CGS 2545: Database Concepts
Summer 2007
Chapter 8 – Advanced SQL

Instructor : Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 823-2790
http://www.cs.ucf.edu/courses/cgs2545/sum2007

School of Electrical Engineering and Computer Science
University of Central Florida

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:2003 enhancements and extensions

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
- 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 2-1 Segment from enterprise data model (Pine Valley Furniture Company)

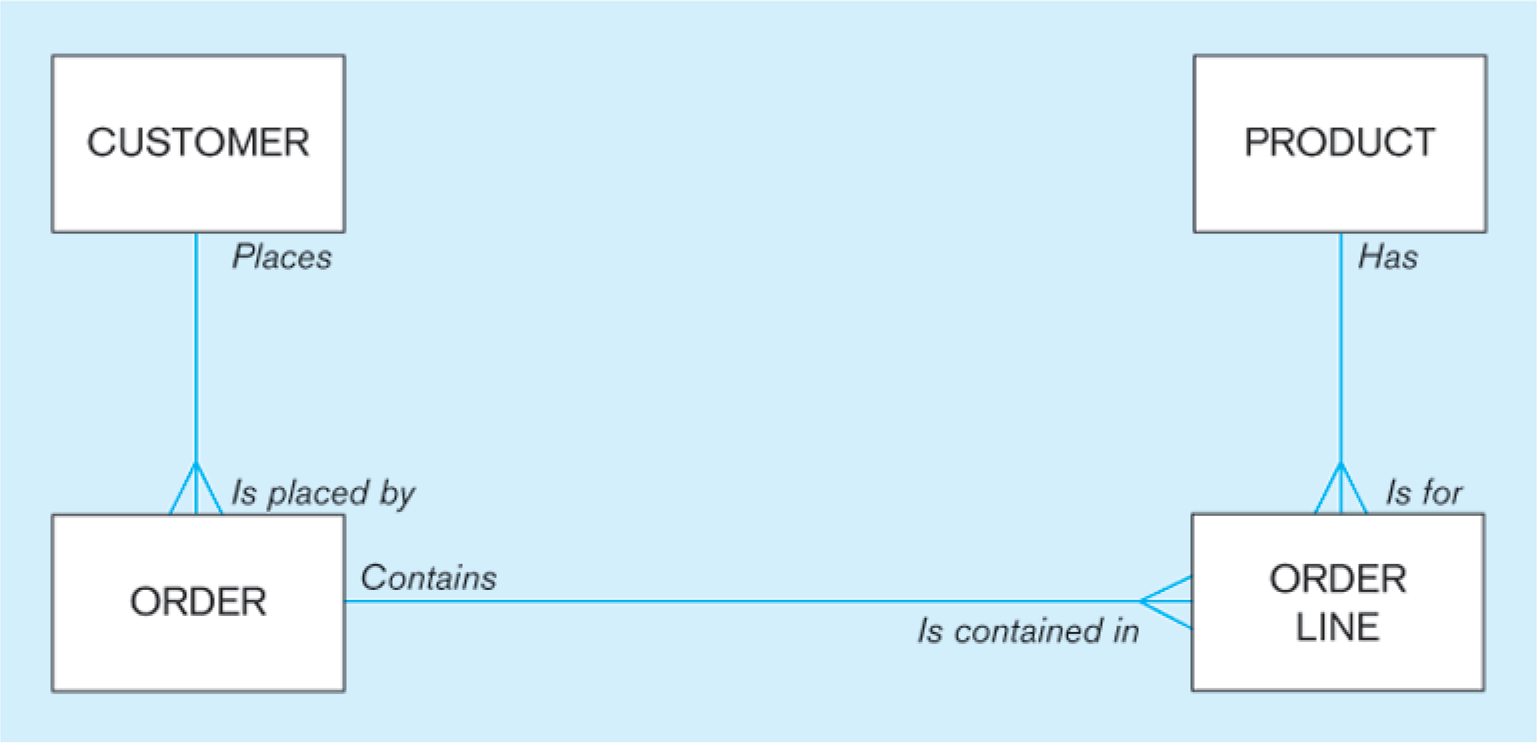
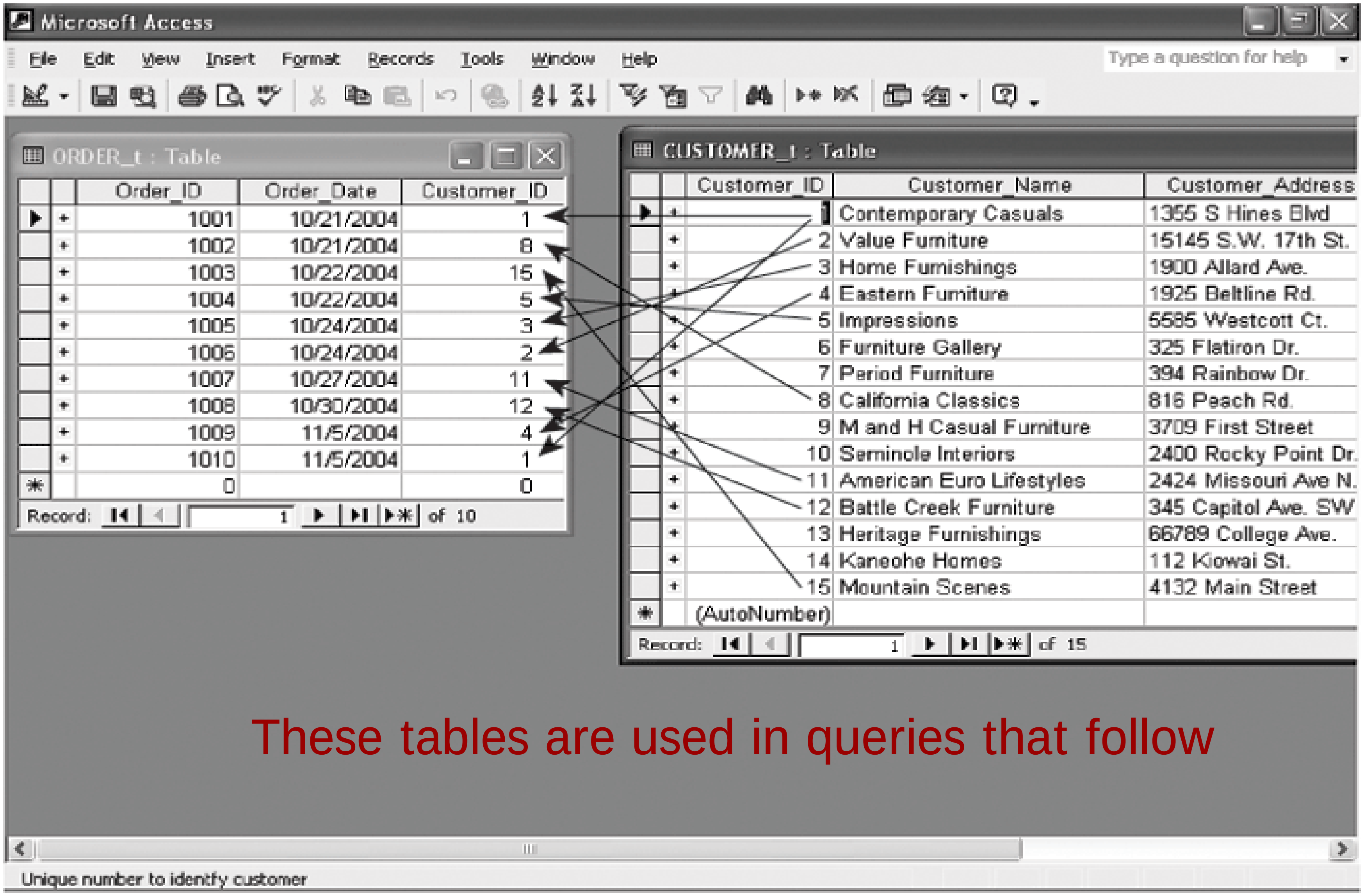


