**Sample Project Chapter 6 - Creating a Normalized Relational Model for The Art Gallery**

In the sample project section at the end of chapter 3, we created an E-R diagram for The Art Gallery. At the end of chapter 4, we saw how to map that diagram to tables. We created an initial relational database for the resulting schema at the end of Chapter 5. In the present section, we want to normalize the schema and create a better relational database. Although we could begin the normalization process by making a single universal relation that lists all attributes and then trying to isolate functional dependencies among the attributes, we can make our job easier by using the entities and relationships we identified in Chapter 3 and the tables to which they mapped at the end of Chapter 4. In practice, tables that result from such mappings are usually almost normalized already. Therefore, we use our mapped tables as a starting point.

* Step 6.1 Begin with the list of the tables that the entities and relationships from the E-R diagram mapped to naturally, from the sample project section at the end of chapter 4. For each table on the list, identify functional dependencies and normalize the relation to BCNF. Then decide whether the resulting tables should be implemented in that form. If not, explain why.

The following tables resulted from the mapping:

**Artist**(firstName, lastName, street, city, state, zip, interviewDate, interviewerName, areaCode, telephoneNumber, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

**PotentialCustomer**(firstName, lastName, areaCode, telephoneNumber, street, city, state, zip, dateFilledIn, *preferredArtistLastName, preferredArtistFirstName*, preferredMedium, preferredStyle, preferredType)

**Artwork**(*artistLastName, artistFirstName*, workTitle, askingPrice, dateListed, dateReturned, dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted, *collectorSocialSecurityNumber*)

**ShownIn**(*artistLastName, artistFirstName, workTitle, showTitle*)

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

**Show**(showTitle, *showFeaturedArtistLastName, showFeaturedArtistFirstName*, showClosingDate, showTheme, showOpeningDate)

**Sale**(InvoiceNumber, *artistLastName, artistFirstName*, *workTitle,* amountRemittedToOwner, saleDate, salePrice, saleSalesPersonCommission, saleTax, SaleTotal, *buyerLastName, buyerFirstName, buyerAreaCode, buyerTelephoneNumber, salespersonSocialSecurityNumber*)

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

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

For the Artist table, let us identify FDs.

firstName + lastName → all attributes

It would appear that

socialSecurityNumber → all attributes

Recall that BCNF permits determinants that are candidate keys to remain in the table, so we do not have a problem with leaving socialSecurityNumber in the table. However, we assumed that we would not always have the social security number of artists whose works were owned by collectors, so the value of this attribute may be null for some artists. However, when it appears, it is unique in the table.

zip → city, state

We might want to consider

areaCode →? city, state

We conclude that with mobile telephones, the area code is not necessarily associated with the city and state of the artist’s residence or studio, so we do not have this as a functional dependency.

We also consider whether the complete telephone number determines the address

areaCode + telephoneNumber →? street, city, state, zip

Is it possible for two artist records with the same telephone number to have two different addresses? If the telephone is a landline, the addresses should be the same (the address of the location of the telephone). If it is a mobile telephone, the two artists would have to be sharing it for the same number to appear in two different records. We will keep this as a functional dependency. We might then want to ask

areaCode + telephoneNumber →? all attributes

We decide this is not the case, since we consider the possibility that two artists may share the same home or studio and the same telephone number there, but still have different names, styles, and so on.

The table is in 1NF and 2NF, but not 3NF or BCNF because of the dependencies we have listed. We therefore decompose the Artist table as follows

**Artist1**(firstName, lastName, interviewDate, interviewerName, *areaCode, telephoneNumber*, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

Phones(areaCode, telephoneNumber, street, *zip*)

Zips(zip, city, state)

However, this design would require that we use the telephone number in order to get the street and zip code of an artist, and that we do two joins whenever we want to get an artist’s complete address. For the sake of efficiency, we will compromise and put the street and zip back in the Artist1 table. We choose to leave the Zips table as it is, noting that complete zip code tables are available for purchase in electronic form. Now our form for the Artist tables is

**Artist2**(firstName, lastName, interviewDate, interviewerName, areaCode, telephoneNumber, street, *zip*, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

Zips(zip, city, state)

In practice, it is better to use a numeric field for a key value than a character string, which is subject to differences in spelling, capitalization and punctuation in entering data. Data entry errors or variations in entering string data can cause errors when we try to compare values; in particular, when the fields are used as foreign keys. Therefore, we will create a unique numeric identifier for each artist, which we will call artistId, and make that the primary key of the first table. All the attributes of the first table will be functionally dependent on it. This type of key is sometimes called a **surrogate key**, and most database management systems have a mechanism to generate values and to keep track of values for surrogate keys. Oracle uses identity columns or sequences for this purpose. Microsoft Access uses an **autonumber** data type for the same purpose. We therefore have as our final artist tables

(1) **Artist3**(artistId, firstName, lastName, interviewDate, interviewerName, *areaCode, telephoneNumber*, street, *zip*, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

(2) **Zips**(zip, city, state)

For PotentialCustomer we have many of the same FDs that we saw for Artist

firstName + lastName + areaCode + telephoneNumber→ all attributes

areaCode + telephoneNumber → street, city, state, zip

zip → city, state

Using the same reasoning as we did for Artist, we will add a unique identifier, potentialCustomerId, to use as the primary key. We might question whether there is a functional dependency between the preferred artist and other preferences. Although there may be some connection, it is not a true functional dependency, since, for example, two people who admire the same artist might prefer works that are in different mediums or styles, perhaps even produced by the same artist.

