Chapter 8: Advanced SQL

Modern Database Management

9th Edition

Jeffrey A. Hoffer, Mary B. Prescott, Heikki Topi

Objectives

- Definition of terms
- Write single and multiple table SQL queries
- Define and use three types of joins
- Write noncorrelated and correlated subqueries
- Establish referential integrity in SQL
- Understand triggers and stored procedures
- Discuss SQL:2003 standard and its extension of SQL-92
- Understand SQL use for OLTP and OLAP
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.
Figure 8-2
Visualization of different join types with results returned in shaded area

- Natural Join: Darker area is result returned.
- Left Outer Join: All records returned from outer table. Matching records returned from joined table.
- Union Join: All records are returned.
The following slides create tables for this enterprise data model

CUSTOMER
Customer_ID
Customer_Name

PRODUCT
Product_ID
Standard_Price

ORDER
Order_ID
Customer_ID
Order_Date

ORDER LINE
Quantity

Places

Is placed by

Has

Contains

Is contained in

Is for
Figure 8-1 Pine Valley Furniture Company Customer and Order tables with pointers from customers to their orders
Natural Join Example

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

\[
\text{SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME, ORDER_ID FROM CUSTOMER_T NATURAL JOIN ORDER_T ON CUSTOMER_T.CUSTOMER_ID = ORDER_T.CUSTOMER_ID;}
\]

- Join involves multiple tables in FROM clause
- 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
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 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 causes customer data to appear even if there is no corresponding order data.**

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

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

16 rows selected.

<table>
<thead>
<tr>
<th>CUSTOMER_ID</th>
<th>CUSTOMER_NAME</th>
<th>ORDER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Contemporary Casuals</td>
<td>1001</td>
</tr>
<tr>
<td>1</td>
<td>Contemporary Casuals</td>
<td>1010</td>
</tr>
<tr>
<td>2</td>
<td>Value Furniture</td>
<td>1006</td>
</tr>
<tr>
<td>3</td>
<td>Home Furnishings</td>
<td>1005</td>
</tr>
<tr>
<td>4</td>
<td>Eastern Furniture</td>
<td>1009</td>
</tr>
<tr>
<td>5</td>
<td>Impressions</td>
<td>1004</td>
</tr>
<tr>
<td>6</td>
<td>Furniture Gallery</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Period Furniture</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>California Classics</td>
<td>1002</td>
</tr>
<tr>
<td>9</td>
<td>M &amp; H Casual Furniture</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Seminole Interiors</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>American Euro Lifestyles</td>
<td>1007</td>
</tr>
<tr>
<td>12</td>
<td>Battle Creek Furniture</td>
<td>1008</td>
</tr>
<tr>
<td>13</td>
<td>Heritage Furnishings</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Kaneohe Homes</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Mountain Scenes</td>
<td>1003</td>
</tr>
</tbody>
</table>
Multiple Table Join Example

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

```sql
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_LINE.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';
```

Four tables involved in this join

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

Chapter 8
Figure 8-4 Results from a four-table join

<table>
<thead>
<tr>
<th>CUSTOMER_ID</th>
<th>CUSTOMER NAME</th>
<th>CUSTOMER ADDRESS</th>
<th>CITY</th>
<th>ST</th>
<th>POSTAL_CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Value Furniture</td>
<td>15145 S.W. 17th St.</td>
<td>Plano</td>
<td>TX</td>
<td>75094 7743</td>
</tr>
<tr>
<td>2</td>
<td>Value Furniture</td>
<td>15145 S.W. 17th St.</td>
<td>Plano</td>
<td>TX</td>
<td>75094 7743</td>
</tr>
<tr>
<td>2</td>
<td>Value Furniture</td>
<td>15145 S.W. 17th St.</td>
<td>Plano</td>
<td>TX</td>
<td>75094 7743</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ORDER_ID</th>
<th>ORDER_DATE</th>
<th>QUANTITY</th>
<th>PRODUCT_NAME</th>
<th>STANDARD_PRICE</th>
<th>STANDARD_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>24-OCT-00</td>
<td>1</td>
<td>Entertainment Center</td>
<td>550</td>
<td>550</td>
</tr>
<tr>
<td>1006</td>
<td>24-OCT-06</td>
<td>2</td>
<td>Writer’s Desk</td>
<td>325</td>
<td>550</td>
</tr>
<tr>
<td>1006</td>
<td>24-OCT-06</td>
<td>2</td>
<td>Dining Table</td>
<td>800</td>
<td>1600</td>
</tr>
</tbody>
</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.

Result:

<table>
<thead>
<tr>
<th>CUSTOMER_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contemporary</td>
</tr>
<tr>
<td>Casuals</td>
</tr>
<tr>
<td>Value Furniture</td>
</tr>
<tr>
<td>Home Furniture</td>
</tr>
<tr>
<td>Eastern Furniture</td>
</tr>
<tr>
<td>Impressions</td>
</tr>
<tr>
<td>California Classics</td>
</tr>
<tr>
<td>American Euro Lifestyles</td>
</tr>
<tr>
<td>Battle Creek Furniture</td>
</tr>
<tr>
<td>Mountain Scenes</td>
</tr>
</tbody>
</table>

9 rows selected.
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 8-6a
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.

