

# Chapter 8: Advanced SQL

***Modern Database Management***  
***8<sup>th</sup> Edition***

***Jeffrey A. Hoffer, Mary B. Prescott,  
Fred R. McFadden***

# Objectives

- Definition of terms
- Write multiple table SQL queries
- Define and use three types of joins
- Write correlated and noncorrelated subqueries
- Establish referential integrity in SQL
- Understand triggers and stored procedures
- Discuss SQL:1999 standard and its extension of SQL-92

# Processing Multiple Tables

- The power of RDBMS is realized when working with multiple tables
- Two tables can be joined when each contains a column that shares a common domain
- The result of a join operation is a single table
- Selected columns from all tables are included
- Each row returned contains data from rows of input tables where values for the common columns match
- Rule of thumb: there should be one condition within the WHERE clause for each pair of tables being joined
- One condition for joining 2 tables, two conditions for joining 3 tables, and so forth.
- Two approaches are used for processing multiple tables:

Join & Subquery

# Processing Multiple Tables–Joins

- **Join** – a relational operation that causes two or more tables with a common domain to be combined into a single table or view
- **Equi-join** – a join in which the joining condition is based on equality between values in the common columns;  
*common columns appear redundantly in the result table*
- **Natural join** – an equi-join in which one of the duplicate columns is eliminated in the result table (no duplications of common column)
- **Outer join** – a join in which rows that do not have matching values in common columns are nonetheless included in the result table (as opposed to *inner join*, in which rows must have matching values in order to appear in the result table)
- **Union join** – includes *all columns* from *each table* in the join, and an instance for each row of each table

The **common columns** in **joined** tables are usually the **primary key** of the **dominant table** and the **foreign key** of the **dependent table** in 1:M relationships

