David M. Kroenke and David J. Auer
Database Processing:
Fundamentals, Design, and Implementation

Chapter Seven:
SQL for Database Construction and Application Processing

Chapter Objectives

- To create and manage table structures using SQL statements
- To understand how referential integrity actions are implemented in SQL statements
- To create and use SQL constraints
- To understand several uses for SQL views
- To use SQL statements to create and use views
- To understand how SQL is used in an application programming
- To understand SQL/Persistent Stored Modules (SQL/PSM)
- To understand how to create and use functions
- To understand how to create and use triggers
- To understand how to create and use stored procedures

SQL DML—INSERT I

- The **SQL INSERT statement**:

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-INSERT-CH07-01 *** */
INSERT INTO ARTIST
     (LastName, FirstName, Nationality, DateOfBirth, DateDeceased)
     VALUES ('Miro', 'Joan', 'Spanish', 1893, 1983);
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-INSERT-CH07-02 *** */
INSERT INTO ARTIST VALUES
     ('Miro', 'Joan', 'Spanish', 1893, 1983);
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-INSERT-CH07-04 *** */
INSERT INTO ARTIST
     (LastName, FirstName, Nationality)
     VALUES ('Miro', 'Joan', 'Spanish');
```

SQL DML—INSERT II

- Bulk INSERT:

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-INSERT-CH07-05 *** */
INSERT INTO ARTIST
     (LastName, FirstName, Nationality, DateOfBirth, DateDeceased)
     SELECT      LastName, FirstName, Nationality,
                 DateOfBirth, DateDeceased
     FROM        IMPORTED_ARTIST;
```

Populating the VRG Tables I

- The VRG database data contain
  non-sequential surrogate key values.

ArtistID	LastName	FirstName	Nationality	DateOfBirth	DateDeceased
1	Miro	Joan	Spanish	1893	1983
2	Kandinsky	Wassily	Russian	1866	1944
3	Klee	Paul	German	1879	1940
4	Matisse	Henri	French	1869	1954
5	Chagall	Marc	French	1887	1985
11	Sargent	John Singer	United States	1856	1925
17	Tobey	Mark	United States	1890	1976
18	Horiuchi	Paul	United States	1906	1999
19	Graves	Morris	United States	1920	2001

Populating the VRG Tables II

- Cannot just use SQL INSERT statement by itself.
- See discussions of how to handle this situation:
  - Chapter 10A - Microsoft SQL Server 2014
  - Chapter 10B - Oracle Database
  - Chapter 10C - MySQL 5.6

SQL DML—UPDATE I

- The **SQL UPDATE statement**:

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-UPDATE-CH07-01 *** */
UPDATE          CUSTOMER
     SET           City = 'New York City'
     WHERE         CustomerID = 1000;
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-UPDATE-CH07-02 *** */
UPDATE          CUSTOMER
     SET           City = 'New York City', State = 'NY'
     WHERE         CustomerID = 1000;
```

SQL DML—UPDATE II

- Bulk UPDATE:

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-UPDATE-CH07-03 *** */
UPDATE          CUSTOMER
     SET           City = 'New York City';
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-UPDATE-CH07-04 *** */
UPDATE          CUSTOMER
     SET           AreaCode = '303'
     WHERE         City = 'Denver';
```

SQL DML—UPDATE III

- Using values from other tables:

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-UPDATE-CH07-06 *** */
UPDATE              PURCHASE_ORDER
     SET            TaxRate =
                    (SELECT   Tax
                     From     TAX_TABLE
                     WHERE    TAX_TABLE.City = PURCHASE_ORDER.City)
     WHERE          PURCHASE_ORDER.Number = 1000;
```

SQL DML—MERGE

- The **SQL MERGE statement**:

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-MERGE-CH07-01 *** */
MERGE INTO ARTIST AS A USING ARTIST_DATA_RESEARCH AS ADR
    ON   (A.LastName = ADR.LastName
          AND
          A.FirstName = ADR.FirstName)
WHEN MATCHED THEN
    UPDATE SET
         A.Nationality = ADR.Nationality,
         A.DateOfBirth = ADR.DateOfBirth,
         A.DateDeceased = ADR.DateDeceased
WHEN NOT MATCHED THEN
    INSERT (LastName, FirstName, Nationality,
            DateOfBirth, DateDeceased);
```

SQL DML—DELETE

- SQL DELETE statement:

```
/* *** EXAMPLE CODE – DO NOT RUN *** */
/* *** SQL-DELETE-CH07-01 *** */
DELETE        FROM CUSTOMER
WHERE         CustomerID = 1000;
```

- If you omit the WHERE clause, you will delete every row in the table.
- Does *not* reset surrogate key values.

