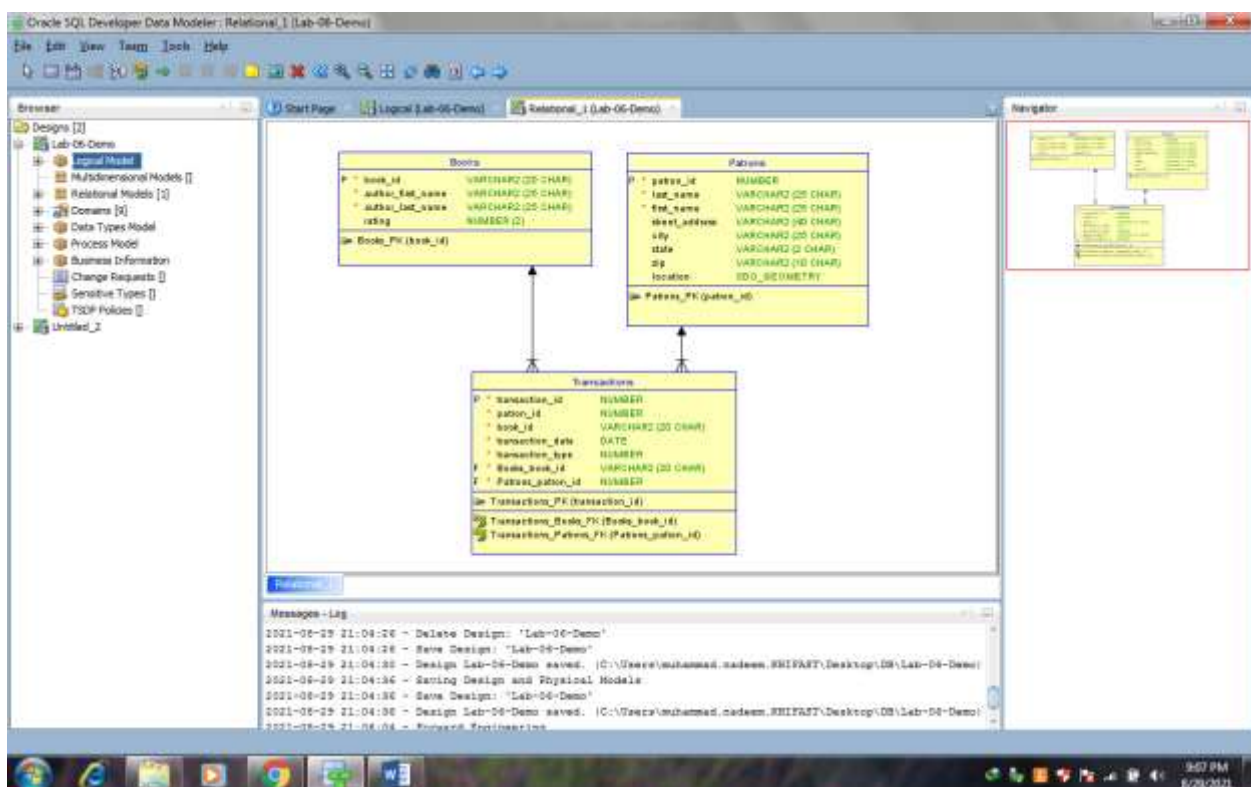