# The following slides create tables for this enterprise data model

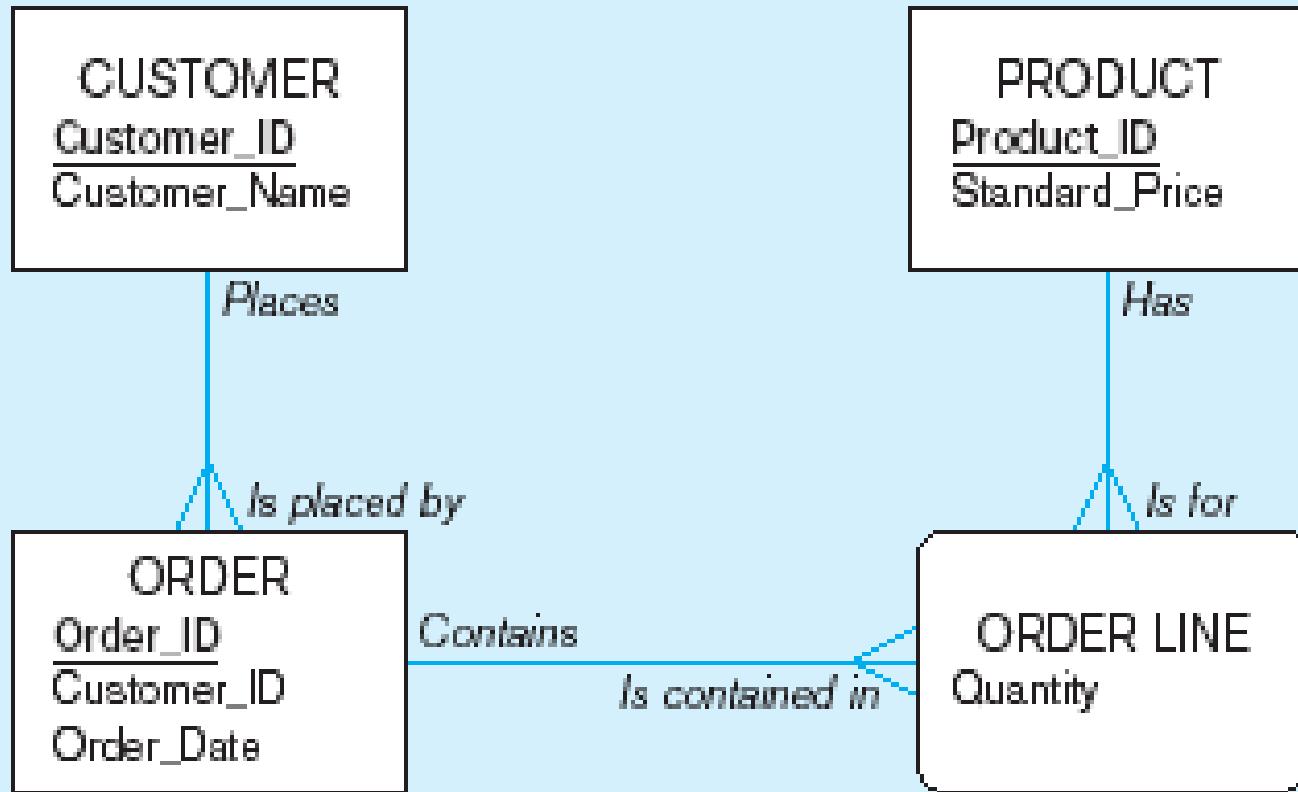


Figure 8-1 Pine Valley Furniture Company Customer and Order tables with pointers from customers to their orders

The screenshot shows two Microsoft Access tables side-by-side:

**ORDER\_t : Table**

	Order_ID	Order_Date	Customer_ID
▶	1001	10/21/2006	1←
▶	1002	10/21/2006	8←
▶	1003	10/22/2006	15←
▶	1004	10/22/2006	5←
▶	1005	10/24/2006	3←
▶	1006	10/24/2006	2←
▶	1007	10/27/2006	11←
▶	1008	10/30/2006	12←
▶	1009	11/5/2006	4←
▶	1010	11/5/2006	1←
*	0		0

Record: [navigation buttons] 1 [navigation buttons] of 10

**CUSTOMER\_t : Table**

	Customer_ID	Customer_Name	Customer_Address
▶	1	Contemporary Casuals	1355 S Hines Blvd
▶	2	Value Furniture	15145 S.W. 17th St.
▶	3	Home Furnishings	1900 Allard Ave.
▶	4	Eastern Furniture	1925 Beltline Rd.
▶	5	Impressions	5585 Westcott Ct.
▶	6	Furniture Gallery	325 Flatiron Dr.
▶	7	Period Furniture	394 Rainbow Dr.
▶	8	California Classics	816 Peach Rd.
▶	9	M and H Casual Furniture	3709 First Street
▶	10	Seminole Interiors	2400 Rocky Point Dr.
▶	11	American Euro Lifestyles	2424 Missouri Ave N.
▶	12	Battle Creek Furniture	345 Capitol Ave. SW
▶	13	Heritage Furnishings	66789 College Ave.
▶	14	Kaneohe Homes	112 Kiowai St.
▶	15	Mountain Scenes	4132 Main Street
▶	(AutoNumber)		

Record: [navigation buttons] 16 [navigation buttons] of 16

Unique number to identify customer

These tables are used in queries that follow

# Natural Join Example

## Microsoft syntax

- For each customer who placed an order, what is the customer's name and order number?

Join involves multiple tables in FROM clause

```
SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME, ORDER_ID  
FROM CUSTOMER_T NATURAL JOIN ORDER_T ON
```

```
CUSTOMER_T.CUSTOMER_ID = ORDER_T.CUSTOMER_ID;
```

ON clause performs the equality check for common columns of the two tables

Note: from Fig. 1, you see that only 10 Customers have links with orders.

→ Only 10 rows will be returned from this INNER join.