Using Aliases

- **Use of aliases:**

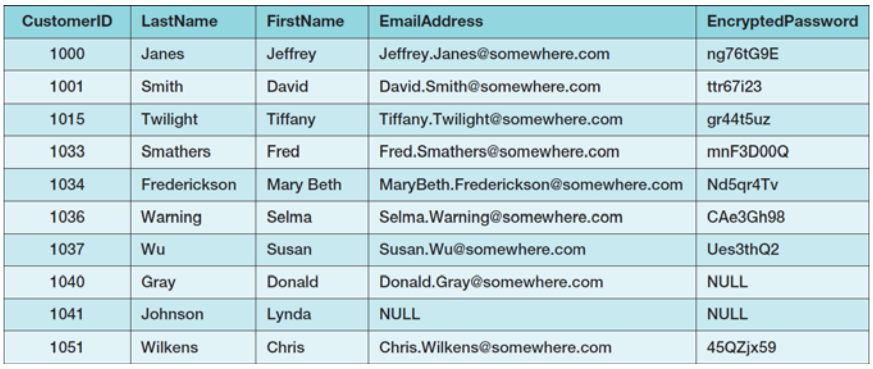
```
SELECT    C.Name, A.Name
FROM      CUSTOMER AS C
          JOIN
          CUSTOMER_ARTIST_INT AS CI
  ON      C.CustomerID = CI.CustomerID
          JOIN ARTIST AS A
                ON CI.ArtistID = A.ArtistID;
```

- DBMS products differ

  `CUSTOMER AS C` versus `CUSTOMER C`

VRG Database Data

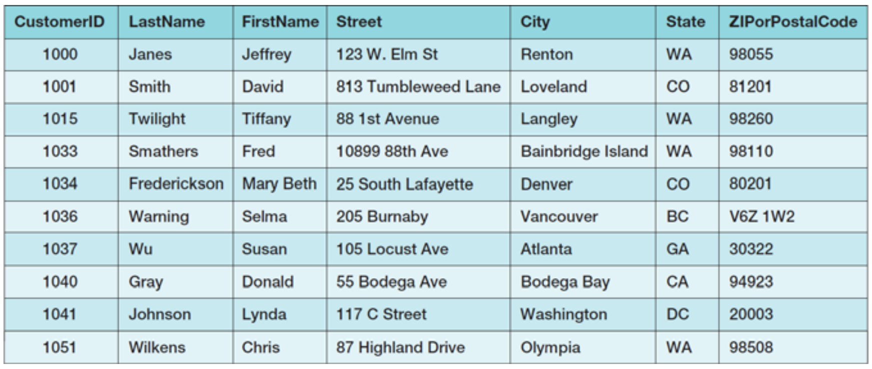
CUSTOMER I



CustomerID	LastName	FirstName	EmailAddress	EncryptedPassword
1000	Janes	Jeffrey	Jeffrey.Janes@somewhere.com	ng76tG9E
1001	Smith	David	David.Smith@somewhere.com	ttr67i23
1015	Twilight	Tiffany	Tiffany.Twilight@somewhere.com	gr44t5uz
1033	Smathers	Fred	Fred.Smathers@somewhere.com	mnF3D00Q
1034	Frederickson	Mary Beth	MaryBeth.Frederickson@somewhere.com	Nd5qr4Tv
1036	Warning	Selma	Selma.Warning@somewhere.com	CAe3Gh98
1037	Wu	Susan	Susan.Wu@somewhere.com	Ues3thQ2
1040	Gray	Donald	Donald.Gray@somewhere.com	NULL
1041	Johnson	Lynda	NULL	NULL
1051	Wilkens	Chris	Chris.Wilkens@somewhere.com	45QZjx59

VRG Database Data

CUSTOMER II



CustomerID	LastName	FirstName	Street	City	State	ZIPorPostalCode
1000	Janes	Jeffrey	123 W. Elm St	Renton	WA	98055
1001	Smith	David	813 Tumbleweed Lane	Loveland	CO	81201
1015	Twilight	Tiffany	88 1st Avenue	Langley	WA	98260
1033	Smathers	Fred	10899 88th Ave	Bainbridge Island	WA	98110
1034	Frederickson	Mary Beth	25 South Lafayette	Denver	CO	80201
1036	Warning	Selma	205 Burnaby	Vancouver	BC	V6Z 1W2
1037	Wu	Susan	105 Locust Ave	Atlanta	GA	30322
1040	Gray	Donald	55 Bodega Ave	Bodega Bay	CA	94923
1041	Johnson	Lynda	117 C Street	Washington	DC	20003
1051	Wilkens	Chris	87 Highland Drive	Olympia	WA	98508

VRG Database Data

CUSTOMER III

CustomerID	LastName	FirstName	Country	AreaCode	PhoneNumber
1000	Janes	Jeffrey	USA	425	543-2345
1001	Smith	David	USA	970	654-9876
1015	Twilight	Tiffany	USA	360	765-5566
1033	Smathers	Fred	USA	206	876-9911
1034	Frederickson	Mary Beth	USA	303	513-8822
1036	Warning	Selma	Canada	604	988-0512
1037	Wu	Susan	USA	404	653-3465
1040	Gray	Donald	USA	707	568-4839
1041	Johnson	Lynda	USA	202	438-5498
1051	Wilkens	Chris	USA	360	876-8822

