**NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES**

**CL 203-Database Systems Lab**

**Lab Session 03**

**SQL Data Modeler**

* Oracle SQL Developer Data Modeler is a standalone, independent product, available for download from the Oracle Technology Network (OTN).
* SQL Developer Data Modeler runs on Windows, Linux and Mac OS X.
* To install SQL Developer Data Modeler simply unzip the downloaded file.
* With SQL Developer Data Modeler users can connect to any supported Oracle Databases. There is also support for IBM DB2 LUW V7 and V8, IBM DB2/390, Microsoft SQL Server 2000 and 2005 or a standard ODBC/JDBC driver for selective import of database objects and data browsing and migration.
* Models are stored as XML files and are easily shared or placed under source code control.

**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 supertype/sub-types entity hierarchies are resolved during forward engineering (transformation) of the logical model, or part of it, to a relational model.

**Physical Models**

SQL Developer Data Modeler supports most Oracle physical objects. It exposes many elements of an object’s structure and definition (for example, partitions and sub partitions) in the object browser.

**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).

If the instructions do not mention a particular dialog box, tab, or field, then do not specify anything for it.

We are using only a subset of the possible steps for the Top-Down Modeling approach. We will perform the following major steps:

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

**2.1 Develop the Logical Model**

The logical model for the database 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). However, before we create the entities,we will create some domains that will make the entity creation (and later DDL generation) more meaningful and specific.

**2.1.1 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.

We therefore decide to add appropriate domains, so that we can later use them to specify data types for attributes when we create the entities. (These added domains will also be available after we exit Data Modeler and restart it later.)

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.

| **Name** | **Logical Type** | **Other Information** |
| --- | --- | --- |
| Person Name | VARCHAR | Size: 25 |
| Address Line | VARCHAR | Size: 40 |
| City | VARCHAR | Size: 25 |
| State | VARCHAR | Size: 2 |
| Zip | VARCHAR | Size: 10 |
| Book Id | VARCHAR | Size: 20 |
| Numeric Id | NUMERIC | Precision: 7, Scale: 0 |
| Title | VARCHAR | Size: 50 |

1. When we have finished defining these domains, click **Save**. This creates a file named defaultdomains.xml in the domains directory (folder) under the location where we installed Data Modeler.
2. Optionally, copy the defaultdomains.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.
3. Click **Close** to close the dialog box.

**2.1.2 Creating the Books Entity**

The Books entity describes each book in the library. Create the Books entity as follows:

1. In the main area (right side) of the SQL Developer Data Modeler window, click the Logical tab.
2. Click the New Entity icon.
3. 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.
4. Click **General** on the left, and specify as follows:

**Name**: Books

1. Click **Attributes** on the left, and use the Add (+) icon to add the following attributes, one at a time. (For datatypes, select from the Domain types except for 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).) |

1. Click **OK** to finish creating the Books entity.

**2.1.3 Creating the Patrons Entity**

The Patrons entity describes each library patron (that is, each person who has a library card and is thus able to borrow books). Create the Patrons entity as follows:

1. In the main area (right side) of the SQL Developer Data Modeler window, click the Logical tab.
2. Click the New Entity icon.
3. 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. (Suggestion: draw the box to the right of the Books box.) The Entity Properties dialog box is displayed.
4. Click **General** on the left, and specify as follows:

**Name**: Patrons

1. Click **Attributes** on the left, and use the Add (+) icon to add the following attributes, one at a time. (For datatypes, select from the Domain types, except for location, which uses the structured type SDO\_GEOMETRY.)

| **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 | Structured type: SDO\_GEOMETRY | Oracle Spatial geometry object representing the patron's geocoded address. |

1. Click **OK** to finish creating the Patrons entity.

**2.1.4 Creating the Transactions Entity**

The Transactions entity describes each transaction that involves a patron and a book, such as someone checking out or returning a book. Each record a single transaction, regardless of how many books the patron brings to the library desk. For example, a patron returning two books and checking out three books causes five transactions to be recorded (two returns and three checkouts). Create the Transactions entity as follows:

1. In the main area (right side) of the SQL Developer Data Modeler window, click the Logical tab.
2. Click the New Entity icon.
3. 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. (Suggestion: Draw the box below and centered between the Books and Patrons boxes.) The [Entity Properties](https://docs.oracle.com/cd/E15276_01/doc.20/e13677/dialogs_data_modeling.htm#BABHACHD) dialog box is displayed.
4. Click **General** on the left, and specify as follows:

**Name**: Transactions

1. Click **Attributes** on the left, and use the Add (+) icon to add the following attributes, one at a time. (For datatypes, select from the Domain types, except for transaction\_date, which uses a Logical type.)