# Example

- **Query:** what are the names of all customers who have placed orders?
- **ORACLE SQL:**
  - **SELECT** Customer\_T.customer\_ID, customer\_name, order\_ID  
**FROM** Customer\_T, Order\_T  
**WHERE** Customer\_T.customer\_ID =  
Order\_T.customer\_ID;
- **Microsoft Access SQL:**
  - **SELECT** Customer\_T.customer\_ID,  
Customer\_T.customer\_Name, Order\_T.order\_ID  
**FROM** Customer\_T **INNER JOIN** Order\_T **ON**  
Customer\_T.customer\_ID = Order\_T.customer\_ID;

# Outer Join Example

- List the customer name, ID number, and order number for all customers. **Include customer information even for customers that do have an order**
- **(Microsoft Syntax)**

```
SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME, ORDER_ID  
FROM CUSTOMER_T LEFT OUTER JOIN ORDER_T  
ON CUSTOMER_T.CUSTOMER_ID = ORDER_T.CUSTOMER_ID;
```

LEFT OUTER JOIN syntax with ON causes customer data to appear even if there is no corresponding order data

**(ORACLE Syntax)**

```
SELECT CUSTOMER_T.CUSTOMER_ID,  
CUSTOMER_NAME, ORDER_ID  
FROM CUSTOMER_T LEFT OUTER JOIN ORDER_T  
WHERE CUSTOMER_T.CUSTOMER_ID =  
ORDER_T.CUSTOMER_ID;
```

Unlike INNER join, this will include customer rows with no matching order rows

## Result:

CUSTOMER_ID	CUSTOMER_NAME	ORDER_ID
1	Contemporary Casuals	1001
1	Contemporary Casuals	1010
2	Value Furniture	1006
3	Home Furnishings	1005
4	Eastern Furniture	1009
5	Impressions	1004
6	Furniture Gallery	
7	Period Furnishings	
8	California Classics	1002
9	M & H Casual Furniture	
10	Seminole Interiors	
11	American Euro Lifestyles	1007
12	Battle Creek Furniture	1008
13	Heritage Furnishings	
14	Kaneohe Homes	
15	Mountain Scenes	1003

## Results

Unlike  
INNER  
join, this  
will include  
customer  
rows with  
no  
matching  
order rows

16 rows selected.

# Multiple Table Join Example

- Assemble all information necessary to create an invoice for order number 1006

Four tables involved in this join

```
SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME,  
CUSTOMER_ADDRESS, CITY, STATE, POSTAL_CODE,  
ORDER_T.ORDER_ID, ORDER_DATE, QUANTITY,  
PRODUCT_DESCRIPTION, STANDARD_PRICE,  
(QUANTITY * UNIT_PRICE)
```

```
FROM CUSTOMER_T, ORDER_T, ORDER_LINE_T, PRODUCT_T
```

```
WHERE CUSTOMER_T.CUSTOMER_ID = ORDER_T.CUSTOMER_ID  
AND ORDER_T.ORDER_ID = ORDER_LINE_T.ORDER_ID  
AND ORDER_LINE_T.PRODUCT_ID = PRODUCT_T.PRODUCT_ID  
AND ORDER_T.ORDER_ID = 1006;
```

Each pair of tables requires an equality-check condition in the WHERE clause, matching primary keys against foreign keys

Figure 8-2 Results from a four-table join

From CUSTOMER\_T table

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_ADDRESS	CUSTOMER_CITY	CUSTOMER_ST	POSTAL_CODE
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094 7743
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094 7743
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094 7743

ORDER_ID	ORDER_DATE	QUANTITY	PRODUCT_NAME	STANDARD_PRICE	(QUANTITY* STANDARD_PRICE)
1006	24-OCT-06	1	Entertainment Center	650	650
1006	24-OCT-06	2	Writer's Desk	325	650
1006	24-OCT-06	2	Dining Table	800	1600

From ORDER\_T ,  
ORDER\_Line\_T tables

From  
PRODUCT\_T table

Computed  
Column

# Processing Multiple Tables Using Subqueries

- **Subquery** – placing an inner query (SELECT statement) inside an outer query
- Options:
  - In a **condition** of the WHERE clause
  - As a “**table**” of the FROM clause
  - Within the HAVING clause
- Subqueries can be:
  - Noncorrelated – executed **once** for the **entire outer query**
  - Correlated – executed **once** for **each row** returned by the **outer query**

# Correlated vs. Noncorrelated Subqueries

- Noncorrelated subqueries:
  - Do not depend on data from the outer query
  - Execute once for the entire outer query
- Correlated subqueries:
  - Make use of data from the outer query
  - Execute once for each row of the outer query
  - Can use the EXISTS operator