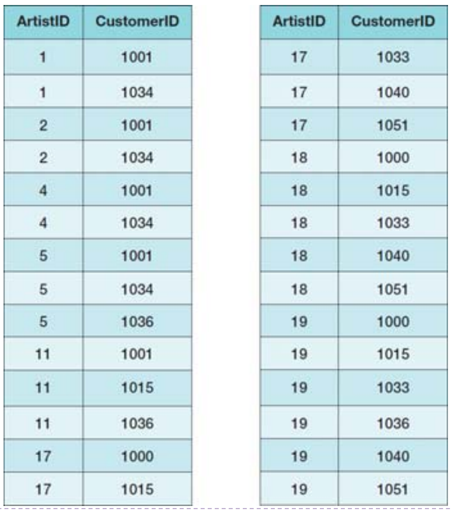
VRG Database Data

ARTIST

ArtistID	LastName	FirstName	Nationality	DateOfBirth	DateDeceased
1	Miro	Joan	Spanish	1893	1983
2	Kandinsky	Wassily	Russian	1866	1944
3	Klee	Paul	German	1879	1940
4	Matisse	Henri	French	1869	1954
5	Chagall	Marc	French	1887	1985
11	Sargent	John Singer	United States	1856	1925
17	Tobey	Mark	United States	1890	1976
18	Horiuchi	Paul	United States	1906	1999
19	Graves	Morris	United States	1920	2001

VRG Database Data

CUSTOMER_ARTIST_INT

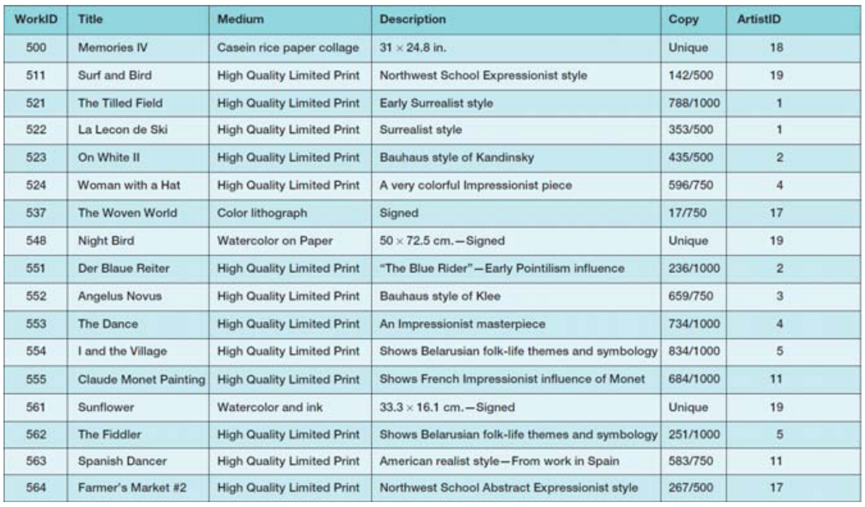


ArtistID	CustomerID
1	1001
1	1034
2	1001
2	1034
4	1001
4	1034
5	1001
5	1034
5	1036
11	1001
11	1015
11	1036
17	1000
17	1015

ArtistID	CustomerID
17	1033
17	1040
17	1051
18	1000
18	1015
18	1033
18	1040
18	1051
19	1000
19	1015
19	1033
19	1036
19	1040
19	1051