Figure 8-1 Pine Valley Furniture Company Customer and Order tables with pointers from customers to their orders



ORDER_t : Table

Order_ID	Order_Date	Customer_ID
1001	10/21/2004	1
1002	10/21/2004	8
1003	10/22/2004	15
1004	10/22/2004	5
1005	10/24/2004	3
1006	10/24/2004	2
1007	10/27/2004	11
1008	10/30/2004	12
1009	11/5/2004	4
1010	11/5/2004	1
0		0

Record: 1 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: 1 of 15

These tables are used in queries that follow

Unique number to identfy customer

Natural Join Example

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

Join involves multiple tables in FROM clause

```
SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME, ORDER_ID
FROM CUSTOMER_T, ORDER_T

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

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

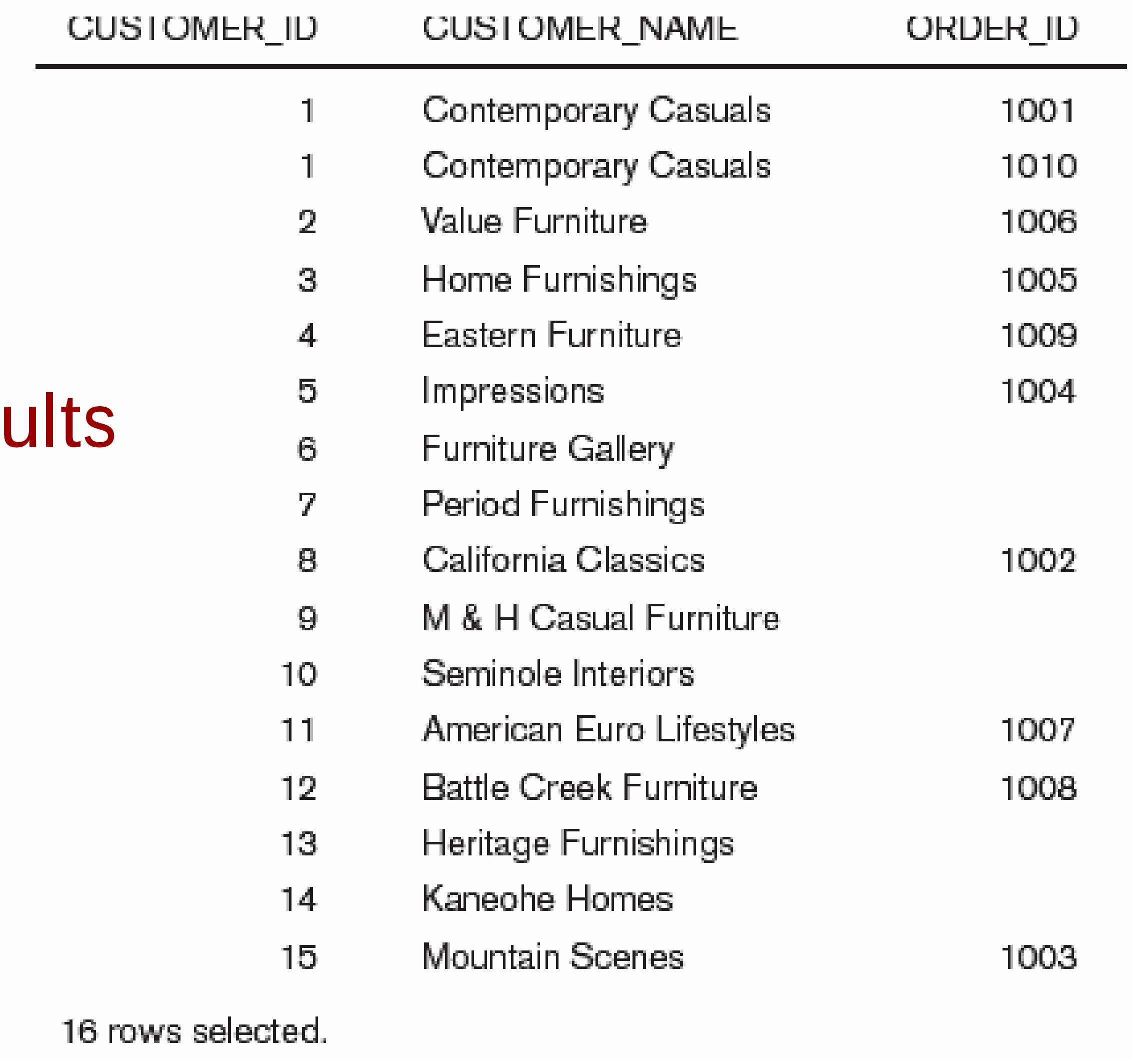
Outer Join Example (Microsoft Syntax)

- List the customer name, ID number, and order number for all customers. Include customer information even for customers that do not have an order.

```
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 keyword instead of WHERE → causes customer data to appear even if there is no corresponding order data

Results



```
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

16 rows selected.
```

Outer Join Example (Oracle Syntax)

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

```
SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME, ORDER_ID
FROM CUSTOMER_T,  ORDER_T
WHERE CUSTOMER_T.CUSTOMER_ID = ORDER_T.CUSTOMER_ID(+);
```

Outer join in Oracle uses regular join syntax, but adds (+) symbol to the side that will have the missing data

