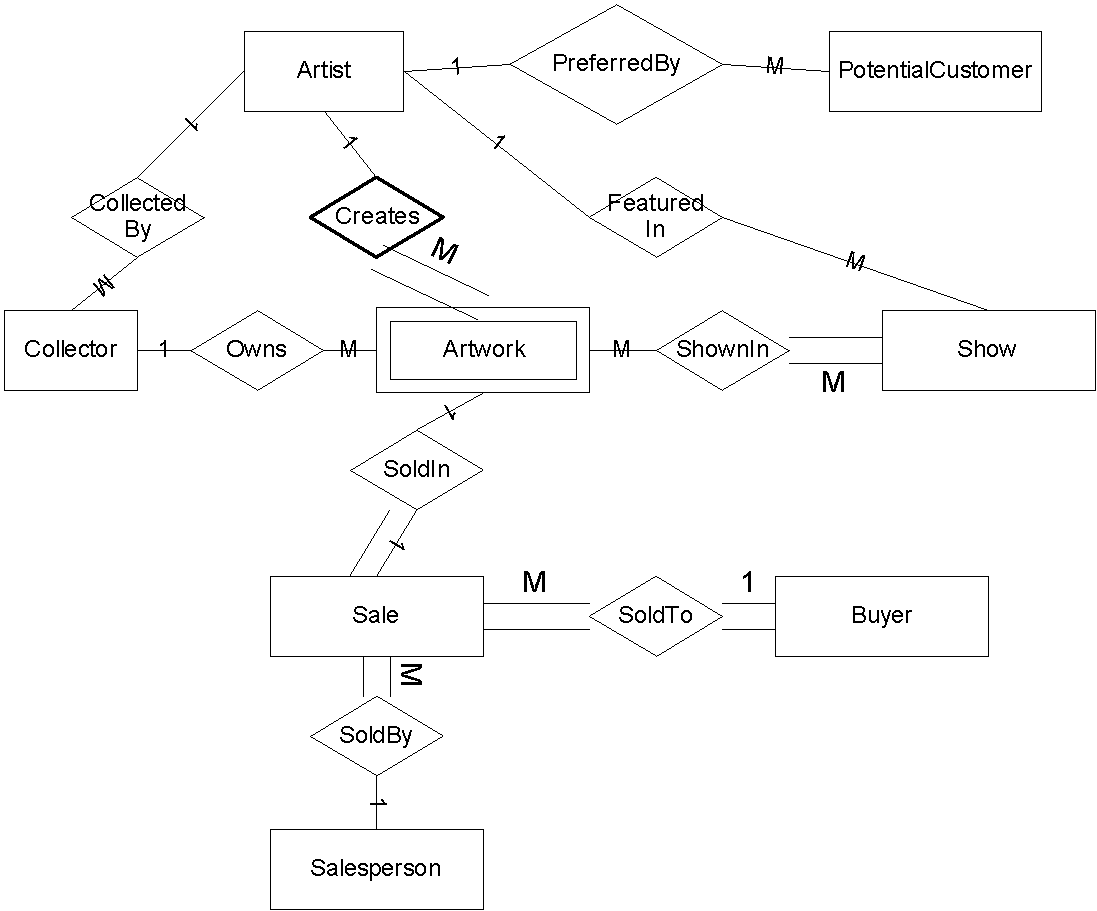
**Sample Project Chapter 7 - Creating a UML diagram for The Art Gallery and converting the UML Diagram to an object-oriented database schema**

* Step 7.1 - Create a UML Diagram for The Art Gallery

Although a UML diagram can be designed initially, we already have an E-R diagram that represents many important elements in our proposed design. Therefore, we will begin with the E-R diagram shown in Figure S.3.1 from the file *SampleUMLTheArtGallery* that appears in this directory. For convenience, we reproduce it here:



Note: The heavy line around the *Creates* diamond represents a double line.