VRG Database Data

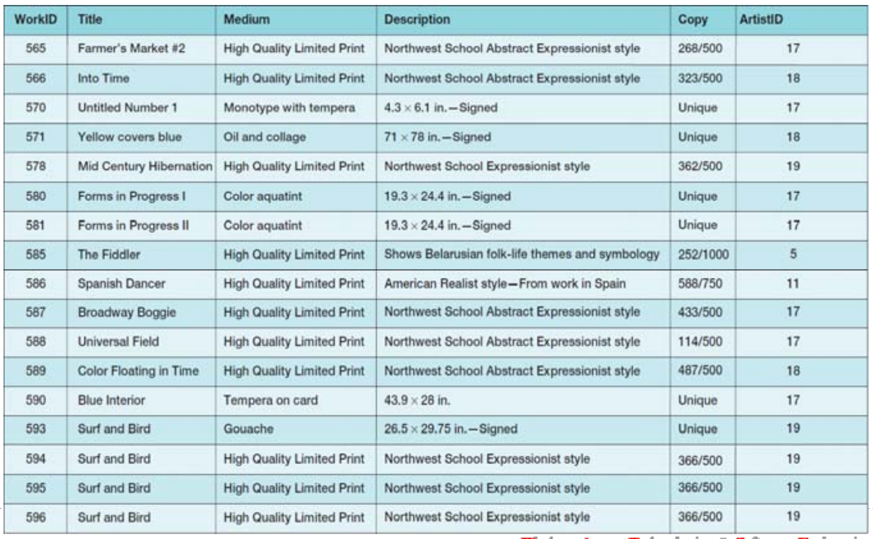
WORK I



WorkID	Title	Medium	Description	Copy	ArtistID
500	Memories IV	Casein rice paper collage	31 × 24.8 in.	Unique	18
511	Surf and Bird	High Quality Limited Print	Northwest School Expressionist style	142/500	19
521	The Tilled Field	High Quality Limited Print	Early Surrealist style	788/1000	1
522	La Lecon de Ski	High Quality Limited Print	Surrealist style	353/500	1
523	On White II	High Quality Limited Print	Bauhaus style of Kandinsky	435/500	2
524	Woman with a Hat	High Quality Limited Print	A very colorful Impressionist piece	596/750	4
537	The Woven World	Color lithograph	Signed	17/750	17
548	Night Bird	Watercolor on Paper	50 × 72.5 cm.—Signed	Unique	19
551	Der Blaue Reiter	High Quality Limited Print	"The Blue Rider"—Early Pointilism influence	236/1000	2
552	Angelus Novus	High Quality Limited Print	Bauhaus style of Klee	659/750	3
553	The Dance	High Quality Limited Print	An Impressionist masterpiece	734/1000	4
554	I and the Village	High Quality Limited Print	Shows Belarusian folk-life themes and symbology	834/1000	5
555	Claude Monet Painting	High Quality Limited Print	Shows French Impressionist influence of Monet	684/1000	11
561	Sunflower	Watercolor and ink	33.3 × 16.1 cm.—Signed	Unique	19
562	The Fiddler	High Quality Limited Print	Shows Belarusian folk-life themes and symbology	251/1000	5
563	Spanish Dancer	High Quality Limited Print	American realist style—From work in Spain	583/750	11
564	Farmer's Market #2	High Quality Limited Print	Northwest School Abstract Expressionist style	267/500	17

VRG Database Data

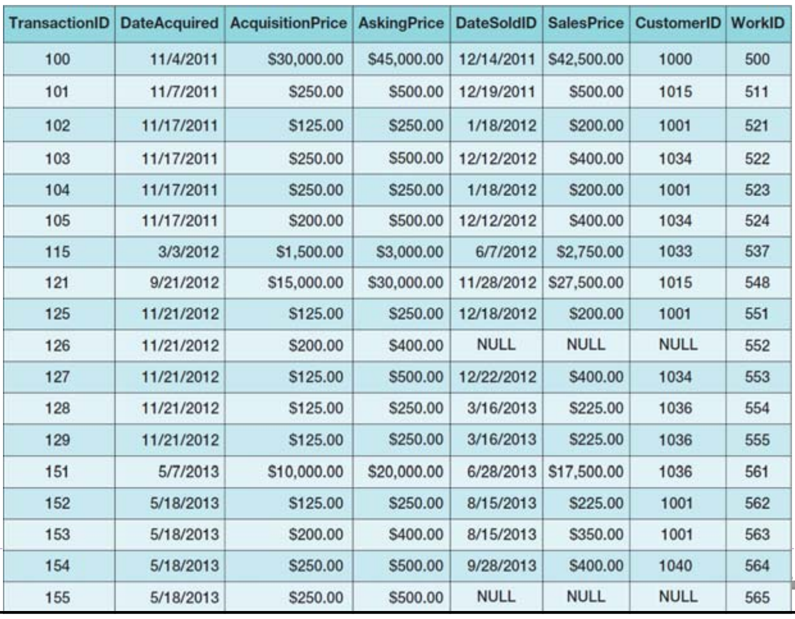
WORK II



WorkID	Title	Medium	Description	Copy	ArtistID
565	Farmer's Market #2	High Quality Limited Print	Northwest School Abstract Expressionist style	268/500	17
566	Into Time	High Quality Limited Print	Northwest School Abstract Expressionist style	323/500	18
570	Untitled Number 1	Monotype with tempera	4.3 × 6.1 in.—Signed	Unique	17
571	Yellow covers blue	Oil and collage	71 × 78 in.—Signed	Unique	18
578	Mid Century Hibernation	High Quality Limited Print	Northwest School Expressionist style	362/500	19
580	Forms in Progress I	Color aquatint	19.3 × 24.4 in.—Signed	Unique	17
581	Forms in Progress II	Color aquatint	19.3 × 24.4 in.—Signed	Unique	17
585	The Fiddler	High Quality Limited Print	Shows Belarusian folk-life themes and symbology	252/1000	5
586	Spanish Dancer	High Quality Limited Print	American Realist style—From work in Spain	588/750	11
587	Broadway Boggie	High Quality Limited Print	Northwest School Abstract Expressionist style	433/500	17
588	Universal Field	High Quality Limited Print	Northwest School Abstract Expressionist style	114/500	17
589	Color Floating in Time	High Quality Limited Print	Northwest School Abstract Expressionist style	487/500	18
590	Blue Interior	Tempera on card	43.9 × 28 in.	Unique	17
593	Surf and Bird	Gouache	26.5 × 29.75 in.—Signed	Unique	19
594	Surf and Bird	High Quality Limited Print	Northwest School Expressionist style	366/500	19
595	Surf and Bird	High Quality Limited Print	Northwest School Expressionist style	366/500	19
596	Surf and Bird	High Quality Limited Print	Northwest School Expressionist style	366/500	19