Multiple Table Join Example

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

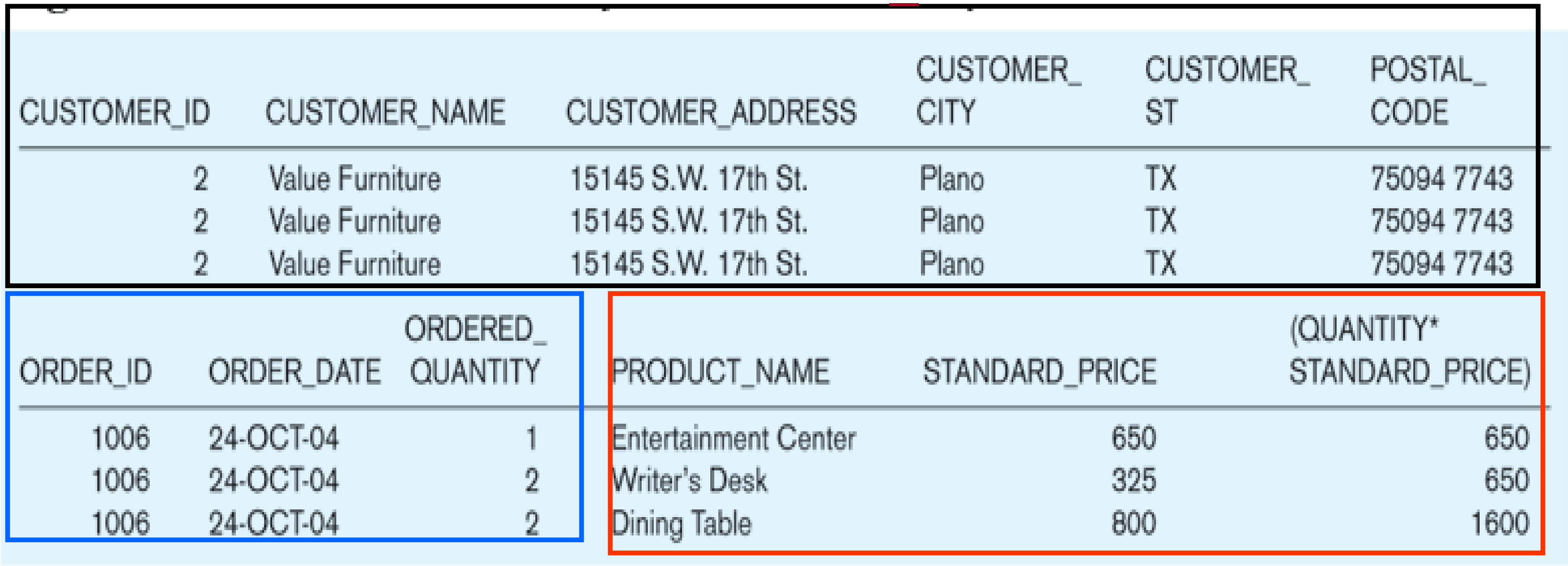
Four tables involved in this join

```
SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME,
   CUSTOMER_ADDRESS, CITY, SATE, POSTAL_CODE,
   ORDER_T.ORDER_ID, ORDER_DATE, QUANTITY,
   PRODUCT_NAME, UNIT_PRICE, (QUANTITY * UNIT_PRICE)
FROM CUSTOMER_T, ORDER_T, ORDER_LINE_T, PRODUCT_T
WHERE  CUSTOMER_T.CUSTOMER_ID =
   ORDER_LINE.CUSTOMER_ID      AND ORDER_T.ORDER_ID =
   ORDER_LINE_T.ORDER_ID
               AND ORDER_LINE_T.PRODUCT_ID =
      PRODUCT_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	ORDER_ID	ORDER_DATE	ORDERED_QUANTITY	PRODUCT_NAME	STANDARD_PRICE	(QUANTITY* STANDARD_PRICE)
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094 7743	1006	24-OCT-04	1	Entertainment Center	650	650
