Processing Multiple Tables – Joins

• The ability to combine, or join, tables on common attributes is one of the most important advantages that relational databases have over other types of databases.

• A join is performed when data are retrieved from more than one table at a time.

• There are several different types of join operations as illustrated on the next page.

• In general, a join operation causes two or more tables with a common domain to be combined into a single table or view.
Processing Multiple Tables – Joins

• Theta-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. If all join conditions are equality, then the operation is known as an equi-join.

• 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.
**Theta-Join and Equijoin Operators**

Type: binary

Symbol/general form: \( r \bowtie_{\text{predicate}} s \)

Schema of result relation: concatenation of operand relations

Definition: \( r \bowtie_{\text{predicate}} s \equiv \sigma_{\text{predicate}}(r \times s) \)

Examples:

\[ r \bowtie_{\text{color='blue' AND size=3}} s \]

\[ r \bowtie_{\text{color='blue' AND size>3}} s \]

- The theta-join operation is a shorthand for a Cartesian product followed by a selection operation.

- The equijoin operation is a special case of the theta-join operation that occurs when all of the conditions in the predicate are equality conditions.

- Neither a theta-join nor an equijoin operation eliminates extraneous tuples by default. Therefore, the elimination of extraneous tuples must be handled explicitly via the predicate.
Theta-Join Operator Examples

\[ r = R \bowtie_{(R.B < S.F)} S \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td>no</td>
<td>7</td>
</tr>
<tr>
<td>c</td>
<td>f</td>
<td>yes</td>
<td>34</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
<td>no</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>r</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>f</td>
<td>yes</td>
<td>34</td>
</tr>
<tr>
<td>m</td>
<td>n</td>
<td>no</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>yes</td>
<td>1</td>
<td>b</td>
<td>r</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>yes</td>
<td>1</td>
<td>c</td>
<td>f</td>
<td>yes</td>
<td>34</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>yes</td>
<td>1</td>
<td>m</td>
<td>n</td>
<td>no</td>
<td>56</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td>no</td>
<td>7</td>
<td>b</td>
<td>r</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td>no</td>
<td>7</td>
<td>c</td>
<td>f</td>
<td>yes</td>
<td>34</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td>no</td>
<td>7</td>
<td>m</td>
<td>n</td>
<td>no</td>
<td>56</td>
</tr>
<tr>
<td>c</td>
<td>f</td>
<td>yes</td>
<td>34</td>
<td>b</td>
<td>r</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>f</td>
<td>yes</td>
<td>34</td>
<td>m</td>
<td>n</td>
<td>no</td>
<td>56</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
<td>no</td>
<td>6</td>
<td>b</td>
<td>r</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
<td>no</td>
<td>6</td>
<td>c</td>
<td>f</td>
<td>yes</td>
<td>34</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
<td>no</td>
<td>6</td>
<td>m</td>
<td>n</td>
<td>no</td>
<td>56</td>
</tr>
</tbody>
</table>
Natural Join Operator

Type: binary
Symbol/general form: \( r \ast s \)

Schema of result relation: concatenation of operand relations with only one occurrence of commonly named attributes

Definition: \( r \ast s \equiv r \bowtie (r.\text{common attributes} = s.\text{common attributes}) \) \( S \)

Examples: \( s \ast s \text{pj} \ast p \)

• The natural-join operation performs an equijoin over all attributes in the two operand relations which have the same attribute name.

• The degree of the result relation of a natural-join is sum of the degrees of the two operand relations less the number of attributes which are common to both operand relations. (In other words, one occurrence of each common attribute is eliminated from the result relation.)

• The natural join is probably the most common of all the forms of the join operation. It is extremely useful in the removal of extraneous tuples. Those attributes which are commonly named between the two operand relations are commonly referred to as the *join attributes*. 
# Natural Join Operator Examples