For strict BCNF we would break up PotentialCustomer as follows:

Customer1(potentialCustomerId, firstname, lastName, *areaCode, telephoneNumber*, dateFilledIn, *preferredArtistLastName, preferredArtistFirstName*, preferredMedium, preferredStyle, preferredType)

Phones(areaCode, telephoneNumber, street, *zip*)

**Zips**(zip, city, state)

Using the same logic that we used for artist information, we choose instead to create a table that keeps the street and zip code with the other customer data, and to use the Zips table that already exists to determine city and state. We also want to use the preferred artistId instead of first and last name. Therefore, we add to the design the table

(3) **PotentialCustomer2**(potentialCustomerId, firstname, lastName, areaCode, telephoneNumber, street, *zip,* dateFilledIn, *preferredArtistId,* preferredMedium, preferredStyle, preferredType)

For Artwork, we have the following FDs

artistLastName + artistFirstName + workTitle → all attributes

We were using the artist name as a foreign key here. Since we changed the primary key of Artist to artistId, we will change the foreign key as well, replacing the name by the Id in the Artwork table.

Although there is some logical connection between the dates, there is no functional dependency between them. We will also create a unique identifier for each work, artworkId

(4) **Artwork**(artworkId, *artistId*, workTitle, askingPrice, dateListed, dateReturned, dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted, *collectorSocialSecurityNumber*)

For the ShownIn table there are no non-trivial functional dependencies, so we could keep the table in its current form. However, we wish to use artworkId to identify the artwork.

(5) **ShownIn**(*artworkId, showTitle*)

In the Collector table, we have the FDs

socialSecurityNumber → all attributes

as well as the FDs we saw previously involving telephone numbers and zip codes. We also want to use artistId in place of the artist’s first and last name. We therefore chose to create the table shown below, and to make use of the existing Zips table.

(6) **Collector1**(socialSecurityNumber, firstName, lastName, street, *zip*, interviewDate, interviewerName, areaCode, telephonenumber, salesLastYear, salesYearToDate, *collectionArtistId*, collectionMedium, collectionStyle, collectionType, SalesLastYear, SalesYearToDate)

For the Show table, we have the FD

showTitle → all attributes

However, we wish to substitute the artistId for the name, as we did for earlier tables. There may be some connection between the featured artist and the show title, but they are not necessarily functionally dependent. For example, two different shows may both feature the same artist, but their titles will be different. The same is true of the theme and the title. There is also some connection, between the opening date and closing date of the show. If we assumed that only one show can open on a given date, then there would be only one closing date associated with that opening date, and we would have a transitive dependency that we would need to remove. Let us add to the assumptions that more than one show can open at the same time. This is a reasonable assumption if the gallery is large enough. In that case there may be shows that have the same opening date but different closing dates. Given these assumptions, the Show table is already normalized.

(7) **Show**(showTitle, *showFeaturedArtistId,* showClosingDate, showTheme, showOpeningDate)

For Buyer, we have the FD

firstName + lastName + areaCode +telephoneNumber → all attributes

as well as the FDs involving telephone numbers and zips codes, as we saw earlier for Artist and for Collector. As we did for Artist, we will create a numeric primary key, buyerId. Using the same pattern as for those tables, we design a new Buyer table and make use of the Zips table designed previously.

(8) **Buyer**(buyerId, firstName, lastName, street, zip, areaCode, telephoneNumber, purchasesLastYear, purchasesYearToDate)

For the Sale table, we have the FD

invoiceNumber → all attributes

Since each artwork is sold at most once, we also have

artworkId → all attributes

Since it is permissible to keep a candidate key in the relation, this does not present a problem.

Again, we will substitute the artworkId for the artist name and title, and the buyerId for the buyer name and telephone number as foreign keys.

If the commission is a constant percentage of the sale price, then we have

salePrice → saleSalesPersonCommission

We do not assume that sale price determines tax, since some buyers, such as non-profit organizations, may be tax-exempt. However, the sale total is just the sum of sale price and tax, so we have

salePrice + tax → saleTotal

Actually, any two of these determine the other. We might think we have the FD between salePrice and amountRemittedToOwner However, it is possible that there may be a delay between the time a work is sold and the time an owner is paid, so we could have two sales with the same sale price and different amounts remitted to the owner, because one has not yet been paid.