VRG Database Data

TRANS I



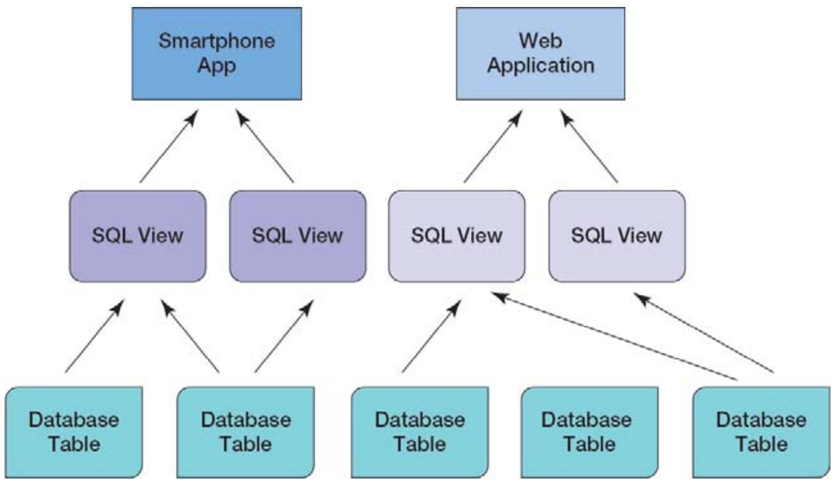
TransactionID	DateAcquired	AcquisitionPrice	AskingPrice	DateSoldID	SalesPrice	CustomerID	WorkID
100	11/4/2011	$30,000.00	$45,000.00	12/14/2011	$42,500.00	1000	500
101	11/7/2011	$250.00	$500.00	12/19/2011	$500.00	1015	511
102	11/17/2011	$125.00	$250.00	1/18/2012	$200.00	1001	521
103	11/17/2011	$250.00	$500.00	12/12/2012	$400.00	1034	522
104	11/17/2011	$250.00	$250.00	1/18/2012	$200.00	1001	523
105	11/17/2011	$200.00	$500.00	12/12/2012	$400.00	1034	524
115	3/3/2012	$1,500.00	$3,000.00	6/7/2012	$2,750.00	1033	537
121	9/21/2012	$15,000.00	$30,000.00	11/28/2012	$27,500.00	1015	548
125	11/21/2012	$125.00	$250.00	12/18/2012	$200.00	1001	551
126	11/21/2012	$200.00	$400.00	NULL	NULL	NULL	552
127	11/21/2012	$125.00	$500.00	12/22/2012	$400.00	1034	553
128	11/21/2012	$125.00	$250.00	3/16/2013	$225.00	1036	554
129	11/21/2012	$125.00	$250.00	3/16/2013	$225.00	1036	555
151	5/7/2013	$10,000.00	$20,000.00	6/28/2013	$17,500.00	1036	561
152	5/18/2013	$125.00	$250.00	8/15/2013	$225.00	1001	562
153	5/18/2013	$200.00	$400.00	8/15/2013	$350.00	1001	563
154	5/18/2013	$250.00	$500.00	9/28/2013	$400.00	1040	564
155	5/18/2013	$250.00	$500.00	NULL	NULL	NULL	565

VRG Database Data

TRANS II

TransactionID	DateAcquired	AcquisitionPrice	AskingPrice	DateSoldID	SalesPrice	CustomerID	WorkID
156	5/18/2013	$250.00	$500.00	9/27/2013	$400.00	1040	566
161	6/28/2013	$7,500.00	$15,000.00	9/29/2013	$13,750.00	1033	570
171	8/23/2013	$35,000.00	$60,000.00	9/29/2013	$55,000.00	1000	571
175	9/29/2013	$40,000.00	$75,000.00	12/18/2013	$72,500.00	1036	500
181	10/11/2013	$250.00	$500.00	NULL	NULL	NULL	578
201	2/28/2014	$2,000.00	$3,500.00	4/26/2014	$3,250.00	1040	580
202	2/28/2014	$2,000.00	$3,500.00	4/26/2014	$3,250.00	1040	581
225	6/8/2014	$125.00	$250.00	9/27/2014	$225.00	1051	585
226	6/8/2014	$200.00	$400.00	NULL	NULL	NULL	586
227	6/8/2014	$250.00	$500.00	9/27/2014	$475.00	1051	587
228	6/8/2014	$250.00	$500.00	NULL	NULL	NULL	588
229	6/8/2014	$250.00	$500.00	NULL	NULL	NULL	589
241	8/29/2014	$2,500.00	$5,000.00	9/27/2014	$4,750.00	1015	590
251	10/25/2014	$25,000.00	$50,000.00	NULL	NULL	NULL	593
252	10/27/2014	$250.00	$500.00	NULL	NULL	NULL	594
253	10/27/2014	$250.00	$500.00	NULL	NULL	NULL	595
254	10/27/2014	$250.00	$500.00	NULL	NULL	NULL	596

SQL Views I



Wireless Access Technologies & Software Engineering

SQL Views II

- An SQL view is a virtual table that is constructed from other tables or views.
- It has no data of its own, but obtains data from tables or other views.
- SELECT statements are used to define views:
  - A view definition may not include an ORDER BY clause.
- SQL views are a subset of the external views:
  - They can be used only for external views that involve one multivalued path through the schema.

SQL Views

Uses of SQL Views
Hide columns or rows.
Display results of computations.
Hide complicated SQL syntax.
Layer built-in functions.
Provide level of isolation between table data and users' view of data.
Assign different processing permissions to different views of the same table.
Assign different triggers to different views of the same table.