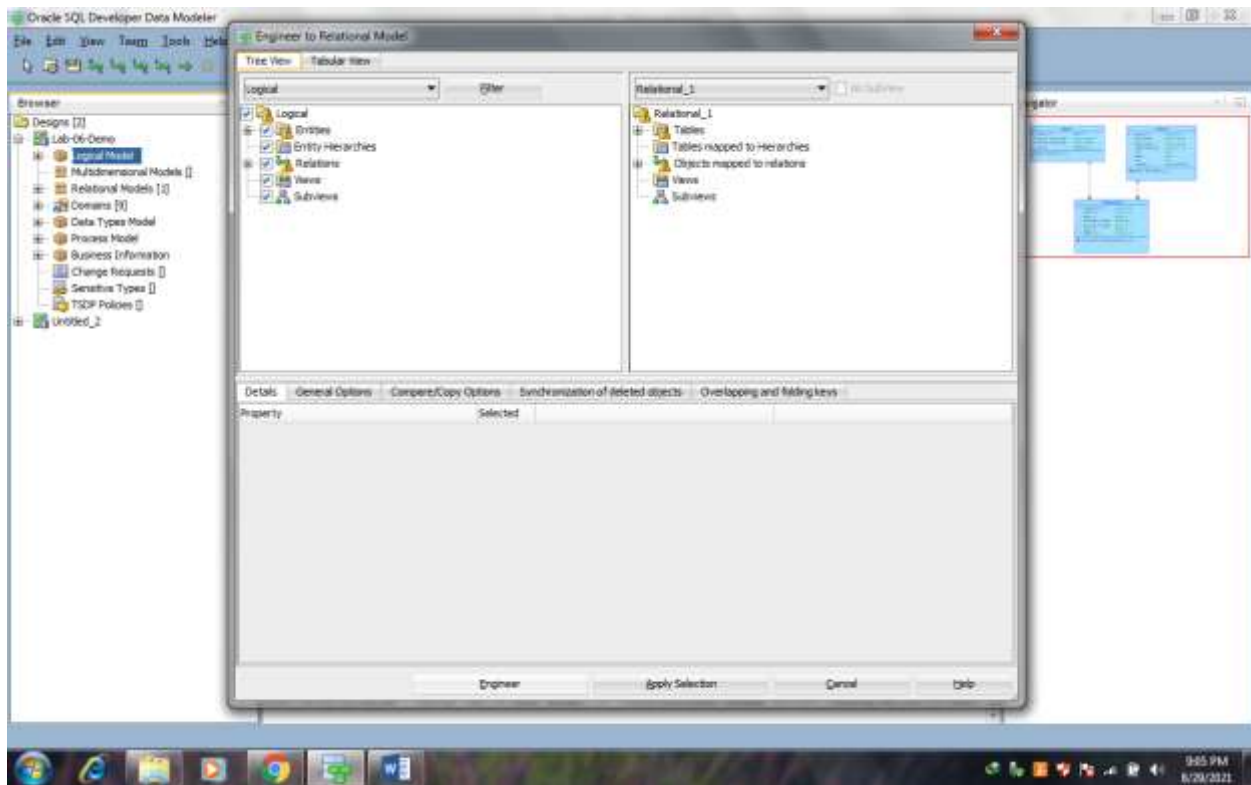