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>r</td>
<td>no</td>
<td>7</td>
</tr>
<tr>
<td>c</td>
<td>f</td>
<td>yes</td>
<td>34</td>
</tr>
<tr>
<td>a</td>
<td>m</td>
<td>no</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>M</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>yes</td>
<td>1</td>
<td>a</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>yes</td>
<td>1</td>
<td>f</td>
<td>yes</td>
<td>34</td>
</tr>
<tr>
<td>a</td>
<td>m</td>
<td>no</td>
<td>6</td>
<td>n</td>
<td>no</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>M</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>r</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>yes</td>
<td>34</td>
</tr>
<tr>
<td>m</td>
<td>n</td>
<td>no</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>f</td>
<td>no</td>
<td>31</td>
</tr>
<tr>
<td>b</td>
<td>r</td>
<td>yes</td>
<td>30</td>
</tr>
</tbody>
</table>
Outer Join Operator

Type: binary
Symbol/general form:  left-outer-join:  \( r \bowtie s \)   right-outer-join:  \( r \triangleright s \)   full outer join:  \( r \bowtie \triangleright s \)

Schema of result relation: concatenation of operand relations

Definition:

\( r \bowtie s \equiv \) natural join of \( r \) and \( s \) with tuples from \( r \) which do not have a match in \( s \) included in the result. Any missing values from \( s \) are set to null.

\( r \triangleright s \equiv \) natural join of \( r \) and \( s \) with tuples from \( s \) which do not have a match in \( r \) included in the result. Any missing values from \( r \) are set to null.

\( r \bowtie \triangleright s \equiv \) natural join of \( r \) and \( s \) with tuples from both \( r \) and \( s \) which do not have a match are included in the result. Any missing values are set to null.

Examples: Let  \( r(A,B) = \{(a, b), (c, d), (b,c)\} \) and let 
\( s(A,C) = \{(a, d), (s, t), (b, d)\} \)

then  \( r \bowtie s = (A,B,C) = \{(a,b,d), (b,c,d), (c,d,null)\}, \)
\( r \triangleright s = (A,B,C) = \{(a,b,d), (b,c,d), (s,null,t)\}, \) and
\( r \bowtie \triangleright s = (A,B,C) = \{(a,b,d), (b,c,d), (s,null,t), (c,d,null)\}, \)
Outer Join Operator Examples

### R

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

### S

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>12</td>
</tr>
</tbody>
</table>

### r = R ⨝ ⨝ ⨝ S

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>null</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>null</td>
</tr>
<tr>
<td>null</td>
<td>6</td>
<td>7</td>
<td>12</td>
</tr>
</tbody>
</table>

### r = R ⨝ ⨝ S

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>null</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>null</td>
</tr>
</tbody>
</table>

### r = R ⨝ ⨝ S

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>11</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>12</td>
<td>null</td>
<td></td>
</tr>
</tbody>
</table>
An Example Database

Suppliers

<table>
<thead>
<tr>
<th>snum</th>
<th>sname</th>
<th>status</th>
<th>city</th>
</tr>
</thead>
</table>

Parts

<table>
<thead>
<tr>
<th>pnum</th>
<th>pname</th>
<th>color</th>
<th>weight</th>
<th>city</th>
</tr>
</thead>
</table>

Jobs

<table>
<thead>
<tr>
<th>jnum</th>
<th>jname</th>
<th>numworkers</th>
<th>city</th>
</tr>
</thead>
</table>

Shipments

<table>
<thead>
<tr>
<th>snum</th>
<th>pnum</th>
<th>jnum</th>
<th>qty</th>
<th>date</th>
</tr>
</thead>
</table>

The last three pages of this set of notes contain screen shots for these for tables from a sample database using these tables. They might help to make some of the following examples more clear.
Query: List only the names (remove duplicates) of those suppliers who have a shipment with a quantity $\geq 15$.

```
SELECT DISTINCT sname
FROM Suppliers NATURAL JOIN Shipments
WHERE quantity $\geq 15$;
```

Note that Access does not support the natural join syntax shown above. In Access this query is expressed as shown on the next page.
Natural Join Example 1 In Access

SELECT DISTINCT suppliers.sname
FROM suppliers, shipments
WHERE quantity >= 15 AND suppliers.snum = shipments.snum;

This comma is the generic join operator in SQL.

This condition is called an “implicit join condition”. It specifies the criteria on which the join is to occur.