SQL CREATE VIEW Statement I

- The **SQL CREATE VIEW statement**:

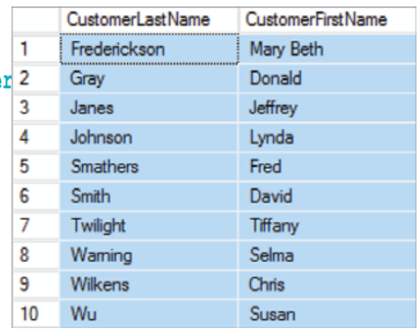
```
/* *** SQL-CREATE-VIEW-CH07-01 *** */
CREATE VIEW CustomerNameView AS
    SELECT        LastName AS CustomerLastName,
                  FirstName AS CustomerFirstName
    FROM          CUSTOMER;
```

- In the SQL standard, views do *not* support the **SQL ORDER BY clause**.
  - Individual DBMS products may support the SQL ORDER BY clause – see documentation.

SQL CREATE VIEW Statement II

- To see the results, use an **SQL SELECT statement** with the **view name** as the table name in the FROM clause:

```
/* *** SQL-Query-View-CH07-01 *** */
SELECT        *
FROM          CustomerNameView
ORDER BY      CustomerLastName, Customer
```



	CustomerLastName	CustomerFirstName
1	Frederickson	Mary Beth
2	Gray	Donald
3	Janes	Jeffrey
4	Johnson	Lynda
5	Smathers	Fred
6	Smith	David
7	Twilight	Tiffany
8	Waming	Selma
9	Wilkens	Chris
10	Wu	Susan

SQL ALTER VIEW Statement I

- The **SQL ALTER VIEW statement**:

```
/* *** EXAMPLE CODE – DO NOT RUN *** */
/* *** SQL-ALTER-VIEW-CH07-01 *** */
ALTER VIEW CustomerNameView AS
    SELECT        FirstName AS CustomerFirstName,
                  LastName AS CustomerLastName,
    FROM          CUSTOMER;
```

- In the Oracle Database or MySQL 5.6, use the **SQL CREATE OR REPLACE VIEW statement**.
  - This allows creation and modification of SQL VIEW code.

Updateable Views

Updatable Views
View based on a single table with no computed columns and all non-null columns present in the view.
View based on any number of tables, with or without computed columns, and INSTEAD OF trigger defined for the view.
Possibly Updatable Views
Based on a single table, primary key in view, some required columns missing from view, update and delete may be allowed. Insert is not allowed.
Based on multiple tables, updates may be allowed on the most subordinate table in the view if rows of that table can be uniquely identified.

Embedding SQL in Program Code

- SQL cursors are used to select one row at a time from pseudo-files.
- Problem: assigning SQL table columns with program variables
  - Solution: object-oriented programming, PL/SQL
- Problem: paradigm mismatch between SQL and application programming language:
  - SQL statements return sets of rows; an application works on one row at a time
  - Solution: process the SQL results as pseudo-files

Variables in Program Code

- Oracle Database and MySQL 5.6 style variables:

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-Code-Example-CH07-01 *** */
SELECT        Count(*) INTO rowCount
FROM          CUSTOMER;
```

- Microsoft SQL Server 2014 style variables:

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-Code-Example-CH07-02 *** */
SELECT        @rowCount = Count(*)
FROM          CUSTOMER;
```

SQL Cursors in Program Code

- SQL can be embedded in triggers, stored procedures, and program code.
- A typical cursor code pattern is:

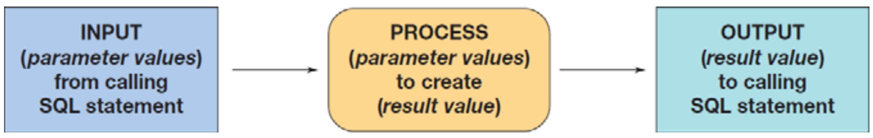
```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-Code-Example-CH07-04 *** */
DECLARE SQLCursor CURSOR FOR (SELECT * FROM CUSTOMER);
/* Opening SQLcursor executes (SELECT * FROM CUSTOMER) */
OPEN SQLcursor;
MOVE SQLcursor to first row of (SELECT * FROM CUSTOMER);
  WHILE (SQLcursor not past the last row) LOOP
    SET CustomerLastName = LastName;
    ...other statements...
    REPEAT LOOP UNTIL DONE;
CLOSE SQLcursor
...other processing...
```

SQL/Persistent Stored Modules (SSL/PSM)

- **SQL/Persistent Stored Modules (SQL/PSM)** is an ANSI/ISO standard for embedding procedural programming functionality into SQL
- Each DBMS product implements SQL/PSM in a different way, with some closer to the standard than others.
  - Microsoft SQL Server 2014 calls its version **Transact-SQL (T-SQL)**.
  - Oracle Database calls its variant **Procedural Language/SQL (PL/SQL)**.
  - MySQL 5.6 implements SQL/PSM, but has no special name for its variant of SQL.

User-Defined Functions I

- A **user-defined function (stored function)** is a stored set of SQL statements that:
  - is *called by name* from another SQL statement
  - may have *input parameters* passed to it by the calling SQL statement, and
  - *returns an output value* to the SQL statement hat called the function.