**Artist** address(street, city, state, zip), interviewDate, interviewerName, name(first, last), phone(areaCode, telephoneNumber), salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType

**Artwork** askingPrice, dateListed, dateReturned, dateShown, status,workMedium, workSize, workStyle, workTitle,workType, workYearCompleted

**Buyer** name(firstName, lastName), address(street, city, state, zip), phone(areaCode, telephoneNumber), purchasesLastYear, purchasesYearToDate

**Collector** name(firstName, lastName), address(street, city, state, zip), interviewDate, interviewerName, phone(areaCode, telephonenumber), salesLastYear, salesYearToDate, collectionArtistFirstName, collectionArtistLastName, collectionMedium, collectionStyle, collectionType, SalesLastYear, SalesYearToDate, SocialSecurityNumber

**PotentialCustomer** address(street, city, state, zip), phone(areaCode, telephoneNumber), name(firstName, lastName),dateFilledIn, preferredArtist, preferredMedium, preferredStyle, preferredType.

**Show** showFeaturedArtist, showClosingDate, showTheme, showTitle, showOpeningDate

**Sale** amountRemittedToOwner, saleDate, InvoiceNumber, salePrice, saleSalesPersonCommission, saleTax, SaleTotal(*calculated field)*

**Salesperson** name(firstName, lastName), socialSecurityNumber, address(street, city, state, zip)

Figure S.3.1 (copy) - E-R Diagram for The Art Gallery

The resulting UML diagram is sketched in Figure S.7.1 below. The process by which we can go from the E-R diagram to the UML diagram is described here. A similar process was used in the sample project for Chapter 3 to create an EE-R diagram, but we repeat it here.

In the E-R diagram, we observe that Artist, Buyer, PotentialCustomer, Collector, and Salesperson all have similar attributes. Since they are all people, we will create a Person class with subclasses. Our subclasses could be Artist, Collector, Salesperson, PotentialCustomer, and Buyer.

In the E-R diagram, we notice that Buyer and PotentialCustomer have some common attributes and they have similar meanings. Both are customers. We see that we keep preferences for potential customers but not for buyers. It makes good business sense to add the preference information for Buyer, and this makes their attributes identical, except that we have additional information about sales totals for buyers. We conclude that a buyer is a customer who has actually made a purchase, i.e. a Buyer is someone who has a relationship with a Sale entity. Therefore, we can create a class called Customer having a subclass Buyer that has a relationship with Sale. The other subclasses of Person remain - Artist, Collector, Salesperson.

The Artwork entity is pictured in the original E-R diagram as having a relationship to Sale but in fact only those works that have been sold have this relationship. The status attribute determines which Artwork entities participate in the relationship. From the possible status values, we can identify three disjoint subclasses of Artwork, namely, SoldWork, WorkForSale, and ReturnedWork, as shown in Figure S.7.1 below. Of these, only SoldWork relates to Sale. We add an attribute, location, for WorkForSale. It will give the current location of the artwork, and we move the dateReturned attribute to ReturnedWork.

The Artist and Collector classes both relate to Artwork. Examining these classes more closely, we see that an Artwork entity is sometimes owned by the artist and sometimes by a collector. In the EE-R model, we created a union called Owner to refer to both types of owners, but in this example we will choose to assume that the Collector relationship exists only for those works that are not owned by the artist. If a work does not participate in this relationship, it is owned by the artist. However, Artist always participates in the Creates relationship with Artwork, whether the artist is the owner or not.