In this case, the primary key in suppliers and the foreign key (part of the primary key) in the shipments table must be the same.
SELECT DISTINCT suppliers.sname
FROM suppliers, shipments
WHERE quantity >= 15 AND suppliers.snum = shipments.snum;

SQL query entered in Access and executed showing results.
Query: List only the names (remove duplicates) of those cities in which both a supplier and a job are located.

```
SELECT DISTINCT Supplier.city
FROM Suppliers NATURAL JOIN Jobs;
```

**Access Version:**

```
SELECT DISTINCT Supplier.city
FROM Suppliers, Jobs
WHERE Suppliers.city = Jobs.city;
```
SQL query entered in Access and executed showing results.

```
SELECT DISTINCT Suppliers.city
FROM Suppliers, Jobs
WHERE Suppliers.city = Jobs.city
```
Query: List only the names (remove duplicates) of those jobs which receive a shipment from supplier number 1.

```
SELECT DISTINCT Jobs.jname
FROM Jobs NATURAL JOIN Shipments
WHERE Shipments.snum = 1;
```

**Access Version:**
```
SELECT DISTINCT Jobs.jname
FROM Jobs, Shipments
WHERE Jobs.jnum = Shipments.jnum
  AND Shipments.snum = 1;
```
SQL query entered in Access and executed showing results.

```
SELECT DISTINCT Jobs.jname 
FROM Jobs, Shipments 
WHERE Jobs.jnum = Shipments.jnum AND Shipments.snum = 1;
```
Left Outer Join Example

- List the supplier numbers and names along with the quantity of each order a supplier has and include supplier information even for suppliers who have no shipments.

**Access version:**

```
SELECT Suppliers.snum, Suppliers.sname, Shipments.quantity
FROM Suppliers LEFT OUTER JOIN Shipments
ON Suppliers.snum = Shipments.snum;
```

```
LEFT OUTER JOIN` syntax with **ON** keyword instead of **WHERE** causes supplier information to appear even if there is no corresponding shipment information for that supplier.
```
Left Outer Join Example

SQL query entered in Access and executed showing results.

```
SELECT Suppliers.snum, Suppliers.sname, Shipments.quantity
FROM Suppliers LEFT OUTER JOIN Shipments
ON Suppliers.snum = Shipments.snum
```

Suppliers 6, 7, 8, and 9 have no shipment.s
Right Outer Join Example

- List all the information about each shipment and the part number of every part that is not shipped by any supplier.

**Access version:**

```sql
SELECT Shipments.*, Parts.pnum
FROM Shipments RIGHT OUTER JOIN Parts
ON Shipments.pnum = Parts.pnum;
```

RIGHT OUTER JOIN syntax with ON keyword instead of WHERE causes part information to appear even if there is no corresponding shipment information for that part.
SQL query entered in Access and executed showing results.

Parts 5, 6, 7 and 10 are not being shipped.
Multiple Table Join Example 1

• List the supplier name and city for every supplier who has a shipment of a blue part.

SQL Version:

```
SELECT Suppliers.sname, Suppliers.city
FROM Suppliers NATURAL JOIN Shipments NATURAL JOIN Parts
WHERE Parts.color = "blue";
```

Access Version:

```
SELECT Suppliers.sname, Suppliers.city
FROM Suppliers, Shipments, Parts
WHERE Suppliers.snum = Shipments.snum
AND Shipments.pnum = Parts.pnum
AND Parts.color = "blue";
```

Each pair of tables requires an implicit join condition in the WHERE clause, matching primary keys against foreign keys.
SQL query entered in Access and executed showing results.
Multiple Table Join Example 2

• List the supplier names for those suppliers who supply a red part to any job in Tampa in a quantity > 20.

SQL Version:

```sql
SELECT Suppliers.sname
FROM Suppliers NATURAL JOIN Shipments NATURAL JOIN Parts
    NATURAL JOIN Jobs
WHERE Parts.color = "red" AND Jobs.city = "Tampa"
    AND Shipments.quantity > 20;
```

Access Version: (see next page)
SQL query entered in Access and executed showing results.
Processing Multiple Tables Using Subqueries

• A subquery is formed by placing a query inside a query, i.e., placing a SELECT statement (the inner query) inside a SELECT statement (the outer query).

• A subquery can occur in several different location: The options are:
  – 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 1 (Where clause)

• List the name of the supplier who shipped shipment number 6.