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094 7743	1006	24-OCT-04	2	Writer's Desk	325	650
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094 7743	1006	24-OCT-04	2	Dining Table	800	1600

From ORDER_T table

From PRODUCT_T table

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.

Subquery Example

- Show all customers who have placed an order.

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

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

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

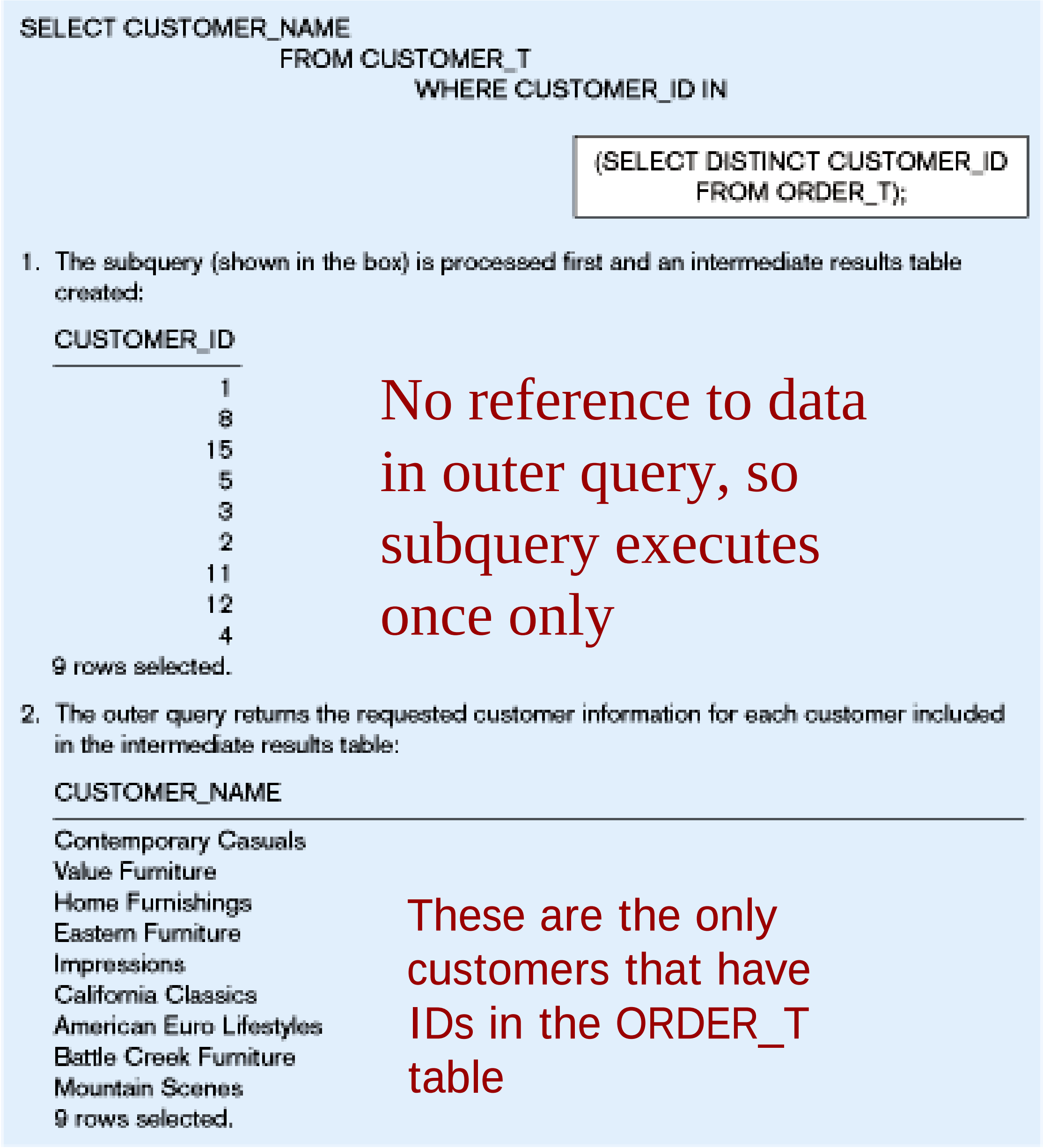
Correlated vs. Noncorrelated Subqueries

- 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.

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

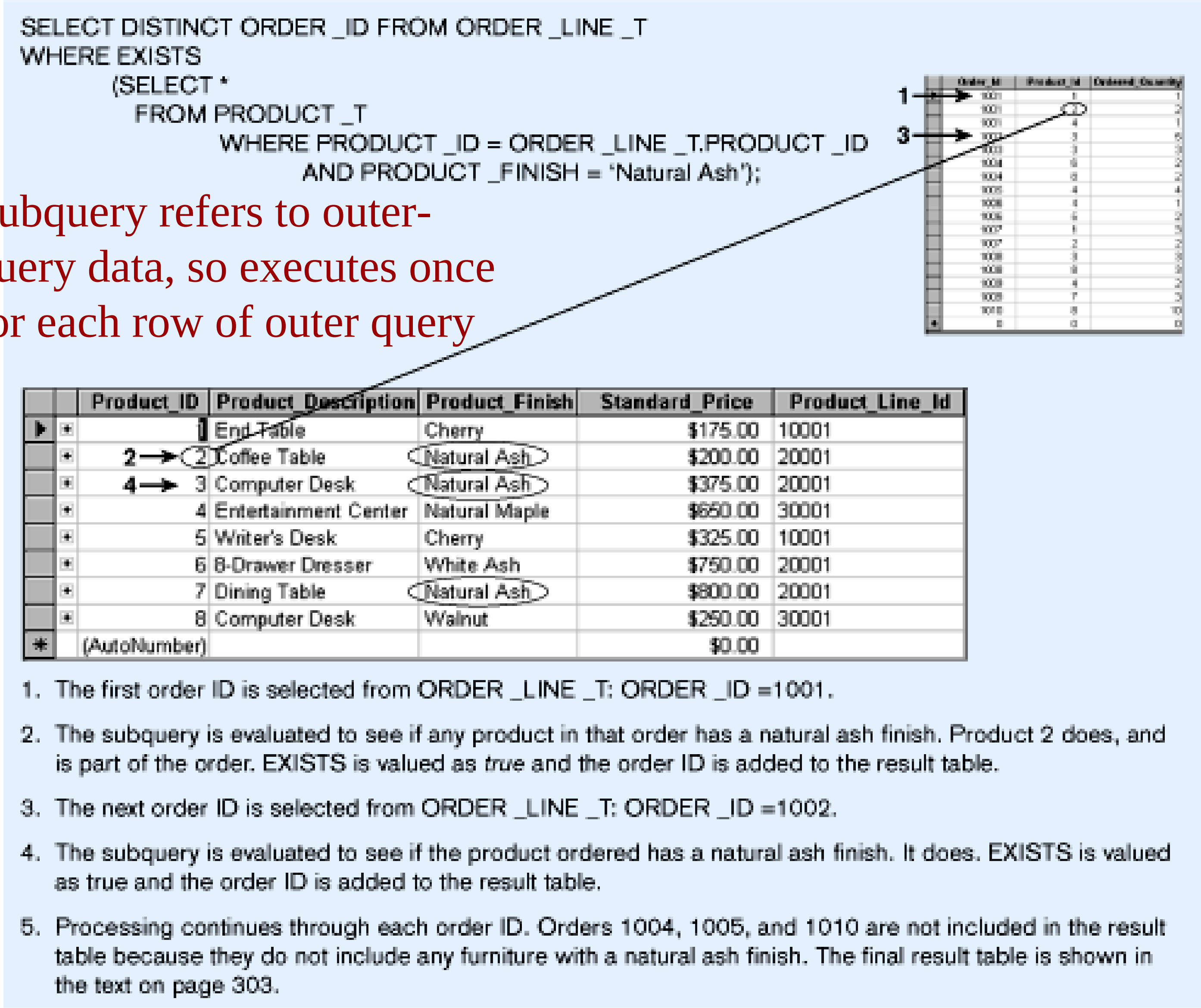
- Show all orders that include furniture finished in natural ash

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