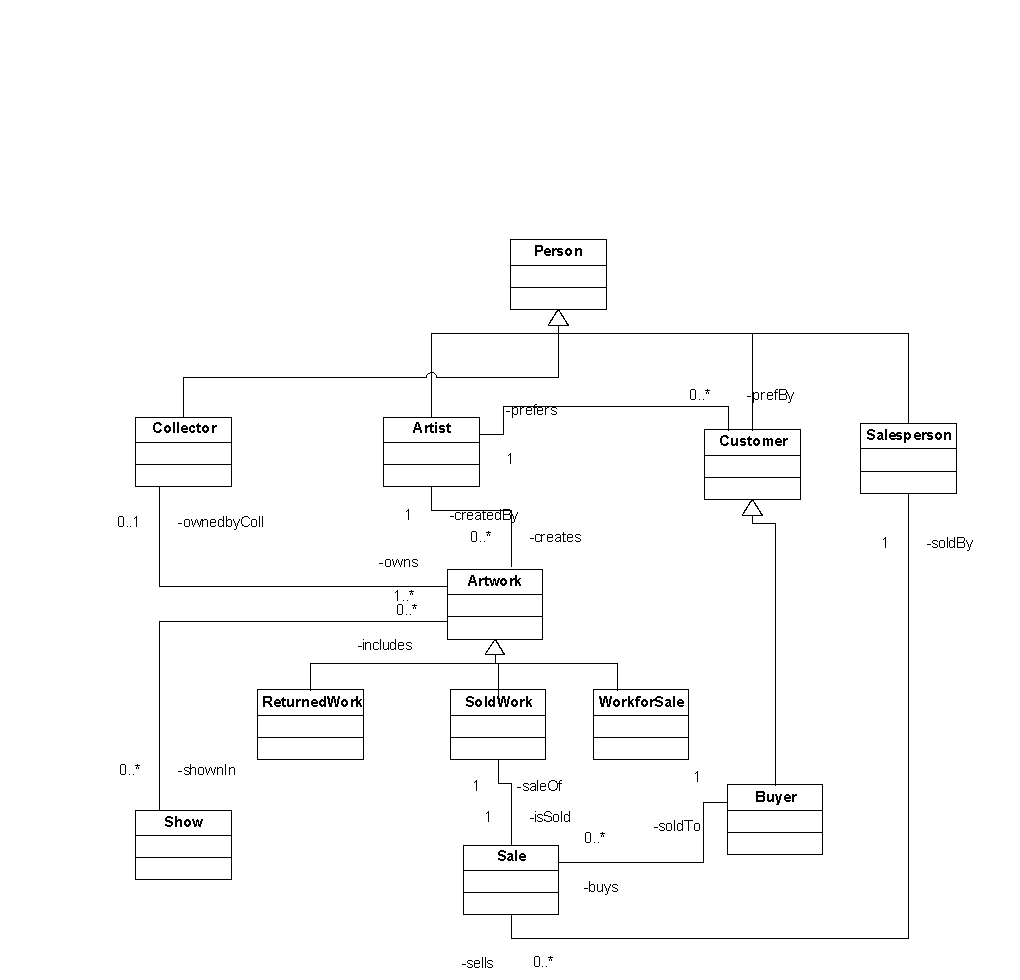
The remaining entities, Show and Sale, become classes in UML. 

Figure S.7.1 - Simplified UML Diagram for The Art Gallery

Figure S.7.1 shows a sketch of a UML diagram for the sample project. To keep the diagram simple, class attributes and methods are not shown. We have created a Person class, with Collector, Artist, Customer, and Salesperson as subclasses. Customer has a subclass, Buyer. The Artwork class has subclasses ReturnedWork, WorkForSale, SoldWork. We have added a relationship between Artwork and Show, since a show might feature a particular artist. We have chosen to make all relationships bidirectional, by providing inverses for each. We have added bidirectional relationships among the classes, but we need to verify them and to decide on the multiplicity indicators, which correspond to cardinality and participation constraints in the ER diagram. We use the *min..max* notation. Figure S.7.2 shows a UML diagram created using Vizio, with the classes, relationships, and multiplicity indicators included. The corresponding Vizio file is stored in this directory under the name *CompleteUMLTheArtGallery*. To produce that diagram, we examined each relationship as described below.

We begin with the relationship between Artist and Artwork. We decided earlier that we may keep data on interviewed artists who do not yet have artworks accepted by the gallery, so for the Artist to Artwork relationship, which we now call creates, the min is 0. The max is n, since an artist can have created many accepted artworks, so we use 0..\* for creates. For the inverse, called createdBy, the Artwork always relates to exactly one artist, so we use 1..1, written simply as 1, for createdBy.

For SoldWork to Sale, called isSold, since each sold work of art is sold exactly once, we use 1. For Sale to SoldWork, called saleof, both min and max are also 1, so we write 1 on this line.

For Sale to Buyer, called soldTo, each sale has exactly one buyer, so we write 1. For Buyer to Sale, called buys, since a buyer is, by definition, someone who has been involved in a Sale, the min is 1. Since a buyer can buy many artworks, the max is n, so we use 1..\*.