| **Attribute Name** | **Type** | **Other Information and Notes** |
| --- | --- | --- |
| transaction\_id | Domain: Numeric Id | Primary UID (unique identifier). (Unique transaction ID number) |
| patron\_id | Domain: Numeric Id | M (mandatory, that is, must not be null). Must match a patron\_id value in the Patrons entity. |
| book\_id | Domain: Book Id | M (mandatory, that is, must not be null). Must match a book\_id value in the Books entity. |
| 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.) |

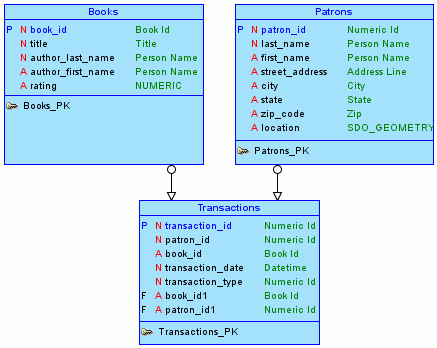
1. Click **OK** to finish creating the Transactions entity.

**2.1.5 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 checkout transactions reflecting several visits; each patron can also return checked out books at any time.

Create the relationships as follows. When we are done, the logical model pane in the main area should look like the following figure (using Bachman notation, which we can change to Barker by clicking View, then Logical Diagram Notation, then Barker Notation):



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.)

Suggestion: Turn off auto line routing for this exercise: right-click in the Logical pane, and ensure that Auto Route is not checked.

1. Click the New 1:N Relation icon.
2. Click first in the Books box, then in the Transactions box. A line with an arrowhead is drawn from Books to Transactions.
3. Click the New 1:N Relation icon.
4. Click first in the Patrons box, then in the Transactions box. A line with an arrowhead is drawn from Patrons to Transactions.
5. Optionally, double-click a line (or right-click a line and select Properties) and view the Relation Propertiesinformation.
6. Go to Section 2.2, "Develop the Relational Model".

**2.2 Develop the Relational Model**

The relational model for the library tutorial database 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 subset 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**.

**2.3 Generate DDL**

Generate Data Definition Language (DDL) statements that we can use to create database objects that reflect the models that we have designed. The DDL statements will implement the physical model (type of database, such as Oracle Database 11*g*) that we specify.

Develop the physical model as follows:

1. Optionally, view the physical model before we generate DDL statements:
   1. 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.
   2. Specify the type of database (for example, Oracle Database 11*g*), 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.
   3. 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.
2. Click **File**, then **Export**, then **DDL File**.
3. Select the database type (for example, Oracle Database 11g) and click **Generate**. The DDL Generation Optionsdialog box is displayed.
4. 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.)
5. Click **Save** to save the statements to a .sql script file (for example, create\_library\_objects.sql) on wer 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.

1. Click **Close** to close the DDL file editor.

**2.4 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**.

**Activity:**

1. Consider the following set of requirements for a UNIVERSITY database that is used to keep track of students’ transcripts.
2. The university keeps track of each student’s name, student number, Social Security number, current address and phone number, permanent address and phone number, birth date, sex, class (freshman, sophomore, ..., graduate), major department, minor department (if any), and degree program (B.A., B.S., ..., Ph.D.). Some user applications need to refer to the city, state, and ZIP Code of the student’s permanent address and to the student’s last name. Both Social Security number and student number have unique values for each student.
3. Each department is described by a name, department code, office number, office phone number, and college. Both name and code have unique values for each department.
4. Each course has a course name, description, course number, number of semester hours, level, and offering department. The value of the course number is unique for each course.
5. Each section has an instructor, semester, year, course, and section number .The section number distinguishes sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, ..., up to the number of sections taught during each semester.
6. A grade report has a student, section, letter grade, and numeric grade (0, 1, 2, 3, or 4).

Design an ER schema for this application, and draw an ER diagram for the schema using Sql data modeler. Perform forward engineering .

1. Consider a *mail order* database in which employees take orders for parts from customers.

The data requirements are summarized as follows:

1. The mail order company has employees identified by a unique employee number, their first and last names, and a zip code where they are located.
2. Customers of the company are uniquely identified by a customer number. In addition, their first and last names and a zip code where they are located are recorded.
3. The parts being sold by the company are identified by a unique part number. In addition, a part name, their price, and quantity in stock are recorded.
4. Orders placed by customers are taken by employees and are given a unique order number. Each order may contain certain quantities of one or more parts and their received date as well as a shipped date is recorded.

Design an Entity-Relationship diagram for the mail order database and enter the design using Sql data modeler.

1. Reverse engineer the employee department schema by importing demobld.sql