# Subquery Example

## Noncorrelated subqueries

- Show all customers who have placed an order

```
SELECT CUSTOMER_NAME FROM CUSTOMER_T  
WHERE CUSTOMER_ID IN  
    (SELECT DISTINCT CUSTOMER_ID FROM ORDER_T);
```

The **IN** operator will test to see if the CUSTOMER\_ID value of a **row** is included in the **list** returned from the subquery

Subquery is embedded in parentheses. In this case it returns a **list** that will be used in the WHERE clause of the outer query

## Figure 8-3a Processing a noncorrelated subquery

1. The subquery executes and returns the customer IDs from the ORDER\_T table
2. The outer query on the results of the subquery

```
SELECT CUSTOMER_NAME
      FROM CUSTOMER_T
     WHERE CUSTOMER_ID IN
          (SELECT DISTINCT CUSTOMER_ID
            FROM ORDER_T);
```

1. The subquery (shown in the box) is processed first and an intermediate results table created:

CUSTOMER_ID
1
8
15
5
3
2
11
12
4

9 rows selected.

No reference to data in outer query, so subquery executes once only

2. The outer query returns the requested customer information for each customer included in the intermediate results table:

CUSTOMER_NAME
Contemporary Casuals
Value Furniture
Home Furnishings
Eastern Furniture
Impressions
California Classics
American Euro Lifestyles
Battle Creek Furniture
Mountain Scenes

9 rows selected.

These are the only customers that have IDs in the ORDER\_T table

# Correlated Subquery Example

- Show **all orders** that include furniture finished in **natural ash**

The **EXISTS** operator will return a **TRUE** value if the subquery resulted in a **non-empty set**, otherwise it returns a **FALSE**

```
SELECT DISTINCT ORDER_ID FROM ORDER_LINE_T  
WHERE EXISTS  
(SELECT * FROM PRODUCT_T  
WHERE PRODUCT_ID = ORDER_LINE_T.PRODUCT_ID  
AND PRODUCT_FINISH = 'Natural ash');
```

The subquery is **testing** for a **value** that comes from the **outer** query

## Figure 8-3b Processing a correlated subquery

Note: only the **orders** that involve products with **Natural Ash** will be included in the final **results**

```
SELECT DISTINCT ORDER_ID FROM ORDER_LINE_T
WHERE EXISTS
  (SELECT *
   FROM PRODUCT_T
   WHERE PRODUCT_ID = ORDER_LINE_T.PRODUCT_ID
     AND PRODUCT_FINISH = 'Natural Ash');
```

Subquery refers to **outer-query** data, so executes **once** for each row of outer query

Order_ID	Product_ID	Ordered_Quantity
1001	1	1
1001	2	2
1001	4	1
1002	3	5
1003	5	1
1004	6	1
1005	4	4
1006	4	1
1007	1	1
1007	2	1
1008	3	1
1009	5	1
1009	4	4
1009	7	1
1010	8	1
1010	9	1

	Product_ID	Product_Description	Product_Finish	Standard_Price	Product_Line_Id
1	1	End-Table	Cherry	\$175.00	10001
2	2	Coffee Table	Natural Ash	\$200.00	20001
3	4	Computer Desk	Natural Ash	\$375.00	20001
4	4	Entertainment Center	Natural Maple	\$650.00	30001
5	5	Writer's Desk	Cherry	\$325.00	10001
6	6	8-Drawer Dresser	White Ash	\$750.00	20001
7	7	Dining Table	Natural Ash	\$800.00	20001
8	8	Computer Desk	Walnut	\$250.00	30001
*	(AutoNumber)			\$0.00	

1. The first order ID is selected from ORDER\_LINE\_T: ORDER\_ID = 1001.
2. The subquery is evaluated to see if any product in that order has a natural ash finish. Product 2 does, and is part of the order. EXISTS is valued as true and the order ID is added to the result table.
3. The next order ID is selected from ORDER\_LINE\_T: ORDER\_ID = 1002.
4. The subquery is evaluated to see if the product ordered has a natural ash finish. It does. EXISTS is valued as true and the order ID is added to the result table.
5. Processing continues through each order ID. Orders 1004, 1005, and 1010 are not included in the result table because they do not include any furniture with a natural ash finish. The final result table is shown in the text on page 303.