For Sale to Salesperson, called soldBy, each sale is made by exactly one salesperson, so we use 1. For Salesperson to Sale, called sells, since a salesperson may not have made any sales yet, the min is 0. A salesperson can make many sales, so the max is n, giving us 0..\*.

For the Collector to Artwork relationship, called owns, we decided to include collector information on people who have been interviewed but have not yet had an artwork accepted, so we will assume we can store collector information without an artwork, making the min 0. A Collector can have many works of art, so the max is n, giving us 0..\*. For Artwork to Collector, called ownedbyColl, a work of art has either 0 or no collector as an owner, making it 0..1. A artwork that has no collector owner is assumed to be still owned by the artist.

For Artwork to Show, called shownIn, the min is 0, since not all works are shown. The max is n, since an artwork can appear in many shows, giving us 0..\*. For Show to Artwork, called includes, we will make the min 1, assuming we could have a showing of a single spectacular piece of art. The max will be n, since a show would normally have many works of art, giving us 1..\*.

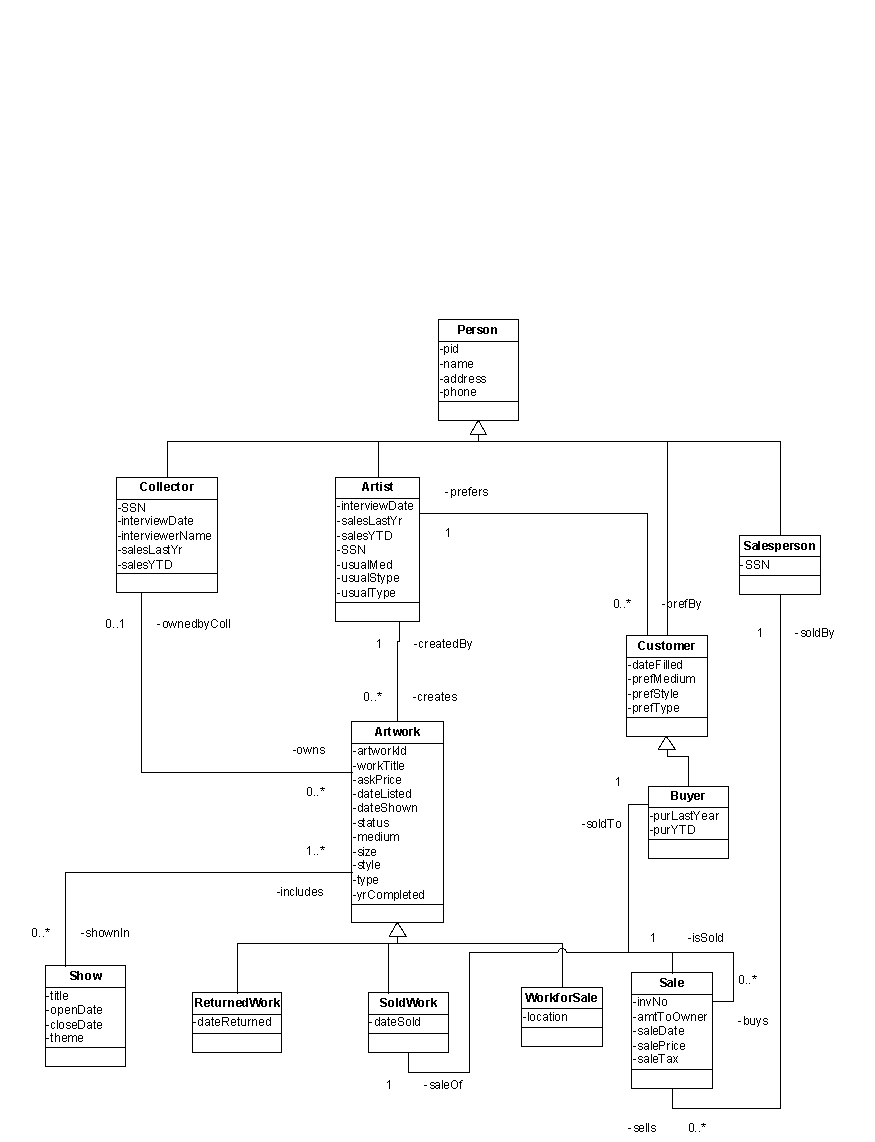
We assume each class has get and set methods for all its attributes, and possibly other methods, but to save space we do not show them in the diagram, nor do we show the data types for attributes. 