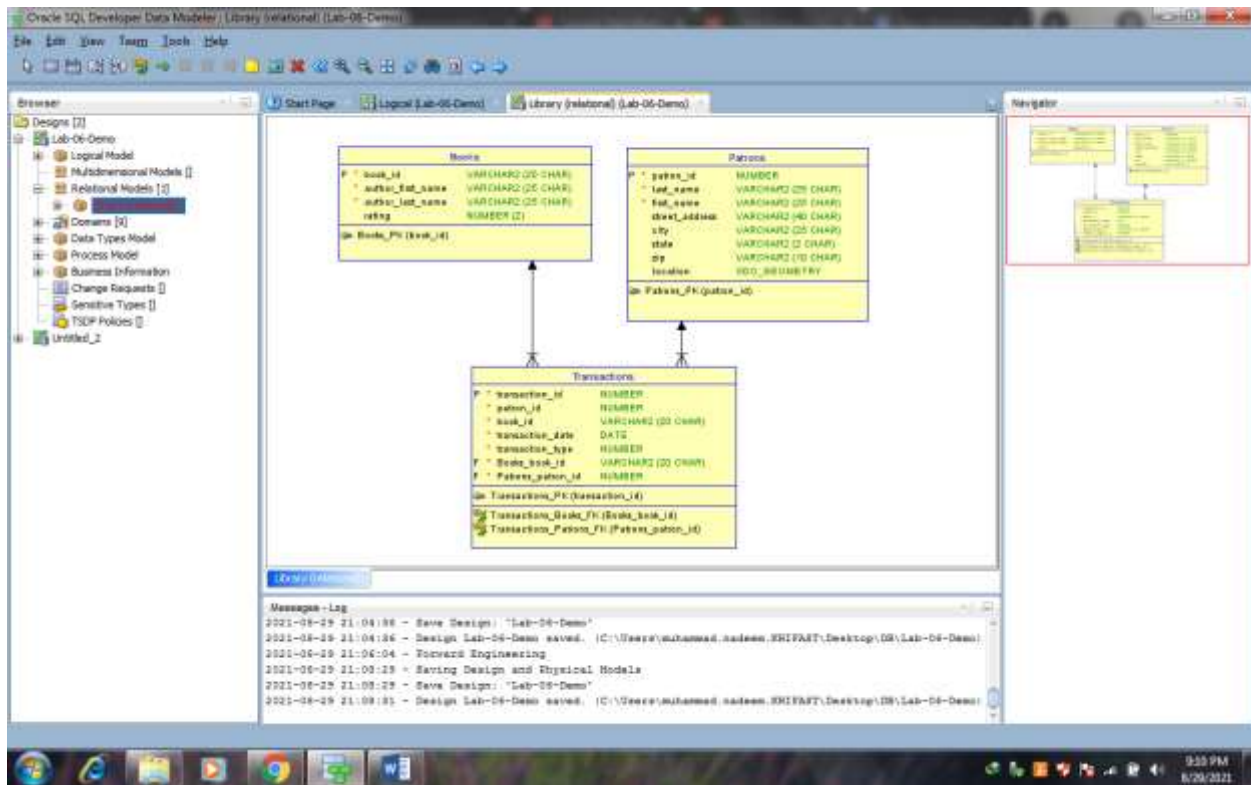
2. Develop the Relational Model

The relational model for the library tutorial data base consists of tables that reflect the entities of the logical model (Books, Patrons, and Transactions) and all attributes of each entity. In the simplified data model for this tutorial, a single relational model reflects the entire logical model; however, for other data models we can create one or more relational models, each reflecting all or a sub set of the logical model. (To have a relational model reflect a subset of the logical model, use the "filter" feature in the dialog box for engineering a relational model.).

Develop the relational model as follows-

1. With the logical model selected, click **Design**, then **Engineer to Relational Model**. The Engineering dialog box is displayed.
2. Accept all defaults (do not filter), and click **Engineer**. This causes the Relational_1 model to be populated with tables and other objects that reflect the logical model.
3. Optionally, expand the Relational Models node in the object browser on the left side of the window, and expand Relational_1 and nodes under it that contain any entries (such as Tables and Columns), to view the objects created.
4. Change the name of the relational model from Relational_1 to something more meaningful for diagram displays, such as Library(relational). Specifically, right-click Relational_1 in the hierarchy display, select **Properties**, in the General pane of the Model Properties-<name>(Relational) dialog box specify **Name** as Library(relational), and click **OK**.