- When **EXISTS** and **NOT EXISTS** is used in a subquery, the select list of the subquery will usually select **all columns (Select \*)**
- It does not matter which columns are returned
- The purpose of the **subquery** is to **test** if any **rows fit the conditions**, not to **return values from particular columns** for **comparison** purposes in the **outer query**
- The **columns** that will be **displayed** are **determined** strictly by the **outer query**

# Another Solutions

## 1. Using subquery

```
SELECT DISTINCT ORDER_ID FROM ORDER_LINE_T  
WHERE PRODUCT_ID =  
  (SELECT PRODUCT_ID FROM PRODUCT_T  
    Where PRODUCT_FINISH = 'Natural ash');
```

## 2. Using Join

```
SELECT DISTINCT L.ORDER_ID  
FROM ORDER_LINE_T L, PRODUCT_T P  
WHERE L.PRODUCT_ID = P.PRODUCT_T  
  AND PRODUCT_FINISH = 'Natural ash';
```

# Another Subquery Example

- Show all products whose standard price is higher than the average price

Subquery forms the derived table used in the FROM clause of the outer query

One column of the subquery is an aggregate function that has an alias name. That alias can then be referred to in the outer query

```
SELECT PRODUCT_DESCRIPTION, STANDARD_PRICE, AVGPRICE
FROM
  (SELECT AVG(STANDARD_PRICE) AVGPRICE FROM PRODUCT_T),
PRODUCT_T
WHERE STANDARD_PRICE > AVGPRICE;
```

# Combining (Union) Queries

- Combine the output (union of multiple queries) together into a single result table

subquery should  
be in the from  
not to be repeated

```
SELECT C1.CUSTOMER_ID,CUSTOMER_NAME,ORDERED_QUANTITY,  
      'Largest Quantity' QUANTITY  
     FROM CUSTOMER_T C1,ORDER_T O1, ORDER_LINE_T Q1  
      WHERE C1.CUSTOMER_ID =O1.CUSTOMER_ID  
        AND O1.ORDER_ID =Q1.ORDER_ID  
        AND ORDERED_QUANTITY =  
              (SELECT MAX(ORDERED_QUANTITY)  
                 FROM ORDER_LINE_T)
```

First query

Combine → UNION

```
SELECT C1.CUSTOMER_ID,CUSTOMER_NAME,ORDERED_QUANTITY,  
      'Smallest Quantity'  
     FROM CUSTOMER_T C1,ORDER_T O1, ORDER_LINE_T Q1  
      WHERE C1.CUSTOMER_ID =O1.CUSTOMER_ID  
        AND O1.ORDER_ID =Q1.ORDER_ID  
        AND ORDERED_QUANTITY =  
              (SELECT MIN(ORDERED_QUANTITY)  
                 FROM ORDER_LINE_T)  
 ORDER BY ORDERED_QUANTITY;
```

Second query

The expression QUANTITY has been created in which the strings “Smallest Quantity” and “Largest Quantity” are inserted

The **result of the Union Query:**

CUSTOMER_id	CUSTOMER_Name	Ordered_Quantity	Quantity
1	Contemporary	1	Smallest Qty
2	Value Furniture	1	Smallest Qty
1	Contemporary	10	Largest Qty

# Tables

- COURSE (CRS#, CTITLE, C#HRS, C#HRSW)
- PREREQ (SUPCRS#, SUBCRS#, CRSCAT)
- OFFER (CRS#, OFF#, EMP#, LOCATION)
- TRAINER (EMP#, FNAME, LNAME, DOB, TEL#, JOB, SALARY, RAT)

# المجموعات Grouping

## دمج نتائج استفسارات متعددة UNION

مثال ٣: أرقام البرامج التدريبية التي ساعاتها الإجمالية أكبر ٦٠ ساعة أو التي نفذها المدرس ١٥ (أو كلاهما). رتب النتائج تنازلياً برقم البرنامج.

```
SELECT CRS#
      FROM COURSE
     WHERE C#HRS > 60
UNION
SELECT CRS#
      FROM OFFER
     WHERE EMP# = 15
ORDER BY 1 DESC ;
```