Figure S.7.2 - UML Diagram for The Art Gallery

* Step 7.2 - Convert the UML Diagram to an Object-Oriented Database Schema

Figure S.7.3 gives the ODL for a database that corresponds to the UML diagram. A few methods are included for the Person class. We added a method to the Artwork class, called OwnerIsArtist, that will return a Boolean value. It tests whether the reference to Collector is null and if so, returns true, indicating that the artist owns the work.

class Person(

key pId)

{ attribute int pId;

attribute Struct NameType(string first, string last) name;

attribute Struct AddressType(string street, sting city, string state, string zip) address;

attribute Struct PhoneType(string areaCode, string telephoneNumber) phone;

int getPId( );

void setPId(int newId);

string getName( );

void setName(string newName);

string getAddress( );

void setAddress(AddressType newAddress);

string getPhone( );

void setPhone(PhoneType newPhone);

};

class Artist extends Person

(extent Artists)

{

attribute Struct DateType(int day, int month, int year) interviewDate;

attribute NameType interviewerName;

attribute real(10,2) salesLastYear;

attribute real(10,2) salesYearToDate;

attribute string socialSecurityNumber;

attribute string usualMedium;

attribute string usualStyle;

attribute string usualType;

DateType getinterviewDate();

void setinterviewDate(DateType newartistinterviewDate);

NameType getinterviewerName();

void setinterviewerName(NameType newartistinterviewerName);

real getsalesLastYear();

void setsalesLastYear(real artistlastyearsalesAmount);

real get salesYearToDate();

void setsalesYearToDate(real artistsalesYTD);

void updatesalesYearToDate(real artistNewSalesAmount);

string getsocialSecurityNumber();

void setsocialSecurityNumber(string artistnewSSN);

string getusualMedium();

void setusualMedium(string artistusualMedium);

string getusualStyle();

void setusualStyle(string artistusualStyle);

string getusualType();

void setusualType(string artistusualType);

relationship Set<Artwork> creates Inverse Artwork::createdBy;

relationship Set<Customer> preferredBy Inverse Customer::prefers;

relationship Set<Show> featuredIn Inverse Show::features;

class Collector extends Person

(extent Collectors)

{

attribute string socialSecurityNumber;

attribute Artist::DateType interviewDate;

attribute NameType interviewerName;

attribute real(10,2) salesLastYear;

attribute real(10,2) salesYearToDate;

string getsocialSecurityNumber();

void setsocialSecurityNumber(string collectornewSSN);

Artist::DateType getinterviewDate();

void setinterviewDate(Artist::DateType newcollectorinterviewDate);

NameType getinterviewerName();

void setinterviewerName(NameType newcollectorinterviewerName);

real getsalesLastYear();

void setsalesLastYear(real collectorsalesLastYearAmount);

real getsalesYearToDate();

void setsalesYearToDate(real collectorsalesYTDAmount);

void updatesalesYearToDate(real collectornewSalesAmount);

relationship Set<Artwork> owns Inverse Artwork::ownedBy;

};

class Artwork

(extent Artworks)

{

attribute string artworkId;

attribute string workTitle;

attribute real(6,2) askingPrice;

attribute Artist::DateType dateListed;

attribute Artist::DateType dateShown;

attribute string status;

attribute string workMedium;

attribute string workSize;

attribute string workStyle;

attribute string workType;

attribute int workYearCompleted;

string getartworkId();

void setartworkId(string newartworkId);

string getartworkTitle();

void setartworkTitle(string newartworkTitle);

real getaskingPrice();

void setaskingPrice(real newaskingPrice);

Artist::DateType getdateListed();

void setdateListed(Artist::DateType newdateListed);