Removing the FDs identified here, we form the new table

(9) **Sale1**(InvoiceNumber, *artworkId,* amountRemittedToOwner, saleDate, salePrice, saleTax, *buyerId, salespersonSocialSecurityNumber*)

We could also have the table

Commissions(salePrice, saleSalesPersonCommission)

but we do would not need to store this, since the commission is easily calculated. Similarly, because of the arithmetic relationship among the attributes, we do not need to store the table

Totals(salePrice, saleTax, SaleTotal)

For Salesperson we have

socialSecurityNumber → all attributes

zip → city, state

Removing the transitive dependency and making use of the existing Zips table, we have

(10) **Salesperson**(socialSecurityNumber, firstName, lastName, street, zip)

The tables in boldface, numbered 1-10, will be used as the final set of tables for the relational design for this database.

* Step 6.2 - Update the data dictionary and list of assumptions as needed.

We need to add to the data dictionary the new identifiers we created, as follows:

***artistId*** *A unique numeric identifier created for each artist.*

***artworkId*** *A unique numeric identifier created for each work of art.*

***buyerId*** *A unique numeric identifier created for each buyer.*

***potentialCustomerId*** *A unique numeric identifier created for each potential customer*.

We make a note of the calculated items that are not to be stored

**saleTotal** The total dollar amount of a sale, including price and tax, for an artwork*; calculated from salePrice and saleTax. Calculated item*

**saleSalesPersonCommission** The dollar amount of commission for a salesperson for the sale of a work of art. *Calculated item*

There are no changes to the list of assumptions.

* Step 6.3 - For each table, write the table name and write out the names, data types, and sizes of all the data items, Identify any constraints, using the conventions of the DBMS you will use for implementation.

. For an Oracle database, the tables will have the structures shown below.