# المجموعات Grouping

## دمج نتائج استفسارات متعددة UNION

مثال ٤ : أرقام الدورات التدريبية التي ساعاتها الإجمالية أكبر من ٦٠ ساعة أو التينفذها المدرس ١٥ (أو كلاهما). رتب النتائج تنازلياً برقم الدورة مع إظهار الصفوف المتكررة .

```
SELECT CRS#
      FROM COURSE
     WHERE C#HRS > 60
UNION ALL
SELECT CRS#
      FROM OFFER
     WHERE EMP# = 15
    ORDER BY 1 DESC ;
```

# تعليمية SELECT Joining Tables ربط الجداول

## استرجاع البيانات من أكثر من جدول

- مثال ١ عرض قائمة بأسماء كل الدورات التدريبية وجميع بيانات مخطط تنفيذها :

```
SELECT CTITLE, OFFER.*  
      FROM COURSE, OFFER  
     WHERE COURSE.CRS# = OFFER.CRS# ;
```

- استخدام ممیز مختصر لجداول

```
SELECT CTITLE, O.*  
      FROM COURSE C, OFFER O  
     WHERE C.CRS# = O.CRS# ;
```

# SELECT تعليمة تابع ربط الجداول Joining Tables

## شروط أخرى في عبارة WHERE

- مثال ٢ عرض قائمة بأسماء كل البرامج التدريبية وجميع بيانات مخطط تنفيذها : للدورات المخططة في ”القاهرة“ :

```
SELECT CTITLE, O.*  
FROM COURSE C, OFFER O  
WHERE C.CRS# = O.CRS#  
AND O.LOCATION = 'Cairo' ;
```

# تعليمية SELECT تابع ربط الجداول Joining Tables

## شروط أخرى في عبارة WHERE

- مثال ٣ عرض قائمة بأرقام وأسماء كل الدورات التدريبية التي لم يحدد لها مكان تنفيذ (الشرط في جدول آخر)

```
SELECT C.CRS#, CTITLE
FROM COURSE C, OFFER O
WHERE C.CRS# = O.CRS#
AND O.LOCATION IS NULL ;
```

# تعليمية SELECT تابع ربط الجداول

## ربط أكثر من جدولين

مثال ٤ عرض قائمة بأسماء الدورات التدريبية المخطط تنفيذها في "القاهرة"  
وأرقام مدربيها

```
SELECT DISTINCT CTITLE, T.EMP#, O.LOCATION  
FROM COURSE C, OFFER O, TRAINER T  
WHERE C.CRS# = O.CRS#  
      AND O.EMP# = T.EMP#  
      AND O.LOCATION = 'Cairo' ;
```

# تعليمية SELECT الاستفسارات المتداخلة Subqueries

(الاستفسارات المتداخلة الفرعية)

```
SELECT ...  
FROM ...  
WHERE (SELECT ...  
       FROM ...  
       WHERE ...) ;
```

all arabic slides  
have sub-query  
in the where  
but that's not  
true it should be  
in the from to be  
executed once

# تعليمية SELECT الاستفسارات المتداخلة Subqueries

تابع الاستفسارات المتداخلة - إرجاع قيمة واحدة

مثال ٢ : قائمة بالموظفين الذين معدلات أدائهم أكبر من المتوسط العام للأداء

```
SELECT LNAME, FNAME  
FROM TRAINER  
WHERE RAT >  
(SELECT AVG (RAT)  
FROM TRAINER) ;
```

# تعليمية SELECT الاستفسارات المتداخلة Subqueries

الاستفسار الفرعى بعد أي من عوامل المقارنة

WHERE ColumnName

=  
>  
<  
>=  
=<  
<>  
IN

(subquery) ;