Artist::DateType getdateShown();

void setdateShown(Artist::DateType newdateShown);

string getstatus();

void setstatus(string newstatus);

string getworkMedium();

void setstworkMedium(string newmedium);

string getworkSize();

void setstworkSize(string newsize);

string getworkStyle();

void setstworkStyle(string newStyle);

string getworkType();

void setstworkType(string newType);

int getworkYearCompleted();

void setworkYearCompleted(int newworkYearCompleted);

boolean artistIsOwner( )//method to determine if the work is owned by the artist;

relationship Artist createdBy Inverse Artist::creates;

relationship Set<Show> inShow Inverse Show::shownIn;

relationship Collector ownedBy Inverse Collector::owns;

};

class SoldWork extends Artwork

{

attribute Artist::DateType dateSold;

Artist::DateType getdateSold();

void setdateSold(DateType newdateSold);

relationship Sale isSold Inverse Sale::saleOf;

relationship Buyer isBought Inverse Buyer::buys;

relationship Salesperson soldBy Inverse Salesperson::sells

};

class ReturnedWork extends Artwork

{

attribute Artist::DateType dateReturned;

Artist::DateType getdateReturned();

void setdateReturned(Artist::DateType newdateReturned);

};

class WorkForSale extends Artwork

{

attribute string location;

string getlocation();

void setlocation(string newlocation);

};

class Customer extends Person

(extent Customers)

{

attribute Artist::DateType dateFilledIn;

attribute string preferredMedium;

attribute string preferredStyle;

attribute string preferredType;

Artist::DateType getdateFilledIn();

void setdateFilledIn(Artist::DateType newdateFilledIn);

string getpreferredMedium();

void setpreferredMedium(string newpreferredMedium);

string getpreferredStyle();

void setpreferredStyle(string newpreferredStyle);

string getpreferredType();

void setpreferredType(string newpreferredType);

relationship Artist prefers Inverse Artist::preferredBy;

};

class Buyer extends Customer

{

attribute real(8,2) purchasesLastYear

attribute real(8,2) purchasesYearToDate;

real getpurchasesLastYear();

void setpurchasesLastYear(real purchasesLastYearAmount);

real getpurchasesYearToDate();

void setpurchasesYearToDate(real purchasesYearToDateAmount);

void updatepurchasesYearToDate(real amountNewPurchases);

relationship Set<SoldWork> buys Inverse SoldWork::isBought;

};

class Show

(extent Shows)

{

attribute string showTitle;

attribute Artist::DateType showOpeningDate;

attribute Artist::DateType showClosingDate;

attribute string showTheme;

string getshowTitle();

void setshowTitle(string newTitle);

Artist::DateType getshowclosingDate();

void setshowclosingDate(string newTitle);

string getshowTheme();

void setshowTheme(string newTitle);

Artist::DateType getshowopeningDate();

void setshowopeningDate(string newTitle);

relationship Artist features Inverse Artist::featuredIn;

relationship Set<Artwork> shownIn Inverse Artwork::inShow;

};

class Sale

(extent Sales)

{

attribute string invoiceNumber;

attribute real(8,2) amountRemittedToOwner;

attribute Artist::DateType saleDate;

attribute real(8,2) salePrice;

attribute real(6,2) saleTax;

string getinvoiceNumber();

void setinvoiceNumber(string newinvoiceNumber);

real get amountremittedToOwner();

void setamountremittedToOwner(real newamountRemitted);

Artist::DateType getsaleDate();

void setsaleDate(Artist::DateType newsaleDate);

real getsalePrice();

void setsalePrice(real newsalePrice);

real getsaleTax();

void setsaleTax(real newtaxAmount);

relationship SoldWork saleOf Inverse SoldWork::isSold;

relationship Salesperson soldBy Inverse Salesperson::sells;

};

class Salesperson extends Person

(extent Salespersons)

{

attribute string socialSecurityNumber;

string getsocialSecurityNumber();

void setsocialSecurityNumber(string newSSN);

relationship Set<Sale> sells Inverse Sale::soldBy;

};

Figure S.7.3 - ODL for The Art Gallery Schema

* Step 7.3 - Using Objectivity, Cache, or another object-oriented DBMS, create an object-oriented database for your schema.