SQL:

```
SELECT Suppliers.sname
FROM Suppliers
WHERE snum = (SELECT snum
               FROM Shipments
               WHERE shipment_id = 6);
```

No reference is made in the inner query to any value in the outer query, hence this is a noncorrelated query.
SQL query entered in Access and executed showing results.

```
SELECT Suppliers.sname
FROM Suppliers
WHERE snum = (SELECT snum
FROM Shipments
WHERE shipment_id = 6)
```
Subquery Example 2 (Where clause)

• List the names of those suppliers who at least one shipment.

SQL:

SELECT Suppliers.sname
FROM Suppliers
WHERE snum IN (SELECT DISTINCT snum
FROM Shipments);

The IN operator is a set operator that checks to see if the left-hand operand (a value or set member instance) is contained in the right-hand operand (a set). The IN operator returns true or false.
SQL query entered in Access and executed showing results.
Subquery Example 3 (Having clause)

- List the part names and the total quantity shipped of that part for parts that are supplied in quantities greater than the average quantity of all parts supplied.

SQL/Access Version:

```
SELECT Shipments.pnum, SUM(Shipments.quantity) AS TotalQuantityShipped
FROM Shipments
GROUP BY pnum
HAVING SUM(Shipments.quantity) >
    (SELECT AVG(Shipments.quantity)
     FROM Shipments);
```
SQL query entered in Access and executed showing results.

```
SELECT Shipments.pnum, SUM(Shipments.quantity) AS TotalQuantityShipped
FROM Shipments
GROUP BY pnum
HAVING SUM(Shipments.quantity) > (SELECT AVG(Shipments.quantity)
FROM Shipments)
```
Subquery Example 4 (From clause)

- List the unique part names for all the blue parts that are shipped.

SQL Version:

```
SELECT DISTINCT pname
FROM Shipments NATURAL JOIN (SELECT pnum
FROM Parts
WHERE color = "blue");  
```

Access Version:

```
SELECT DISTINCT PB.pname
FROM (SELECT * FROM Parts INNER JOIN [Shipments] ON Parts.pnum = Shipments.pnum ) AS PB
WHERE PB.color="blue"
```
SQL query entered in Access and executed showing results.
Processing a noncorrelated subquery

1. The subquery executes and returns the customer IDs from the ORDER_T table.

   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.

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

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

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

   9 rows selected.

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.
SQL query entered in Access and executed showing results.
Correlated Subquery Example

Use this database for the next couple of examples.
Correlated Subquery Example

• Show all orders that include products finished in natural ash

SELECT DISTINCT order-id FROM order-line
WHERE EXISTS
  (SELECT * FROM product
   WHERE product-id = order-line.product-id
   AND finish = 'Natural ash');

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

The subquery is testing for a value that comes from the outer query.
Correlated Subquery Example

SQL query entered in Access and executed showing 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');
```
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

```
SELECT description, price, avg-price
FROM
(SELECT AVG(price) AS avgprice FROM product), product
WHERE price > avgprice;
```

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
Another Subquery Example

SQL query entered in Access and executed showing results.