TABLE Zips

Item Datatype Size Constraints Comments

Zip CHAR 10 PRIMARY KEY

city VARCHAR2 15 NOT NULL

state CHAR 2 NOT NULL

TABLE Artist

artistId NUMBER 6 PRIMARY KEY

firstName VARCHAR2 15 NOT NULL; (firstName, lastName) UNIQUE

lastName VARCHAR2 20 NOT NULL; (firstName, lastName) UNIQUE

interviewDate DATE

interviewerName VARCHAR2 35

areaCode CHAR 3

telephoneNumber CHAR 7

street VARCHAR2 50

zip CHAR 5 FOREIGN KEY REF Zips

salesLastYear NUMBER 8,2

salesYearToDate NUMBER 8,2

socialSecurityNumber CHAR 9 UNIQUE

usualMedium VARCHAR 15

usualStyle VARCHAR 15

usualType VARCHAR 20

TABLE Collector

socialSecurityNumber CHAR 9 PRIMARY KEY

firstName VARCHAR2 15 NOT NULL

lastName VARCHAR2 20 NOT NULL

interviewDate DATE

interviewerName VARCHAR2 35

areaCode CHAR 3

telephoneNumber CHAR 7

street VARCHAR2 50

zip CHAR 5 FOREIGN KEY Ref Zips

salesLastYear NUMBER 8,2

salesYearToDate NUMBER 8,2

collectionArtistId NUMBER 6 FOREIGN KEY REF Artist

collectionMedium VARCHAR 15

collectionStyle VARCHAR 15

collectionType VARCHAR 20

TABLE PotentialCustomer

potentialCustomerId NUMBER 6 PRIMARY KEY

firstname VARCHAR2 15 NOT NULL

lastName VARCHAR2 20 NOT NULL

areaCode CHAR 3

telephoneNumber CHAR 7

street VARCHAR2 50

zip CHAR 5 FOREIGN KEY REF Zips

dateFilledIn DATE

preferredArtistId NUMBER 6 FOREIGN KEY REF Artist

preferredMedium VARCHAR2 15

preferredStyle VARCHAR2 15

preferredType VARCHAR2 20

TABLE Artwork

artworkId NUMBER 6 PRIMARY KEY

artistId NUMBER 6 FOREIGN KEY REF Artist; NOT NULL; (artistId, workTitle) UNIQUE

workTitle VARCHAR2 50 NOT NULL; ( artistId, workTitle) UNIQUE

askingPrice NUMBER 8,2

dateListed DATE

dateReturned DATE

dateShown DATE

status VARCHAR2 15

workMedium VARCHAR2 15

workSize VARCHAR2 15

workStyle VARCHAR2 15

workType VARCHAR2 20

workYearCompleted CHAR 4

collectorSocialSecurityNumber CHAR 9 FOREIGN KEY REF Collector

TABLE Show

showTitle VARCHAR2 50 PRIMARY KEY

showFeaturedArtistId NUMBER 6 FOREIGN KEY REF Arist

showClosingDate DATE

showTheme VARCHAR2 50

showOpeningDate DATE

TABLE ShownIn

artworkId NUMBER 6 PRIMARY KEY(artworkId, showTitle); FOREIGN KEY REFArtwork

showTitle VARCHAR2 50 PRIMARY KEY(artworkId, showTitle); FOREIGN KEY REF Show

TABLE Buyer

buyerId NUMBER6 PRIMARY KEY

firstName VARCHAR2 15 NOT NULL

lastName VARCHAR2 20 NOT NULL

street VARCHAR2 50

zip CHAR 5 FOREIGN KEY REF Zips

areaCode CHAR 3

telephoneNumber CHAR 7

purchasesLastYear NUMBER 8,2

purchasesYearToDate NUMBER 8,2

TABLE Salesperson

socialSecurityNumber CHAR 9 PRIMARY KEY

firstName VARCHAR2 15 NOT NULL; (firstName,lastName) UNIQUE

lastName VARCHAR2 20 NOT NULL; (firstName,lastName) UNIQUE

street VARCHAR2 50

zip CHAR 5 FOREIGN KEY REF Zips

TABLE Sale

invoiceNumber NUMBER 6 PRIMARY KEY

artworkId NUMBER 6 NOT NULL; UNIQUE; FOREIGN KEY REF Artwork

amountRemittedToOwner NUMBER 8,2 DEFAULT 0.00

saleDate DATE

salePrice NUMBER 8,2

saleTax NUMBER 6,2

buyerId NUMBER 6 NOT NULL; FOREIGN KEY REF Buyer

salespersonSocialSecurityNumber CHAR 9

* Step 6.4 - Write and execute SQL statements to create all the tables needed to implement the design.

Since we wish to specify foreign keys as we create the tables, we must be careful of the order in which we create, because the “home table” has to exist before the table containing the foreign key is created. Therefore, we will use the following order: Zips, Artist, Collector, Potential Customer, Artwork, Show, ShownIn, Buyer, Salesperson, Sale.

Since this schema is an improvement on the database we created at the end of Chapter 5, we could alter those tables to conform to the new schema. However, to provide more practice in creating tables, we will first drop all the tables for that schema and begin again, creating a complete new schema. The DDL statements to drop the old tables and to create the new tables are shown below, and a text file with these commands, named *DDLTheArtGallery* is in this directory. You should use this file to create your own database. We are using Oracle syntax, but the DDL statements should work, with minor modifications, for any relational DBMS.

-- drop the existing tables for the Art Gallery, in reverse order

drop table Sale;

drop table Salesperson;

drop table Buyer;

drop table ShownIn;

drop table Show;

drop table Artwork;

drop table PotentialCustomer;

drop table Collector;

drop table Artist;

-- create the new tables in normalized form

CREATE TABLE Zips(

zip CHAR(5),

city VARCHAR2(15) NOT NULL,

state CHAR(2) NOT NULL,

CONSTRAINT Zips\_zip\_pk PRIMARY KEY (zip));

CREATE TABLE Artist(

ArtistId NUMBER(6),

firstName VARCHAR2(15) NOT NULL,

lastName VARCHAR2(20) NOT NULL,

interviewDate DATE,

interviewerName VARCHAR2(35),

areaCode CHAR(3),

telephoneNumber CHAR(7),

street VARCHAR2(50),

zip CHAR(5),

salesLastYear NUMBER(8,2),

salesYearToDate NUMBER(8,2),

socialSecurityNumber CHAR(9),

usualMedium VARCHAR(15),

usualStyle VARCHAR(15),

usualType VARCHAR(20),

CONSTRAINT Artist\_ArtistId\_pk PRIMARY KEY (ArtistId),

CONSTRAINT Artist\_socialSecurityNumber\_uk UNIQUE (socialSecurityNumber),

CONSTRAINT Artist\_firstName\_lastName\_uk UNIQUE (firstName, lastName),

CONSTRAINT Artist\_zip\_fk FOREIGN KEY (zip) REFERENCES Zips (zip));

CREATE TABLE Collector(

socialSecurityNumber CHAR(9),

firstName VARCHAR2(15) NOT NULL,

lastName VARCHAR2(20) NOT NULL,

interviewDate DATE,

interviewerName VARCHAR2(35),

areaCode CHAR(3),

telephoneNumber CHAR(7),

street VARCHAR2(50),

zip CHAR(5),

salesLastYear NUMBER(8,2),

salesYearToDate NUMBER(8,2),

collectionArtistId NUMBER(6),

collectionMedium VARCHAR(15),

collectionStyle VARCHAR(15),

collectionType VARCHAR(20),

CONSTRAINT Collector\_SSN\_pk PRIMARY KEY (socialSecurityNumber),

CONSTRAINT Collector\_collArtistid\_fk FOREIGN KEY (collectionArtistId) REFERENCES Artist (artistId),