# تعليمية SELECT الاستفسارات المتداخلة Subqueries

## تابع الاستفسارات المتداخلة - إرجاع قيمة واحدة

مثال ٣: قائمة بأسماء الموظفين ومرتباتهم التي تزيد عن متوسط المرتبات

```
SELECT LNAME, FNAME, SALARY  
FROM TRAINER  
WHERE SALARY >  
(SELECT AVG (SALARY) FROM TRAINER) ;
```

move it here

# تعليمية SELECT الاستفسارات المتداخلة Subqueries

تابع الاستفسارات المتداخلة - إرجاع قيمة واحدة

مثال ٤ : أرقام الدورات التدريبية التي نفذها المدرس 'RASHED ALSALEH'

```
SELECT DISTINCT CRS#
  FROM OFFER
 WHERE EMP# = (SELECT EMP#
                  FROM TRAINER
                 WHERE LNAME = 'ALSALEH'
                   AND FNAME = 'RASHED') ;
```

# تعليمية SELECT الاستفسارات المتداخلة Subqueries

تابع الاستفسارات المتداخلة - إرجاع قيمة واحدة - ربط الاستفسارات الفرعية  
مثال ٦ : أسماء الموظفين الذين لهم أعلى وأقل معدل أداء

```
SELECT LNAME, FNAME, RAT
FROM TRAINER
WHERE RAT =
    (SELECT MAX (RAT)
     FROM TRAINER)
OR RAT =
    (SELECT MIN (RAT)
     FROM TRAINER) ;
```

# Ensuring Transaction Integrity

- **Transaction** = A discrete unit of work that must be completely processed **or** not processed at all
  - May involve multiple updates
  - If any update fails, then **all** other updates must be cancelled
- SQL commands for **transactions**
  - **BEGIN TRANSACTION/END TRANSACTION**
    - Marks **boundaries** of a transaction
  - **COMMIT**
    - Makes all updates **permanent**
  - **ROLLBACK**
    - Cancels updates since the last COMMIT

## Figure 8-5 An SQL Transaction sequence (in pseudocode)

```
BEGIN transaction  
  
    INSERT Order_ID, Order_date, Customer_ID into Order_t;  
  
    INSERT Order_ID, Product_ID, Quantity into Order_line_t;  
    INSERT Order_ID, Product_ID, Quantity into Order_line_t;  
    INSERT Order_ID, Product_ID, Quantity into Order_line_t;  
  
END transaction
```

Valid information inserted.

COMMIT work



All changes to data  
are made permanent.

Invalid Product\_ID entered



Transaction will be ABORTED.  
ROLLBACK all changes made to Order\_t



All changes made to Order\_t  
and Order\_line\_t are removed.  
Database state is just as it was  
before the transaction began.