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

   These are the only customers that have IDs in the ORDER_T table.

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

<table>
<thead>
<tr>
<th>CUSTOMER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

9 rows selected.
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 8-6b  Processing a correlated subquery

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

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 334.
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

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

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.

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.

Chapter 8 © 2009 Pearson Education, Inc. Publishing as Prentice Hall
Union Queries

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

```sql
SELECT C1.CUSTOMER_ID, C1.CUSTOMER_NAME, ORDERED_QUANTITY,
QUANTITY AS 'Largest 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)
UNION

SELECT C1.CUSTOMER_ID, C1.CUSTOMER_NAME, ORDERED_QUANTITY,
QUANTITY AS '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;
```
Figure 8-7 Combining queries using UNION

```sql
SELECT C1.CUSTOMER_ID, C1.CUSTOMER_NAME, ORDERED_QUANTITY, QUANTITY AS 'Largest Quantity'
FROM CUSTOMER_T C1, ORDER_T O1, ORDER_LINE_T O1
WHERE C1.CUSTOMER_ID = O1.CUSTOMER_ID
AND O1.ORDER_ID = O1.ORDER_ID
AND ORDERED_QUANTITY =
  (SELECT MAX(ORDERED_QUANTITY)
   FROM ORDER_LINE_T)
1. In the above query, the subquery is processed first and an intermediate results table created. It contains the maximum quantity ordered from ORDER_LINE_T and has a value of 10.
2. Next the main query selects customer information for the customer or customers who ordered 10 of any item. Contemporary Casuals has ordered 10 of some unspecified item.

SELECT C1.CUSTOMER_ID, C1.CUSTOMER_NAME, ORDERED_QUANTITY, QUANTITY AS 'Smallest Quantity'
FROM CUSTOMER_T C1, ORDER_T O1, ORDER_LINE_T O1
WHERE C1.CUSTOMER_ID = O1.CUSTOMER_ID
AND O1.ORDER_ID = O1.ORDER_ID
AND ORDERED_QUANTITY =
  (SELECT MIN(ORDERED_QUANTITY)
   FROM ORDER_LINE_T)
ORDER BY ORDERED_QUANTITY;
1. In the second main query, the same process is followed but the result returned is for the minimum order quantity.
2. The results of the two queries are joined together using the UNION command.
3. The results are then ordered according to the value in ORDERED_QUANTITY. The default is ascending value, so the orders with the smallest quantity, 1, are listed first.
Conditional Expressions Using Case Syntax

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

Figure 8-8

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

- Be familiar with the data model (entities and relationships)
- Understand the desired results
- Know the attributes desired in result
- Identify the entities that contain desired attributes
- Review ERD
- Construct a WHERE for each link
- Fine tune with GROUP BY and HAVING clauses if needed
- Consider the effect on unusual data
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 8-9 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
- Some examples in Oracle 10g
  - DBA_TABLES—descriptions of tables
  - DBA_CONSTRAINTS—description of constraints
  - DBA_USERS—information about the users of the system
- Examples in Microsoft SQL Server 2000
  - SYSCOLUMNS—table and column definitions
  - SYSDEPENDS—object dependencies based on foreign keys
  - SYSPERMISSIONS—access permissions granted to users
SQL: 1999 and SQL: 200\textsuperscript{N} Enhancements/Extensions

- User-defined data types (UDT)
  - Subclasses of standard types or an object type
- Analytical functions (for OLAP)
  - CEILING, FLOOR, SQRT, RANK, DENSE_RANK
  - WINDOW—improved numerical analysis capabilities
- New Data Types
  - BIGINT, MULTISET (collection), XML
- CREATE TABLE LIKE—create a new table similar to an existing one
- MERGE
SQL: 1999 and SQL: 200\textsubscript{N}

Enhancements/Extensions (cont.)

- 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)
Triggers are event-driven

Procedures are called explicitly

Figure 8-10 Triggers contrasted with stored procedures

Source: adapted from Mullins, 1995.
Figure 8-11 Simplified trigger syntax, SQL:200n

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 8-12 Create routine syntax, SQL:200n

{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

Chapter 8   © 2009 Pearson Education, Inc.  Publishing as Prentice Hall
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
OLAP SQL

- Online Transaction Processing (OLTP)
  - Real-time processing of SQL transactions, characterized by fast data entry and retrieval in multi-user environments
  - Smaller size, less data sources, many users, simpler queries, more normalized, standard SQL

- Online Analytical Processing (OLAP)
  - Use of graphical tools providing users with multidimensional views of data for analysis purposes
  - Larger size, more data sources, fewer users, complex queries, less normalized, OLAP SQL and statistical packages (SPSS/SAS)