CONSTRAINT Collector\_zip\_fk FOREIGN KEY (zip) REFERENCES Zips (zip));

CREATE TABLE PotentialCustomer(

potentialCustomerId NUMBER(6),

firstname VARCHAR2(15) NOT NULL,

lastName VARCHAR2(20) NOT NULL,

areaCode CHAR(3),

telephoneNumber CHAR(7),

street VARCHAR2(50),

zip CHAR(5),

dateFilledIn DATE,

preferredArtistId NUMBER(6),

preferredMedium VARCHAR2(15),

preferredStyle VARCHAR2(15),

preferredType VARCHAR2(20),

CONSTRAINT PotCust\_potCustId\_pk PRIMARY KEY (potentialCustomerId),

CONSTRAINT PotentialCustomer\_zip\_fk FOREIGN KEY (zip) REFERENCES Zips(zip),

CONSTRAINT PotCust\_prefArtistId\_fk FOREIGN KEY (preferredArtistId) REFERENCES Artist(artistId));

CREATE TABLE Artwork(

artworkId NUMBER(6),

artistId NUMBER(6)NOT NULL,

workTitle VARCHAR2(50)NOT NULL,

askingPrice NUMBER(8,2),

dateListed DATE,

dateReturned DATE,

dateShown DATE,

status VARCHAR2(15),

workMedium VARCHAR2(15),

workSize VARCHAR2(15),

workStyle VARCHAR2(15),

workType VARCHAR2(20),

workYearCompleted CHAR(4),

collectorSocialSecurityNumber CHAR(9),

CONSTRAINT Artwork\_workId\_pk PRIMARY KEY (artworkId),

CONSTRAINT Artwork\_artistId\_title\_uk UNIQUE (artistId, workTitle),

CONSTRAINT Artwork\_artistId\_fk FOREIGN KEY (artistId) REFERENCES Artist(artistId),

CONSTRAINT Artwork\_collectorSSN\_fk FOREIGN KEY (collectorSocialSecurityNumber) REFERENCES Collector(socialSecurityNumber));

CREATE TABLE Show(

showTitle VARCHAR2(50),

showFeaturedArtistId NUMBER(6),

showClosingDate DATE,

showTheme VARCHAR2(50),

showOpeningDate DATE,

CONSTRAINT Show\_showTitle\_pk PRIMARY KEY (showTitle),

CONSTRAINT Show\_FeatArtistId\_fk FOREIGN KEY (showFeaturedArtistId) REFERENCES Artist (artistId) );

CREATE TABLE ShownIn(

artworkId NUMBER(6),

showTitle VARCHAR2(50),

CONSTRAINT ShownIn\_workid\_showTitle\_pk PRIMARY KEY (artworkId, showTitle),

CONSTRAINT ShownIn\_workId\_fk FOREIGN KEY (artworkId) REFERENCES Artwork (artworkId),

CONSTRAINT ShownIn\_showTitle\_fk FOREIGN KEY (showTitle) REFERENCES Show (showTitle));

CREATE TABLE Buyer(

buyerId NUMBER(6),

firstName VARCHAR2(15)NOT NULL,

lastName VARCHAR2(20) NOT NULL,

street VARCHAR2(50),

zip CHAR(5),

areaCode CHAR(3),

telephoneNumber CHAR(7),

purchasesLastYear NUMBER(8,2),

purchasesYearToDate NUMBER(8,2),

CONSTRAINT Buyer\_buyerId\_pk PRIMARY KEY (buyerId),

CONSTRAINT Buyer\_zip\_fk FOREIGN KEY(zip) REFERENCES Zips(zip) );

CREATE TABLE Salesperson (

socialSecurityNumber CHAR(9),

firstName VARCHAR2(15) NOT NULL,

lastName VARCHAR2(20) NOT NULL,

street VARCHAR2(50),

zip CHAR(5),

CONSTRAINT Salesperson\_SSN\_pk PRIMARY KEY (socialSecurityNumber),

CONSTRAINT Salesperson\_Name\_uk UNIQUE (firstName,lastName),

CONSTRAINT Salesperson\_zip\_fk FOREIGN KEY(zip) REFERENCES Zips(zip));

CREATE TABLE Sale (

invoiceNumber NUMBER(6),

artworkId NUMBER(6) NOT NULL,

amountRemittedToOwner NUMBER(8,2) DEFAULT 0.00,

saleDate DATE,

salePrice NUMBER(8,2),

saleTax NUMBER(6,2),

buyerId NUMBER(6)NOT NULL,

salespersonSSN CHAR(9),

CONSTRAINT Sale\_invoiceNumber\_pk PRIMARY KEY (invoiceNumber),

CONSTRAINT Sale\_artworkId\_uk UNIQUE (artworkId),

CONSTRAINT Sale\_artworkId\_fk FOREIGN KEY (artworkId) REFERENCES Artwork(artworkId),

CONSTRAINT Sale\_buyerId\_fk FOREIGN KEY (buyerId) REFERENCES Buyer (buyerId),

CONSTRAINT Sale\_salespersonSSN\_fk FOREIGN KEY (salespersonSSN) REFERENCES Salesperson (socialSecurityNumber));

* Step 6.5 Create indexes for foreign keys and any other columns that will be used most often for queries.