```
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');
```

Order_Id	Product_Id	Ordered_Quantity
1001	1	1
1001	2	2
1001	4	1
1002	3	5
1003	3	3
1004	6	2
1004	8	2
1005	4	4
1006	4	1
1006	5	2
1007	1	3
1007	2	2
1008	3	3
1008	8	3
1009	4	2
1009	7	3
1010	8	10
0	0	0

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

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

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

Another Subquery Example

- Show all products whose price is higher than the average

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 > AVG_PRICE;
```

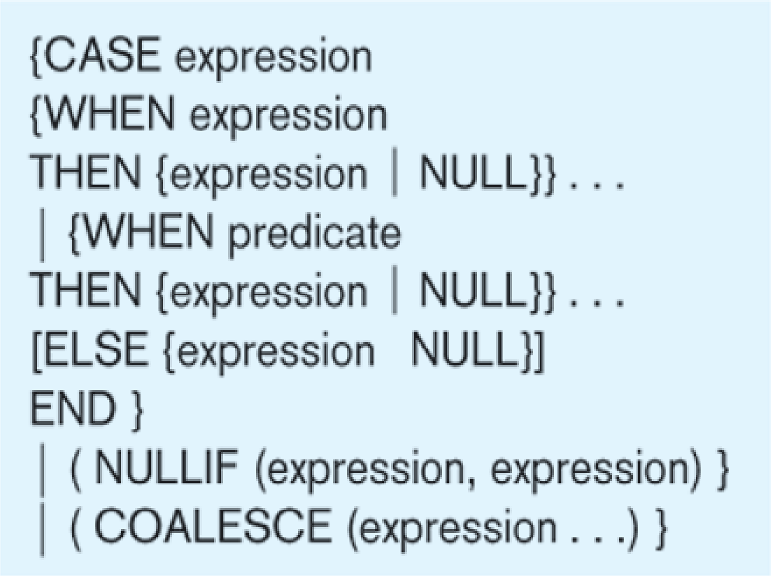
The WHERE clause normally cannot include aggregate functions, but because the aggregate is performed in the subquery its result can be used in the outer query’s WHERE clause

Conditional Expressions Using Case Syntax

This is available with newer versions of SQL, previously not part of the standard

Figure 8-4

CASE conditional syntax

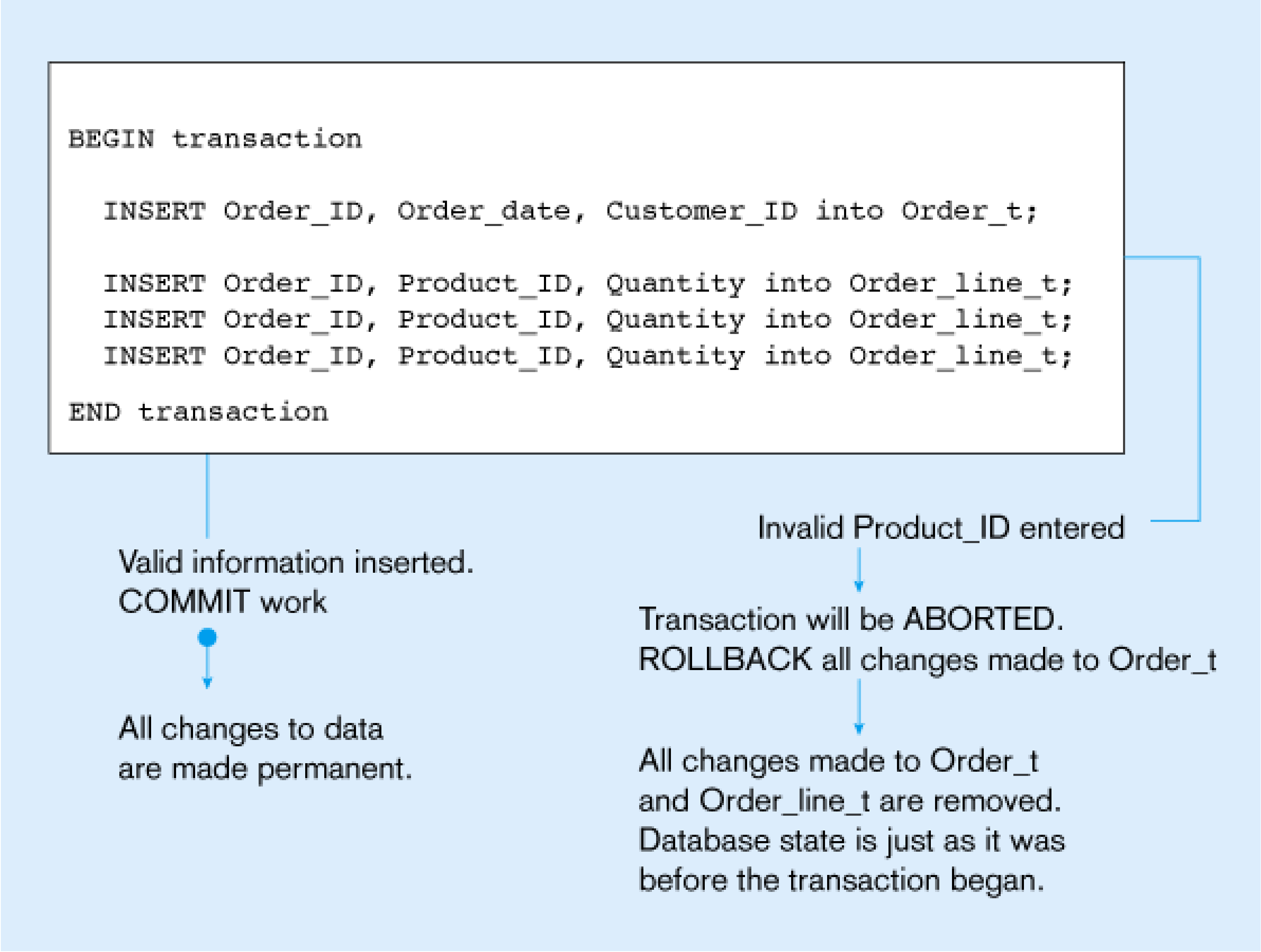


```
{CASE expression
{WHEN expression
THEN {expression | NULL}} . . .
 | {WHEN predicate
THEN {expression | NULL}} . . .
[ELSE {expression   NULL}]
END }
 | ( NULLIF (expression, expression) }
 | ( COALESCE (expression . . .) }
```