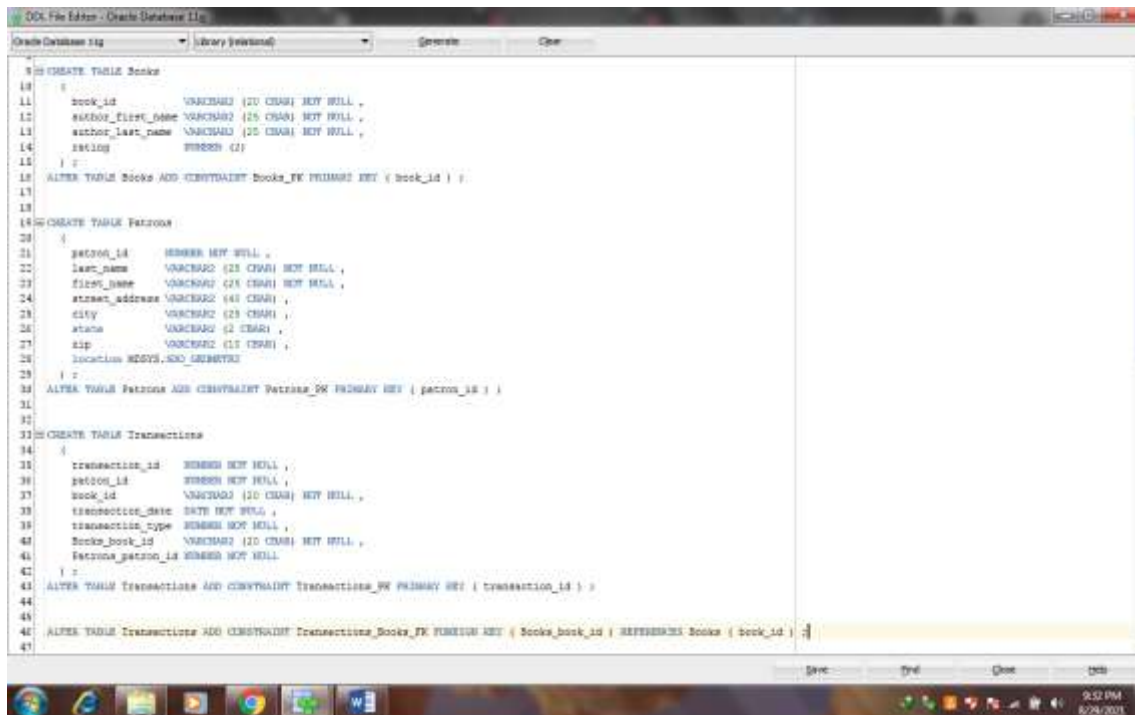




3. Generate DDL

Develop the physical model as follows-

- Optionally, view the physical model before we generate DDL statements-
 - With the Library logical model selected, click Physical, then Open Physical Model. A dialog box is displayed for selecting the type of database for which to create the physical model.
 - Specify the type of database (for example, Oracle Database11g), and click OK. In the hierarchy display on the left side of the window, a Physical Models node is added under the Library relational model node, and a physical model reflecting the type of database is created under the Physical Models node.
 - Expand the Physical Models node under Library (the relational model), and expand the newly created physical model and nodes under it that contain any entries (such as Tables and Columns), to view the objects created.
- Click File, then Export, then DDL File.
- Select the database type (for example, Oracle Database11g) and click Generate. The DDL Generation Options dialog box is displayed.
- Accept all defaults, and click OK. A DDL file editor is displayed, with SQL statements to create the tables and add constraints. (Although we can edit statements in this window, do not edit any statements for this tutorial exercise.)
- Click Save to save the statements to a SQL script file (for example, create_library_objects.sql) on were local system.
Later, run the script (for example, using a database connection and SQL Worksheet in SQL Developer) to create the objects in the desired database.