The DDL statements to create the indexes are shown below and are stored in a text file in this directory called *IndexesforTheArtGallery.* Use that file to create indexes for your database.

CREATE UNIQUE INDEX Artist\_lastName\_firstName ON Artist(lastName, firstName);

CREATE UNIQUE INDEX Artist\_socialSecurityNumber ON Artist(socialSecurityNumber);

CREATE INDEX Artist\_zip ON Artist(zip);

CREATE INDEX Collector\_collectionArtistId ON Collector(collectionArtistId);

CREATE INDEX Collector\_zip ON Collector(zip);

CREATE INDEX Collector\_lastName\_firstName ON Collector(lastName, firstName);

CREATE INDEX PotentialCustomer\_zip ON PotentialCustomer(zip);

CREATE INDEX PotCust\_prefArtistId ON PotentialCustomer(preferredArtistId);

CREATE INDEX PotCust\_last\_first ON PotentialCustomer(lastName, firstName);

CREATE INDEX Artwork\_artistId ON Artwork(artistId);

CREATE INDEX Artwork\_collectorSSN ON Artwork(collectorSocialSecurityNumber);

CREATE INDEX Show\_FeatArtistId ON Show (showFeaturedArtistId);

CREATE INDEX ShownIn\_artworkId ON ShownIn (artworkId);

CREATE INDEX ShownIn\_showTitle ON ShownIn (showTitle);

CREATE INDEX Buyer\_zip ON Buyer(zip);

CREATE INDEX Buyer\_last\_first ON Buyer (lastName, firstName);

CREATE UNIQUE INDEX Salesperson\_last\_first ON Salesperson (lastName, firstName);

CREATE INDEX Salesperson\_zip ON Salesperson (zip);

CREATE UNIQUE INDEX Sale\_artworkId ON Sale (artworkId);

CREATE INDEX Sale\_buyerId ON Sale (buyerId);

* Step 6.6 - Insert about five records in each table, preserving all constraints. Put in enough data to demonstrate how the database will function.

The INSERT statements are shown below and are stored in a text file in this directory called *DataforTheArtGallery.* Use that file to create the INSERT statements for your database.

Although it is easier to use identity fields for identifiers, we wish to demonstrate Oracle’s sequences so we created sequences for each of artistId, potentialCustomerId, artworkId, and buyerId, using the command

CREATE SEQUENCE *sequence-name*

[START WITH *starting-value*]

[INCREMENT BY *step*]…;

We chose to start with 1 and increment by 1, the defaults. To generate each new value, we use the command <*sequence\_name>*.NEXTVAL, as shown in the INSERT commands. We assume invoiceNumber is a number preprinted on the invoice form, not system-generated, so we do not need a sequence for that number.

INSERT INTO Zips VALUES('10101', 'New York', 'NY');

INSERT INTO Zips VALUES('10801', 'New Rochelle', 'NY');

INSERT INTO Zips VALUES('92101', 'San Diego', 'CA');

INSERT INTO Zips VALUES('33010', 'Miami', 'FL');

INSERT INTO Zips VALUES('60601', 'Chicago', 'IL');

CREATE SEQUENCE artistId\_sequence;

INSERT INTO Artist VALUES(artistId\_sequence.NEXTVAL, 'Leonardo', 'Vincenti', '10-Oct-2015', 'Hughes', '212','5559999','10 Main Street', '10101', 9000, 4500,'099999876', 'oil', 'realism', 'painting');

INSERT INTO Artist VALUES(artistId\_sequence.NEXTVAL, 'Vincent', 'Gogh', '15-Jun-2014', 'Hughes', '914','5551234','55 West 18 Street', '10801', 9500, 5500,'099999877', 'oil', 'impressionism', 'painting');

INSERT INTO Artist VALUES(artistId\_sequence.NEXTVAL, 'Winslow', 'Homes', '05-Jan-2015', 'Hughes', '619', '1234567','100 Water Street', '92101', 14000, 4000,'083999876', 'watercolor', 'realism', 'painting');

INSERT INTO Artist VALUES(artistId\_sequence.NEXTVAL, 'Alexander', 'Calderone', '10-Feb-2014', 'Hughes', '212','5559999', '10 Main Street', '10101', 20000, 20000,'123999876', 'steel', 'cubism', 'sculpture');