# Routines and Triggers

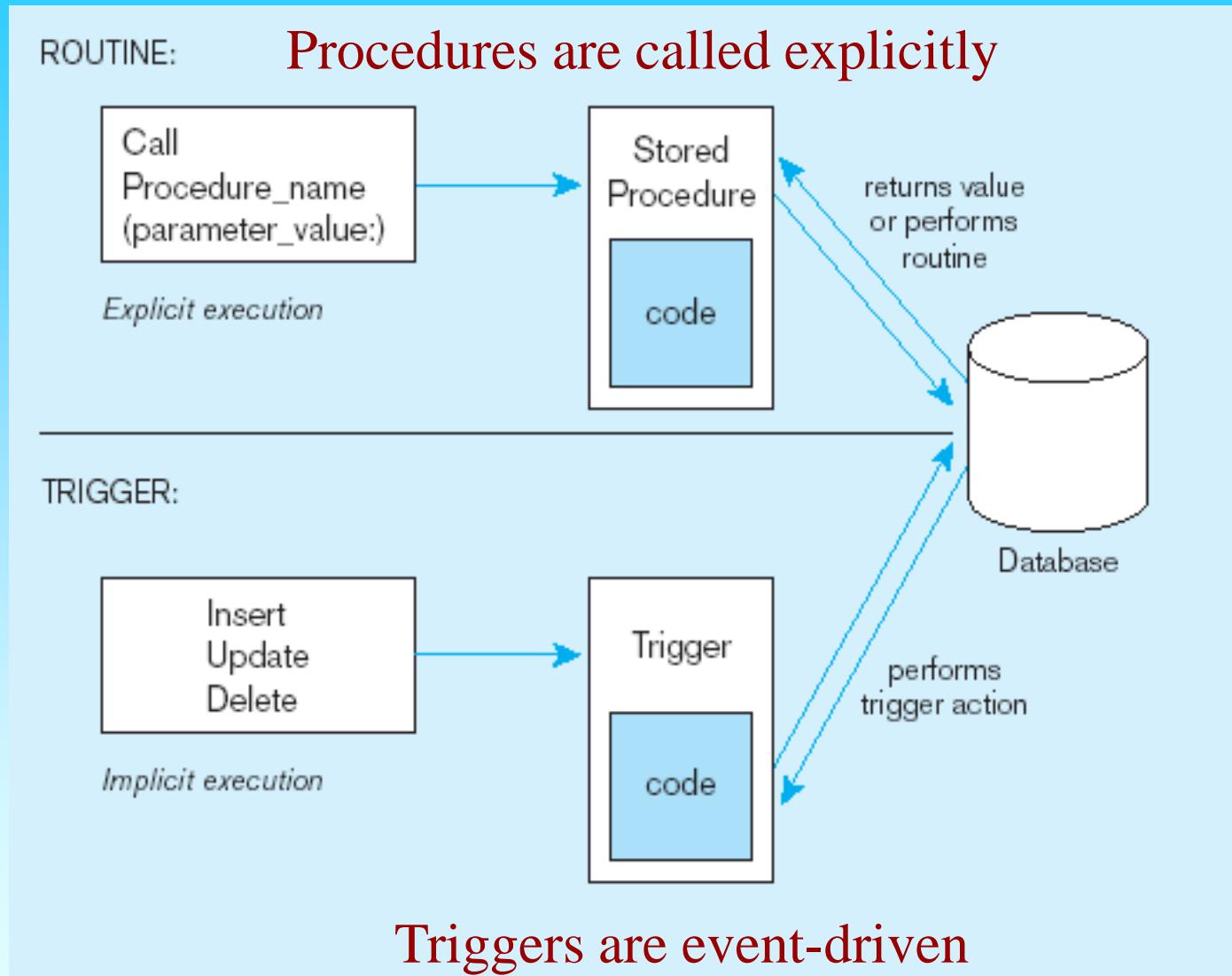
## ■ **Routines**

- Program modules that execute on demand
- **Functions** – routines that return values and take input parameters
- **Procedures** – routines that do not return values and can take input or output parameters

## ■ **Triggers**

- Routines that execute in response to a database **event** (INSERT, UPDATE, or DELETE)

Figure 8-6



Source: adapted from Mullins, 1995

Figure 8-7 Simplified trigger syntax, SQL:2003

```
CREATE TRIGGER trigger_name
  {BEFORE | AFTER | INSTEAD OF} {INSERT | DELETE | UPDATE} ON
  table_name
  [FOR EACH {ROW | STATEMENT}] [WHEN (search condition)]
  <triggered SQL statement here>;
```

Figure 8-8 Create routine syntax, SQL:2003

```
{CREATE PROCEDURE | CREATE FUNCTION} routine_name
([parameter {[,parameter] . . .}])
[RETURNS data_type result_cast] /* for functions only */
[LANGUAGE {ADA | C | COBOL | FORTRAN | MUMPS | PASCAL | PLI | SQL}]
[PARAMETER STYLE {SQL | GENERAL}]
[SPECIFIC specific_name]
[DETERMINISTIC | NOT DETERMINISTIC]
[NO SQL | CONTAINS SQL | READS SQL DATA | MODIFIES SQL DATA]
[RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT]
[DYNAMIC RESULT SETS unsigned_integer] /* for procedures only */
[STATIC DISPATCH] /* for functions only */
[NEW SAVEPOINT LEVEL | OLD SAVEPOINT LEVEL]
routine_body
```

# Routine Example (PL/SQL)

Create or Replace **Procedure** Product\_Line\_Sale  
AS Begin

```
update Product_T
    set sale_price = 0.9 * standard_price
    where standard_price >= 400;
update Product_T
    set sale_price = 0.85 * standard_price
    where standard_price < 400;
```

End;

SQL> Exec Product\_Line\_sale