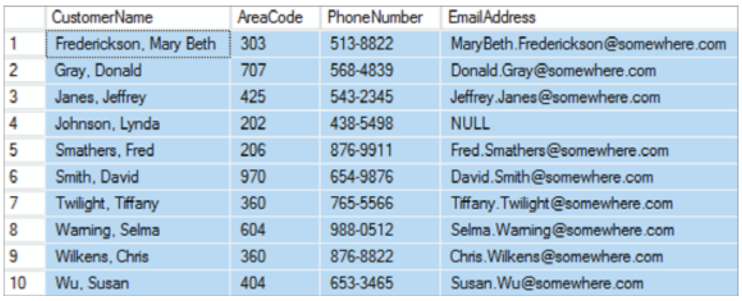
User-Defined Functions II
The *NameConcatenation* Function

```
CREATE FUNCTION dbo.NameConcatenation

-- These are the input parameters
       (
       @FirstName    CHAR(25),
       @LastName     CHAR(25)
       )
RETURNS VARCHAR(60)
AS
BEGIN
       -- This is the variable that will hold the value to be returned
       DECLARE @FullName VARCHAR(60);

       -- SQL statements to concatenate the names in the proper order
       SELECT @FullName = RTRIM(@LastName) + ',  ' + RTRIM(@FirstName);

       -- Return the concatentate name
       RETURN @FullName;
END;
```

User-Defined Functions III
The *NameConcatenation* Function

- Using the NameConcatenation function:

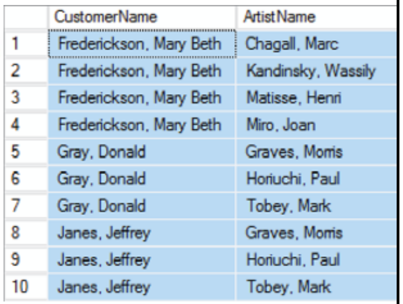
```
/* *** SQL-Query-CH07-02 *** */
SELECT      dbo.NameConcatenation(FirstName, LastName) AS CustomerName,
            AreaCode, PhoneNumber, EmailAddress
FROM        CUSTOMER
ORDER BY    CustomerName;
```



	CustomerName	AreaCode	PhoneNumber	EmailAddress
1	Frederickson, Mary Beth	303	513-8822	MaryBeth.Frederickson@somewhere.com
2	Gray, Donald	707	568-4839	Donald.Gray@somewhere.com
3	Janes, Jeffrey	425	543-2345	Jeffrey.Janes@somewhere.com
4	Johnson, Lynda	202	438-5498	NULL
5	Smathers, Fred	206	876-9911	Fred.Smathers@somewhere.com
6	Smith, David	970	654-9876	David.Smith@somewhere.com
7	Twilight, Tiffany	360	765-5566	Tiffany.Twilight@somewhere.com
8	Warning, Selma	604	988-0512	Selma.Warning@somewhere.com
9	Wilkens, Chris	360	876-8822	Chris.Wilkens@somewhere.com
10	Wu, Susan	404	653-3465	Susan.Wu@somewhere.com

User-Defined Functions IV
The *NameConcatenation* Function

```
/* *** SQL-Query-CH07-04 *** */
SELECT      dbo.NameConcatenation(C.FirstName, C.LastName) AS CustomerName,
            dbo.NameConcatenation(A.FirstName, A.LastName) AS ArtistName
FROM        CUSTOMER AS C JOIN CUSTOMER_ARTIST_INT AS CAI
    ON      C.CustomerID = CAI.CustomerID
            JOIN      ARTIST AS A
              ON      CAI.ArtistID = A.ArtistID
ORDER BY    CustomerName, ArtistName;
```



	CustomerName	ArtistName
1	Frederickson, Mary Beth	Chagall, Marc
2	Frederickson, Mary Beth	Kandinsky, Wassily
3	Frederickson, Mary Beth	Matisse, Henri
4	Frederickson, Mary Beth	Miro, Joan
5	Gray, Donald	Graves, Morris
6	Gray, Donald	Horiuchi, Paul
7	Gray, Donald	Tobey, Mark
8	Janes, Jeffrey	Graves, Morris
9	Janes, Jeffrey	Horiuchi, Paul
10	Janes, Jeffrey	Tobey, Mark

Triggers I

- A trigger is a stored program that is executed by the DBMS whenever a specified event occurs on a specified table or view.
- Three trigger types:
  BEFORE, INSTEAD OF, and AFTER
  - Each type can be declared for Insert, Update, and Delete.
  - Resulting in a total of nine trigger types.
- Oracle supports all nine trigger types.
- SQL Server supports six trigger types (INSTEAD OF and AFTER).
- MySQL supports six trigger types (BEFORE and AFTER).