```
SELECT product_description, standard_price, avgprice
FROM (SELECT AVG(standard_price) AS avgprice
      FROM product_t, product_t
     WHERE standard_price > avgprice)
```

<table>
<thead>
<tr>
<th>Product_Description</th>
<th>Standard_Price</th>
<th>avgprice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entertainment Center</td>
<td>$650.00</td>
<td>$440.63</td>
</tr>
<tr>
<td>8-Drawer Desk</td>
<td>$750.00</td>
<td>$440.63</td>
</tr>
<tr>
<td>Dining Table</td>
<td>$800.00</td>
<td>$440.63</td>
</tr>
</tbody>
</table>
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 contrasted with stored procedures

Procedures are called explicitly

- Call Procedure_name (parameter_value):
- Stored Procedure
  - code
  - returns value as performs routine

Triggers are event-driven

- Insert
- Update
- Delete
- Trigger
  - code
  - performs trigger action

Database
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;

SQL:20XX 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 | PL/I | 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
### Suppliers Table Instance

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>snum</td>
<td>AutoNumber</td>
</tr>
<tr>
<td>sname</td>
<td>Text</td>
</tr>
<tr>
<td>status</td>
<td>Number</td>
</tr>
<tr>
<td>city</td>
<td>Text</td>
</tr>
</tbody>
</table>

**Table Instance**

<table>
<thead>
<tr>
<th>snum</th>
<th>sname</th>
<th>status</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mark</td>
<td>4</td>
<td>Oviedo</td>
</tr>
<tr>
<td>2</td>
<td>Dave</td>
<td>30</td>
<td>Orlando</td>
</tr>
<tr>
<td>3</td>
<td>Tiffany</td>
<td>2</td>
<td>Winter Springs</td>
</tr>
<tr>
<td>4</td>
<td>Kristi</td>
<td>1</td>
<td>Orlando</td>
</tr>
<tr>
<td>5</td>
<td>Karen</td>
<td>3</td>
<td>Longwood</td>
</tr>
<tr>
<td>6</td>
<td>Cat</td>
<td>4</td>
<td>Oviedo</td>
</tr>
<tr>
<td>7</td>
<td>Tami</td>
<td>3</td>
<td>Winter Springs</td>
</tr>
<tr>
<td>8</td>
<td>Cindy</td>
<td>2</td>
<td>Tampa</td>
</tr>
<tr>
<td>9</td>
<td>Candace</td>
<td>17</td>
<td>London</td>
</tr>
</tbody>
</table>

* (New) 0
### Parts Table Instance

<table>
<thead>
<tr>
<th>pnum</th>
<th>pname</th>
<th>color</th>
<th>weight</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>bolt</td>
<td>red</td>
<td>3</td>
<td>Orlando</td>
</tr>
<tr>
<td>4</td>
<td>nut</td>
<td>blue</td>
<td>14</td>
<td>Tampa</td>
</tr>
<tr>
<td>5</td>
<td>flange</td>
<td>red</td>
<td>7</td>
<td>Miami</td>
</tr>
<tr>
<td>6</td>
<td>clamp</td>
<td>black</td>
<td>3</td>
<td>Orlando</td>
</tr>
<tr>
<td>7</td>
<td>nut</td>
<td>red</td>
<td>4</td>
<td>Orlando</td>
</tr>
<tr>
<td>8</td>
<td>nut</td>
<td>blue</td>
<td>5</td>
<td>Tampa</td>
</tr>
<tr>
<td>9</td>
<td>switch</td>
<td>green</td>
<td>3</td>
<td>Oviedo</td>
</tr>
<tr>
<td>10</td>
<td>gasket</td>
<td>brown</td>
<td>1</td>
<td>Tampa</td>
</tr>
</tbody>
</table>

(New)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pnum</td>
<td>AutoNumber</td>
</tr>
<tr>
<td>pname</td>
<td>Text</td>
</tr>
<tr>
<td>color</td>
<td>Text</td>
</tr>
<tr>
<td>weight</td>
<td>Number</td>
</tr>
<tr>
<td>city</td>
<td>Text</td>
</tr>
</tbody>
</table>
### Jobs Table Instance

<table>
<thead>
<tr>
<th>jnum</th>
<th>jname</th>
<th>numworkers</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>tiny job</td>
<td>1</td>
<td>Oviedo</td>
</tr>
<tr>
<td>5</td>
<td>small job</td>
<td>4</td>
<td>Tampa</td>
</tr>
<tr>
<td>6</td>
<td>bigger job</td>
<td>15</td>
<td>Jacksonville</td>
</tr>
<tr>
<td>7</td>
<td>huge job</td>
<td>45</td>
<td>Miami</td>
</tr>
<tr>
<td></td>
<td>(New)</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

#### Field Name

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>jnum</td>
<td>AutoNumber</td>
</tr>
<tr>
<td>jname</td>
<td>Text</td>
</tr>
<tr>
<td>numworkers</td>
<td>Number</td>
</tr>
<tr>
<td>city</td>
<td>Text</td>
</tr>
</tbody>
</table>
### Shipments Table Instance

<table>
<thead>
<tr>
<th>snum</th>
<th>pnum</th>
<th>jnum</th>
<th>quantity</th>
<th>shipment_ID</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>14</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
<td>22</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>7</td>
<td>25</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>5</td>
<td>12</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>4</td>
<td>15</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(New)</td>
</tr>
</tbody>
</table>

**Field Name** | **Data Type**
--- | ---
`snum` | Number
`pnum` | Number
`jnum` | Number
`quantity` | Number
`shipment_ID` | AutoNumber