- Close to close the DDL file editor.



```

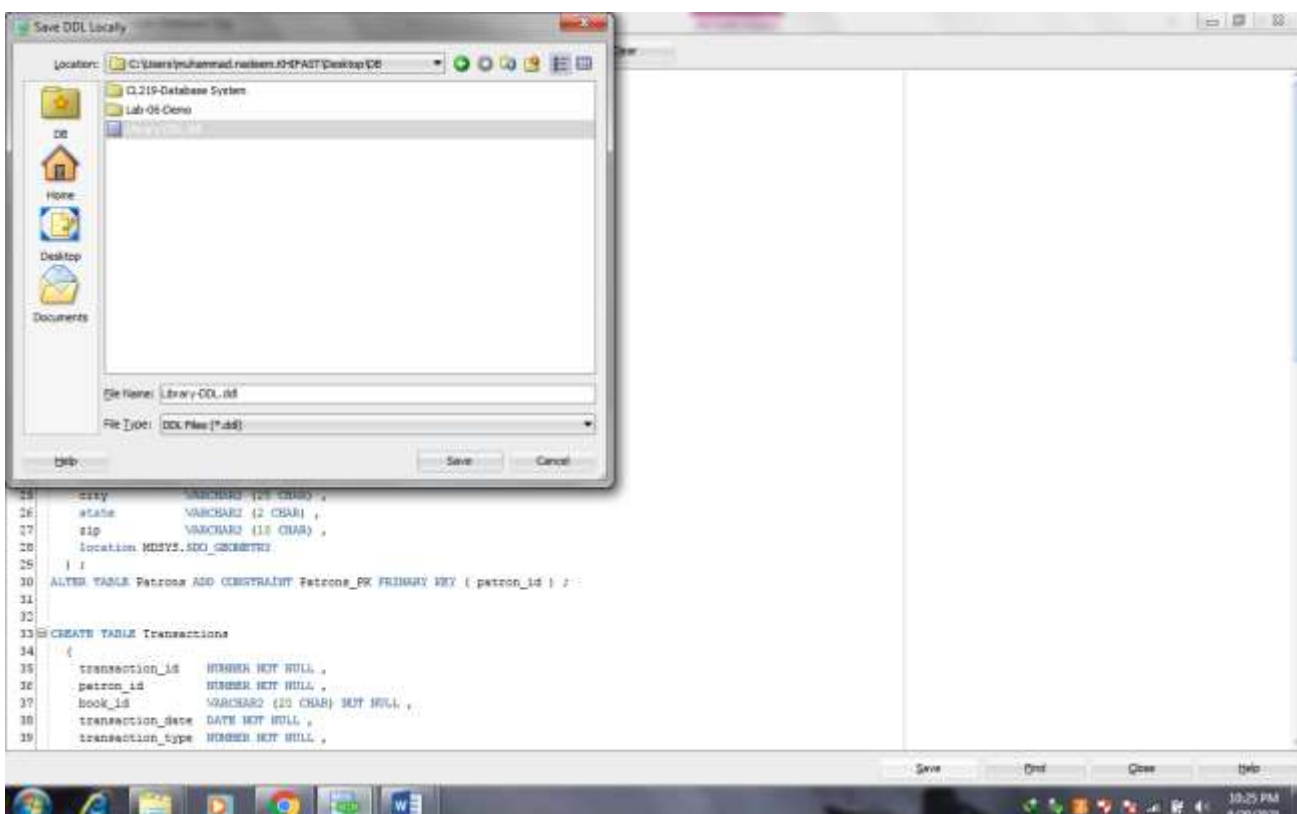
9 CREATE TABLE Books
10 (
11     book_id          VARCHAR2 (20 CHAR) NOT NULL,
12     author_first_name VARCHAR2 (25 CHAR) NOT NULL,
13     author_last_name  VARCHAR2 (25 CHAR) NOT NULL,
14     title             VARCHAR2 (40 CHAR),
15     isbn              NUMBER (12)
16 ) ;
17 ALTER TABLE Books ADD CONSTRAINT Books_PK PRIMARY KEY ( book_id ) ;
18
19 CREATE TABLE Patrons
20 (
21     patron_id         NUMBER NOT NULL,
22     last_name          VARCHAR2 (25 CHAR) NOT NULL,
23     first_name         VARCHAR2 (25 CHAR) NOT NULL,
24     street_address     VARCHAR2 (40 CHAR),
25     city               VARCHAR2 (25 CHAR),
26     state              VARCHAR2 (2 CHAR),
27     zip                VARCHAR2 (10 CHAR),
28     location           NUMBER(3,0) CHECK (location <= 3)
29 ) ;
30 ALTER TABLE Patrons ADD CONSTRAINT Patrons_PK PRIMARY KEY ( patron_id ) ;
31
32
33 CREATE TABLE Transactions
34 (
35     transaction_id     NUMBER NOT NULL,
36     patron_id          NUMBER NOT NULL,
37     book_id            VARCHAR2 (20 CHAR) NOT NULL,
38     transaction_date    DATE NOT NULL,
39     transaction_type     NUMBER NOT NULL,
40     books_book_id       VARCHAR2 (20 CHAR) NOT NULL,
41     patrons_patron_id   NUMBER NOT NULL
42 ) ;
43 ALTER TABLE Transactions ADD CONSTRAINT Transactions_PK PRIMARY KEY ( transaction_id ) ;
44
45
46 ALTER TABLE Transactions ADD CONSTRAINT Transactions_Books_FK FOREIGN KEY ( books_book_id ) REFERENCES Books ( book_id ) ;
47