Triggers II

Trigger Type / DML Action	BEFORE	INSTEAD OF	AFTER
INSERT	Oracle Database MySQL	Oracle Database SQL Server	Oracle Database SQL Server MySQL
UPDATE	Oracle Database MySQL	Oracle Database SQL Server	Oracle Database SQL Server MySQL
DELETE	Oracle Database MySQL	Oracle Database SQL Server	Oracle Database SQL Server MySQL

Firing Triggers

- When a trigger is fired, the DBMS supplies:
  - Old and new values for the update
  - New values for inserts
  - Old values for deletions
- The way the values are supplied depends on the DBMS product.
- Trigger applications include:
  - Providing default values
  - Enforcing data constraints
  - Updating views
  - Performing referential integrity actions

```
/* *** EXAMPLE CODE - DO NOT RUN ***                                          */

CREATE TRIGGER TRANS_AskingPriceInitialValue
       AFTER INSERT ON TRANS

DECLARE
       rowCount         Int;
       sumNetProfit     Numeric(10,2);
       avgNetProfit     Numeric(10,2);
BEGIN
       /* First find if work has been here before                             */

       SELECT    Count(*) INTO rowCount
       FROM      TRANS AS T
       WHERE     new:WorkID = T.WorkID;

       IF (rowCount = 1)
       THEN
          /* This is first time work has been in gallery                      */

          new:AskingPrice = 2 * new:AcquisitionPrice;

       ELSE
          IF rowCount > 1
          THEN
             /* Work has been here before                                     */

             SELECT    SUM(NetProfit) into sumNetProfit
             FROM      ArtistWorkNetView AWNV
             WHERE     AWNV.WorkID = new.WorkID
             GROUP BY  AWNV.WorkID;

             avgNetProfit = sumNetProfit / (rowCount - 1);

             /* Now choose larger value for the new AskingPrice               */

             IF ((new:AcquisitionPrice + avgNetProfit)
                   > (2 * new:AcquisitionPrice))
             THEN
                new:AskingPrice = (new:AcquisitionPrice + avgNetProfit);
             ELSE
                new:AskingPrice = (2 * new:AcquisitionPrice);
             END IF;
          ELSE
             /* Error, rowCount cannot be less than 1                         */
             /* Do something!                                                 */
          END IF;
       END IF;
END;
```

```
/* *** EXAMPLE CODE - DO NOT RUN ***                                   */

CREATE TRIGGER CustomerInterestView_UpdateCustomerLastName
       INSTEAD OF UPDATE ON CustomerInterestView

DECLARE

       rowCount         Int;

BEGIN

       SELECT    COUNT(*) into rowCount
       FROM      CUSTOMER
       WHERE     CUSTOMER.LastName = old:LastName

       IF  (rowCount = 1)
       THEN

          /* If get here, then only one customer has this last name.     */
          /* Make the name change.                                       */

          UPDATE    CUSTOMER
          SET       CUSTOMER.LastName = new:LastName
          WHERE     CUSTOMER.LastName = old:LastName;

       ELSE

          IF (rowCount > 1 )
          THEN

             /* Send a message to the user saying cannot update because  */
             /* there are too many customers with this last name.        */

          ELSE
             /* Error, if rowcount <= 0 there is an error!               */
             /* Do something!                                            */
          END IF;
       END IF;
END;
```

```
/* *** EXAMPLE CODE - DO NOT RUN ***                                   */

CREATE TRIGGER EMPLOYEE_DeleteCheck
       INSTEAD OF DELETE ON DeleteEmployeeView

DECLARE

       rowCount          Int;

BEGIN

       /*  First determine if this is the last employee in the department */

       SELECT    Count(*) into rowCount
       FROM      EMPLOYEE
       WHERE     EMPLOYEE.EmployeeNumber = old:EmployeeNumber;

       IF (rowCount > 1)
       THEN

          /* Not last employee, allow deletion                         */

          DELETE    EMPLOYEE
          WHERE     EMPLOYEE.EmployeeNumber = old:EmployeeNumber;

       ELSE

          /* Send a message to user saying that the last employee      */
          /* in a department cannot be deleted.                        */

       END IF;

END;
```

```
/* *** EXAMPLE CODE - DO NOT RUN ***                                          */

CREATE TRIGGER EMPLOYEE_DEPARTMENT_DeleteCheck
       INSTEAD OF DELETE ON DeleteEmployeeDepartmentView
DECLARE
       rowCount         Int;
BEGIN
       /*  First determine if this is the last employee in the department     */

       SELECT    Count(*) into rowCount
       FROM      EMPLOYEE
       WHERE     EMPLOYEE.EmployeeNumber = old:EmployeeNumber;

       /* Delete Employee row regardless of whether Department is deleted     */

       DELETE    EMPLOYEE
       WHERE     EMPLOYEE.EmployeeNumber = old:EmployeeNumber;

       IF (rowCount = 1)
       THEN

          /* Last employee in Department, delete Department                   */

          DELETE    DEPARTMENT
          WHERE     DEPARTMENT.DepartmentName = old:DepartmentName;

       END IF;

END;
```

Stored Procedures

- A **stored procedure** is a program that is stored within the database and is compiled when used.
  – In Oracle, it can be written in PL/SQL or Java.
  – In SQL Server, it can be written in TRANSACT-SQL.
- Stored procedures can receive input parameters and they can return results.
- Stored procedures can be called from:
  – Programs written in standard languages, e.g., Java, C#.
  – Scripting languages, e.g., JavaScript, VBScript.
  – SQL command prompt, e.g., SQL Plus, Query Analyzer.