The Person, Artist, and Artwork classes and the necessary definitions appear in the textfile called *CacheCodeArtGallery* in this directory and is shown here:

Class ArtGallery.Address Extends %SerialObject

{

/// street

Property street As %String(MAXLEN = 60);

/// city

Property city As %String;

/// state

Property state As %String;

/// zip

Property zip As %String(MAXLEN = 10);

}

Class ArtGallery.telephone Extends %SerialObject

{

/// The area code

Property areaCode As %String(MAXLEN = 3);

/// The 7 digit phone number

Property phoneNumber As %String(MAXLEN = 8);

}

Class ArtGallery.Person Extends %Persistent

{

/// The Person's ID

Property pID As %Integer;

/// The person's name

Property personName As %Name;

/// The person's address

Property personAddress As Address;

/// The person's phone

Property personPhone As telephone;

}

Class ArtGallery.Artist Extends (ArtGallery.Person, %Persistent, %XML.Adaptor, %ZEN.DataModel.Adaptor)

{

/// Date artist was interviewed

Property interviewDate As %Date;

/// Employee who interviewed the artist

Property interviewerName As %Name(MAXLEN = 40);

/// Total of the artist's sales last year

Property salesLastYear As %Currency;

/// Total of the artist's sales to date this year

Property salesThisYear As %Currency;

/// The artist's social security number

Property socialSecurityNumber As %String(MAXLEN = 9);

/// The artistic medium the artist usually uses.

Property usualMedium As %String(MAXLEN = 25);

/// The style the artist usually uses.

Property usualStyle As %String(MAXLEN = 25);

/// The artistic medium the artist usually uses.

Property usualType As %String(MAXLEN = 25);

Relationship Creates As ArtGallery.Artwork [ Cardinality = many, Inverse = CreatedBy ];

}

Class ArtGallery.Artwork Extends %Persistent

{

/// Unique ID of the work of art

Property artworkID As %String(MAXLEN = 8);

/// The title of the work of art

Property workTitle As %String(MAXLEN = 80);

/// The asking price of the work of art

Property askingPrice As %Currency;

/// The date the work of art was first listed for sale by the gallery

Property dateListed As %Date;

/// The last date the work of art was included in a show by the gallery

Property dateShown As %Date;

/// The medium used for the work of art

Property workMedium As %String;

/// The dimensions of the work of art

Property workSize As %String;

/// The style used for the work of art

Property workStyle As %String;

/// The type of the work of art

Property workType As %String;

/// The year the work of art was completed

Property yearCompleted As %String;

Relationship CreatedBy As ArtGallery.Artist [ Cardinality = one, Inverse = Creates ];

Index CreatedByIndex On CreatedBy;

}

* Step 7.4 - Enter at least five records for each class in your database.

We used Cache’s Zen framework to develop a simple server page for each table, and entered data in each one by hand. A sample page appears here:

**ArtGallery.Artwork**



|  |  |
| --- | --- |
| **artworkID:** |  |
| **workTitle:** |  |
| **askingPrice:** |  |
| **dateListed:** |  |
| **dateShown:** |  |
| **workMedium:** |  |
| **workSize:** |  |
| **workStyle:** |  |
| **workType:** |  |
| **yearCompleted:** |  |
|  |  |



After the records are entered, the relationships have to be established using the Cache terminal.

* Step 7.5 - Write out some queries for your database and print out the questions, the queries, and the results. We used the Cache’s *System Management Portal* to write SQL queries for the database. Some results are shown below.

Find title, asking price, and year completed of all paintings.

SQL Query:

SELECT workTitle,workType,askingPrice,yearCompleted

FROM artgallery.artwork

WHERE worktype ='painting'

Result:

|  |  |  |
| --- | --- | --- |
| The results of executing the SQL query are shown below: | Last update: |  |

|  |  |  |
| --- | --- | --- |
| SQLCODE: [**100**](about:blank)  Row count: **3** |  | Performance: **0.000** seconds  **27** global references |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **workTitle** | **workType** | **askingPrice** | **yearCompleted** |
| 1 | Flight | painting | 15000 | 2009 |
| 2 | Mediterrean Coast | painting | 4000 | 2000 |
| 3 | Bermuda Sunset | painting | 8000 | 2003 |
| *Complete* | | | | |