```

4. Save the Design

Save the design by clicking File, then save. Specify the location and name for the XML file to contain the basic structural information (for example, library_design.xml).

A directory or folder structure will also be created automatically to hold the detailed information about the design. Continue creating and modifying design objects, if we wish. When we are finished, save the design again if we have made any changes, then exit SQL Developer Data Modeler by clicking File, then Exit.



Lab Tasks-

A prestigious university is revamping its student examination management system to modernize the process of conducting and managing examinations for a diverse student body. The new system is expected to handle all aspects of student assessments, including various types of examinations and evaluations. Here are the details,

1. **Exam Types-** The university conducts different types of exams, such as regular written exams, practical exams, oral exams, and project-based assessments. Each exam type has specific requirements and evaluation criteria.
2. **Course Offerings-** The university offers a wide range of courses across various disciplines. Each course has its own set of exams. Professors and instructors are responsible for defining the exams for their respective courses.
3. **Student Registrations-** Students register for courses and, consequently, the associated exams. The system should facilitate student registration and maintain a record of enrolled courses and their associated exams.
4. **Exam Scheduling-** The system needs to schedule exams, ensuring that there are no overlaps in exam timings for students who are registered for multiple courses. Professors and administrative staff are responsible for setting exam dates, times, and locations.
5. **Exam Administration-** On the day of the exam, professors, teaching assistants, and invigilators play a role in administering the exams, ensuring adherence to university rules and regulations. The system should provide a means to record attendance and monitor the exam process.
6. **Grading and Evaluation-** After the exams are completed, professors and teaching assistants evaluate student responses. Some exams may involve multiple-choice questions, while others require subjective assessments. The system should facilitate grading and provide tools for professors to input scores and comments.
7. **Results Publication-** The university needs to publish exam results securely. Students should be able to access their results, and professors need access to individual and aggregate student performance data for further analysis.
8. **Exception Handling-** The system must handle exceptional cases such as make-up exams for students who missed an exam due to illness or other valid reasons.
9. **Data Security-** Protecting the integrity and security of exam data is crucial. The system should have robust security measures to prevent unauthorized access or tampering with exam records.
10. **Reporting and Analytics-** The system should provide reporting and analytics features for the university administration to gain insights into student performance and course effectiveness.

To design the logical model, ER model, and generate DDL for this scenario, you will need to define entities, attributes, relationships, and business rules for each aspect of the scenario. The resulting database should efficiently store and manage information related to courses, exams, student registrations, exam scheduling, exam administration, grading, and exam results. Additionally, it should allow for scalability to accommodate the university's evolving examination needs.

Entities and Attributes	
Courses <ul style="list-style-type: none"> Course_ID (Primary Key) Course_Name Department Instructor_ID (Foreign Key) 	Registrations <ul style="list-style-type: none"> Registration_ID (Primary Key) Student_ID (Foreign Key) Course_ID (Foreign Key) Registration_Date
Exams <ul style="list-style-type: none"> Exam_ID (Primary Key) Course_ID (Foreign Key) Exam_Type Exam_Date Start_Time End_Time Location 	Exam_Enrollments <ul style="list-style-type: none"> Enrollment_ID (Primary Key) Student_ID (Foreign Key) Exam_ID (Foreign Key) Attendance
Students <ul style="list-style-type: none"> Student_ID (Primary Key) First Name Last Name Email Registration_Date 	Make-Up Exams <ul style="list-style-type: none"> Make_Up_ID (Primary Key) Student_ID (Foreign Key) Exam_ID (Foreign Key) Make_Up_Date Reason

Task 2:

A university campus is looking to modernize its hostel mess management system to efficiently cater to the dining needs of its students and staff. The existing system is manual and poses challenges in managing meal plans, dietary preferences, and billing. The university is keen on implementing a robust database system to streamline hostel mess operations. Here are the details of the scenario:

1. Hostel Residents

The university has multiple hostels, each housing a different set of students and staff. Hostel residents are expected to register for a meal plan. The system should maintain a database of residents with their details, including hostel ID, room number, and contact information.

2. Meal Plans

The system offers various meal plans to residents, allowing them to choose from options like full board, half board, or specific meal packages. Meal plans may also include dietary preferences or restrictions, such as vegetarian, vegan, or allergen-free meals.

3. Mess Operations

The hostel mess serves meals at specific timings and in designated dining areas. The system should manage mess operations, including meal scheduling, menu planning, and food inventory.

4. Billing and Payments

The system should generate bills for meal plans and allow residents to make payments. It should keep a record of billing cycles, payment history, and overdue payments.

5. Meal Booking

Residents should be able to book their meals in advance. They can specify the meal plan, dietary preferences, and any guest meals required. The system should provide a way to confirm or modify bookings.

6. Dietary Preferences and Allergies

The database should store dietary preferences, restrictions, and allergies of residents to ensure that meals are prepared according to individual needs.

7. Guest Meals

Occasionally, residents may have guests or visitors who need meal accommodations. The system should handle guest meal bookings and charges accordingly.

8. Feedback and Complaints

Residents should have a channel to provide feedback on the quality of meals and report any issues or complaints. The system should track and address these concerns.

9. Inventory Management

The hostel mess needs to manage food inventory efficiently. It should monitor the availability of ingredients, place orders for supplies, and ensure minimal food wastage.

10. Reports and Analytics

The system should offer reporting and analytics features for administrators to assess meal plan popularity, billing trends, and feedback analysis.

To design the logical model, ER model, and generate DDL for this scenario, you will need to define entities, attributes, relationships, and business rules for each aspect of the scenario. The resulting database should efficiently store and manage information related to residents, meal plans, billing, meal bookings, dietary preferences, and feedback. Additionally, it should allow for scalability to accommodate the university's changing dining needs.