INSERT INTO Artist VALUES(artistId\_sequence.NEXTVAL, 'Georgia', 'Keefe', '05-Oct-2015', 'Hughes', '305', '1239999','5 Chestnut Street', '33010', 19000, 14500,'987999876', 'oil', 'realism', 'painting');

INSERT INTO Collector VALUES('102345678', 'John','Jackson', '01-Feb-2015', 'Hughes', '917', '7771234','24 Pine Avenue','10101', 4000,3000, 1, 'oil', 'realism', 'collage');

INSERT INTO Collector VALUES('987654321', 'Mary','Lee', '01-Mar-2014', 'Jones', '305', '5551234','10 Ash Street',33010, '2000',3000, 2, 'watercolor', 'realism', 'painting');

INSERT INTO Collector VALUES('034345678', 'Ramon','Perez', '15-Apr-2015', 'Hughes', '619', '8881234','15 Poplar Avenue','92101', 4500,3500, 3, 'oil', 'realism', 'painting');

INSERT INTO Collector VALUES('888881234', 'Rick','Lee', '20-Jun-2015', 'Hughes', '212', '9991234','24 Pine Avenue','10101', 4000,3000, 3, 'oil', 'realism', 'sculpture');

INSERT INTO Collector VALUES('777345678', 'Samantha','Torno', '05-May-2015', 'Jones', '305', '5551234','10 Ash Street','33010', 40000,30000, 1, 'acrylic', 'realism', 'painting');

CREATE SEQUENCE potentialCustomerId\_sequence;

INSERT INTO PotentialCustomer VALUES(potentialCustomerId\_sequence.NEXTVAL, 'Adam','Burns','917','3456789', '1 Spruce Street', '10101', '12-Dec-2014', 1, 'watercolor', 'impressionism', 'painting');

INSERT INTO PotentialCustomer VALUES(potentialCustomerId\_sequence.NEXTVAL, 'Carole','Burns','917','3456789', '1 Spruce Street', '10101', '12-Dec-2015', 2, 'watercolor', 'realism', 'sculpture');

INSERT INTO PotentialCustomer VALUES(potentialCustomerId\_sequence.NEXTVAL, 'David','Engel','914','7777777', '715 North Avenue', '10801', '08-Aug-2015', 3, 'watercolor', 'realism', 'painting');

INSERT INTO PotentialCustomer VALUES(potentialCustomerId\_sequence.NEXTVAL, 'Frances','Hughes','619','3216789', '10 Pacific Avenue', '92101', '05-Jan-2015', 2, 'oil', 'impressionism', 'painting');

INSERT INTO PotentialCustomer VALUES(potentialCustomerId\_sequence.NEXTVAL, 'Irene','Jacobs','312','1239876', '1 Windswept Place', '60601', '21-Sep-2015', 5, 'watercolor', 'abst expression', 'painting');

CREATE SEQUENCE artworkId\_sequence;

INSERT INTO Artwork VALUES(artworkid\_sequence.NEXTVAL, 1, 'Flight', 15000.00, '08-Sep-2015',null,null, 'for sale','oil', '36 in X 48 in', 'realism', 'painting','2014', null);

INSERT INTO Artwork VALUES(artworkid\_sequence.NEXTVAL, 3, 'Bermuda Sunset', 8000.00, '15-Mar-2015',null ,'01-Apr-2015' , 'sold','watercolor', '22 in X 28 in', 'realism', 'painting','2014', null);

INSERT INTO Artwork VALUES(artworkid\_sequence.NEXTVAL, 3, 'Mediterranean Coast', 4000.00, '18-Oct-2014',null ,'01-Apr-2015', 'for sale','watercolor', '22 in X 28 in', 'realism', 'painting','2001','102345678');

INSERT INTO Artwork VALUES(artworkid\_sequence.NEXTVAL, 5, 'Ghost orchid', 18000.00, '05-Jun-2015',null,null , 'sold','oil', '36 in X 48 in', 'realism', 'painting','2014','034345678' );

INSERT INTO Artwork VALUES(artworkid\_sequence.NEXTVAL, 4, 'Five Planes', 15000.00, '10-Jan-2015',null ,'10-Mar-2015' , 'for sale','steel', '36 X 30 X 60 in', 'cubism', 'sculpture','2015','034345678' );

INSERT INTO Show VALUES('The Sea in Watercolor',3, '30-Apr-2015','seascapes','01-Apr-2015');

INSERT INTO Show VALUES('Calderone''s Mastery of Space',4,'20-Mar-2015',null, '10-Mar-2015');

INSERT INTO ShownIn VALUES(2,'The Sea in Watercolor');

INSERT INTO ShownIn VALUES(3, 'The Sea in Watercolor');