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.

Data Dictionary Facilities

- System tables that store metadata
- Users usually can view some of these tables
- Users are restricted from updating them
- Examples in Oracle 9i
  - DBA_TABLES – descriptions of tables
  - DBA_CONSTRAINTS – description of constraints
  - DBA_USERS – information about the users of the system
- Examples in Microsoft SQL Server
  - SYSCOLUMNS – table and column definitions
  - SYSDEPENDS – object dependencies based on foreign keys
  - SYSPERMISSIONS – access permissions granted to users

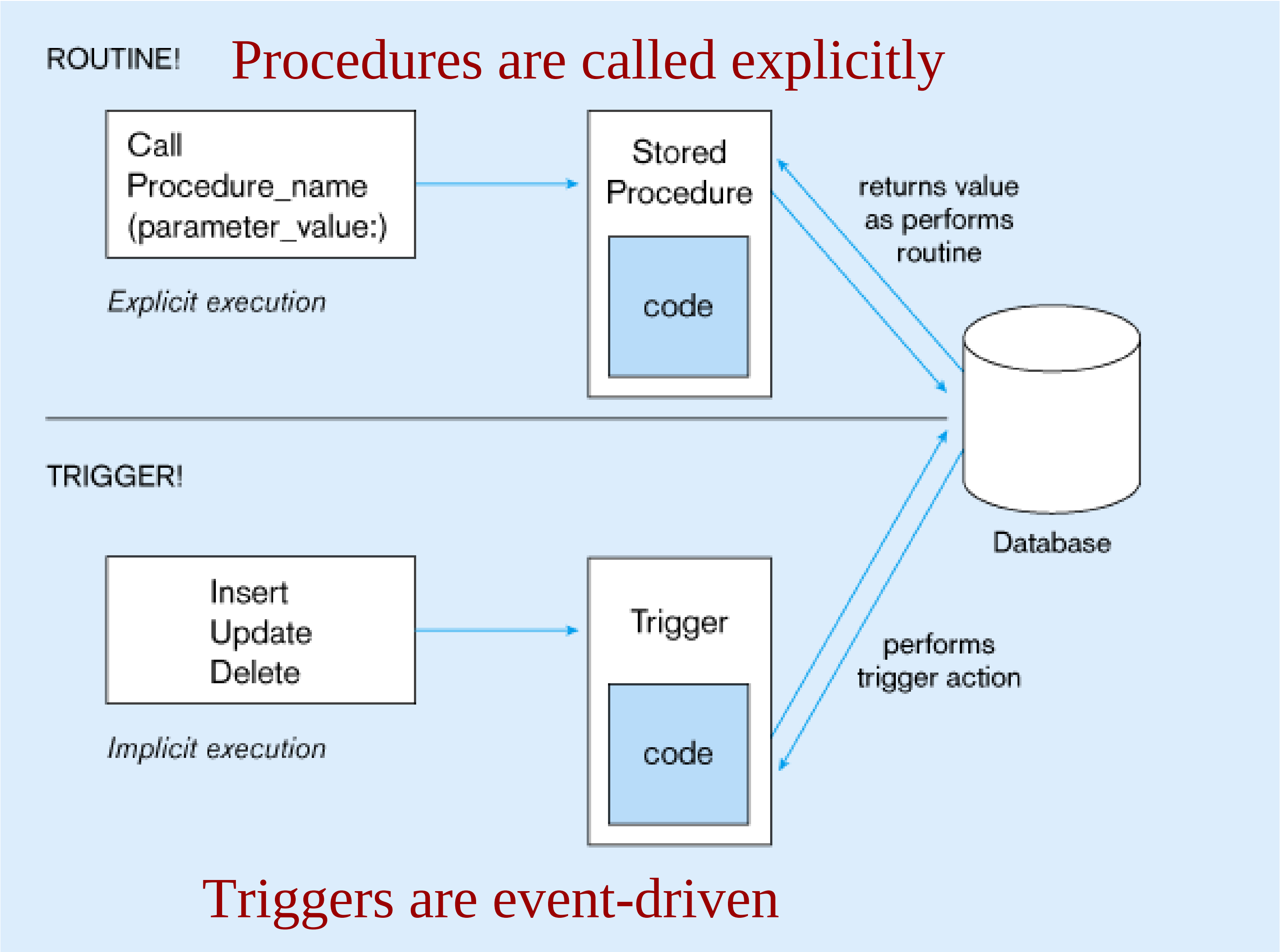
SQL:2003 Enhancements/Extensions

- User-defined data types (UDT)
  - Subclasses of standard types or an object type
- Analytical functions (for OLAP)
- Persistent Stored Modules (SQL/PSM)
  - Capability to create and drop code modules
  - New statements:
    - CASE, IF, LOOP, FOR, WHILE, etc.
    - Makes SQL into a procedural language
- Oracle has propriety version called PL/SQL, and Microsoft SQL Server has Transact/SQL