INSERT INTO ShownIn VALUES(5, 'Calderone''s Mastery of Space');

CREATE SEQUENCE buyerId\_sequence;

INSERT INTO Buyer VALUES (BuyerId\_sequence.NEXTVAL, 'Valerie', 'Smiley', '15 Hudson Street', '10101', '718','5551234', 5000, 7500);

INSERT INTO Buyer VALUES (BuyerId\_sequence.NEXTVAL, 'Winston', 'Lee', '20 Liffey Avenue', '60601', '312','7654321', 3000, 0);

INSERT INTO Buyer VALUES (BuyerId\_sequence.NEXTVAL, 'Samantha', 'Babson', '25 Thames Lane', '92101', '619','4329876', 15000, 0);

INSERT INTO Buyer VALUES (BuyerId\_sequence.NEXTVAL, 'John', 'Flagg', '22 Amazon Street', '10101', '212','7659876', 3000, 0);

INSERT INTO Buyer VALUES (BuyerId\_sequence.NEXTVAL, 'Terrence', 'Smallshaw', '5 Nile Street', '33010', '305','2323456', 15000, 17000);

INSERT INTO Salesperson VALUES('102445566', 'John','Smith', '10 Sapphire Row', '10801');

INSERT INTO Salesperson VALUES('121344321', 'Alan','Hughes', '10 Diamond Street', '10101');

INSERT INTO Salesperson VALUES('101889988', 'Mary','Brady', '10 Pearl Avenue', '10801');

INSERT INTO Salesperson VALUES('111223344', 'Jill','Fleming', '10 Ruby Row', '10101');

INSERT INTO Salesperson VALUES('123123123', 'Terrence','DeSimone', '10 Emerald Lane', '10101');

INSERT INTO Sale VALUES(1234, 2,0 ,'05-Apr-2015',7500,600, 1, '102445566');

INSERT INTO Sale VALUES(1235, 4, 0,'06-Apr-2015',17000,1360, 5, '121344321');

* Step 6.7 Write SQL statements that will process five non-routine requests for information from the database just created. For each, write the request in English, followed by the corresponding SQL command.

The queries shown below are stored in a text file in this directory called *QueriesforTheArtGallery*. You should copy and execute each query to see the results, and devise and execute additional queries that you design yourself.

1. Find the names of all artists who were interviewed after January 1, 2014 but who have no works of art listed.

SELECT firstName, lastName

FROM Artist

WHERE interviewDate> '01-Jan-2014' AND NOT EXISTS

(SELECT \*

FROM Artwork

WHERE artistId =Artist.artistId);

2. Find the total commission for salesperson John Smith earned between the dates April 1, 2015 and April 15,2015. Recall that the gallery charges 10% commission, and the salesperson receives one-half of that, which is 5% --of the selling price.

SELECT .05 \* SUM(salePrice)

FROM Sale

WHERE saleDate>='01-Apr-2011' AND saleDate<='15-Apr-2011' AND

salespersonSSN = (SELECT socialSecurityNumber

FROM Salesperson

WHERE firstName= 'John' AND lastName ='Smith');

3. Find the collector names, artist names and titles of all artworks that are owned by collectors, not by the --artists themselves, in order by the collector’s last name.

SELECT Collector.firstName, Collector.lastName, Artist.firstName, Artist.lastName, workTitle

FROM Artist, Artwork, Collector

WHERE Artist.artistId = Artwork.artistId AND Artwork.collectorsocialSecurityNumber = Collector.socialSecurityNumber AND collectorsocialSecurityNumber IS NOT NULL

ORDER BY Collector.lastName, Collector.firstName;

4. For each potential buyer, find information about shows that feature his or her preferred artist.

SELECT firstName, lastName, showTitle, showOpeningDate, showClosingDate

FROM Show, PotentialCustomer

WHERE showFeaturedArtistId = PotentialCustomer.preferredArtistId

ORDER BY potentialCustomerId;

5. Find the average sale price of works of artist Georgia Keefe.

SELECT AVG(salePrice)

FROM Sale

WHERE artworkId IN (SELECT artworkId

FROM Artwork

WHERE artistId = (SELECT ArtistId

FROM Artist

WHERE lastName = 'Keefe' AND firstName ='Georgia'));

* Step 6.8 Create at least one trigger and write the code for it.

Code for the trigger is stored in a text file in this directory called *TriggerforTheArtGallery*. You should copy that file and create the trigger, then test it by adding a new Sale record and observing whether the corresponding Buyer record is updated.

--This trigger will update the amount of the buyer’s purchases year-to-date whenever a sale is completed.

create or replace trigger UPDATEBUYERYTD

after insert on Sale

for each row

begin

update Buyer

set purchasesYearToDate = purchasesYearToDate + :NEW.salePrice

where Buyer.buyerID = :NEW.buyerID;

end;

/