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: Triggers contrasted with stored procedures

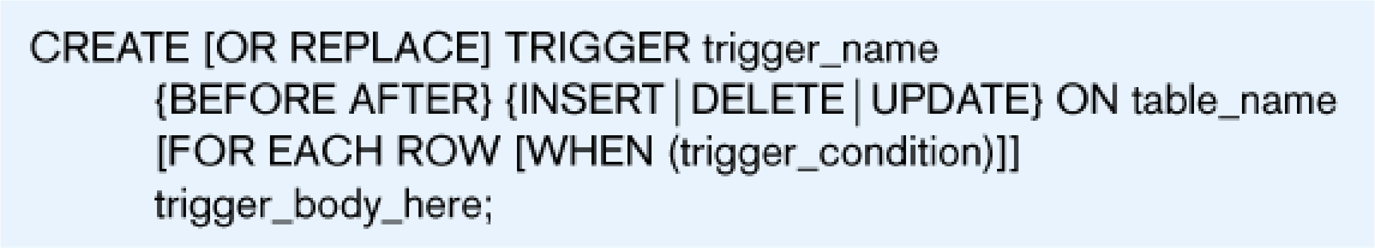


Procedures are called explicitly

Triggers are event-driven

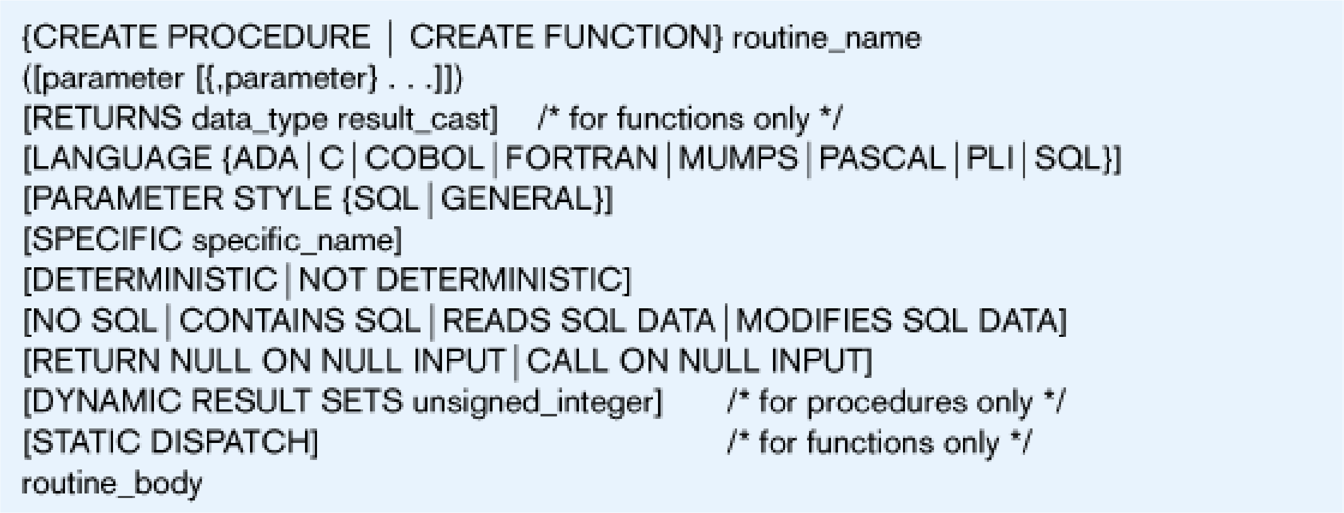
Source: adapted from Mullins, 1995.

Figure 8-7: Oracle PL/SQL trigger syntax



```
CREATE [OR REPLACE] TRIGGER trigger_name
        {BEFORE AFTER} {INSERT | DELETE | UPDATE} ON table_name
        [FOR EACH ROW [WHEN (trigger_condition)]]
        trigger_body_here;
```

Figure 8-8: SQL:2003 Create routine syntax



```
{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]
[RETURN NULL ON NULL INPUT | CALL ON NULL INPUT]
[DYNAMIC RESULT SETS unsigned_integer]      /* for procedures only */
[STATIC DISPATCH]                           /* for functions only */
routine_body
```

Embedded and Dynamic SQL

- Embedded SQL
  - Including hard-coded SQL statements in a program written in another language such as C or Java
- Dynamic SQL
  - Ability for an application program to generate SQL code on the fly, as the application is running