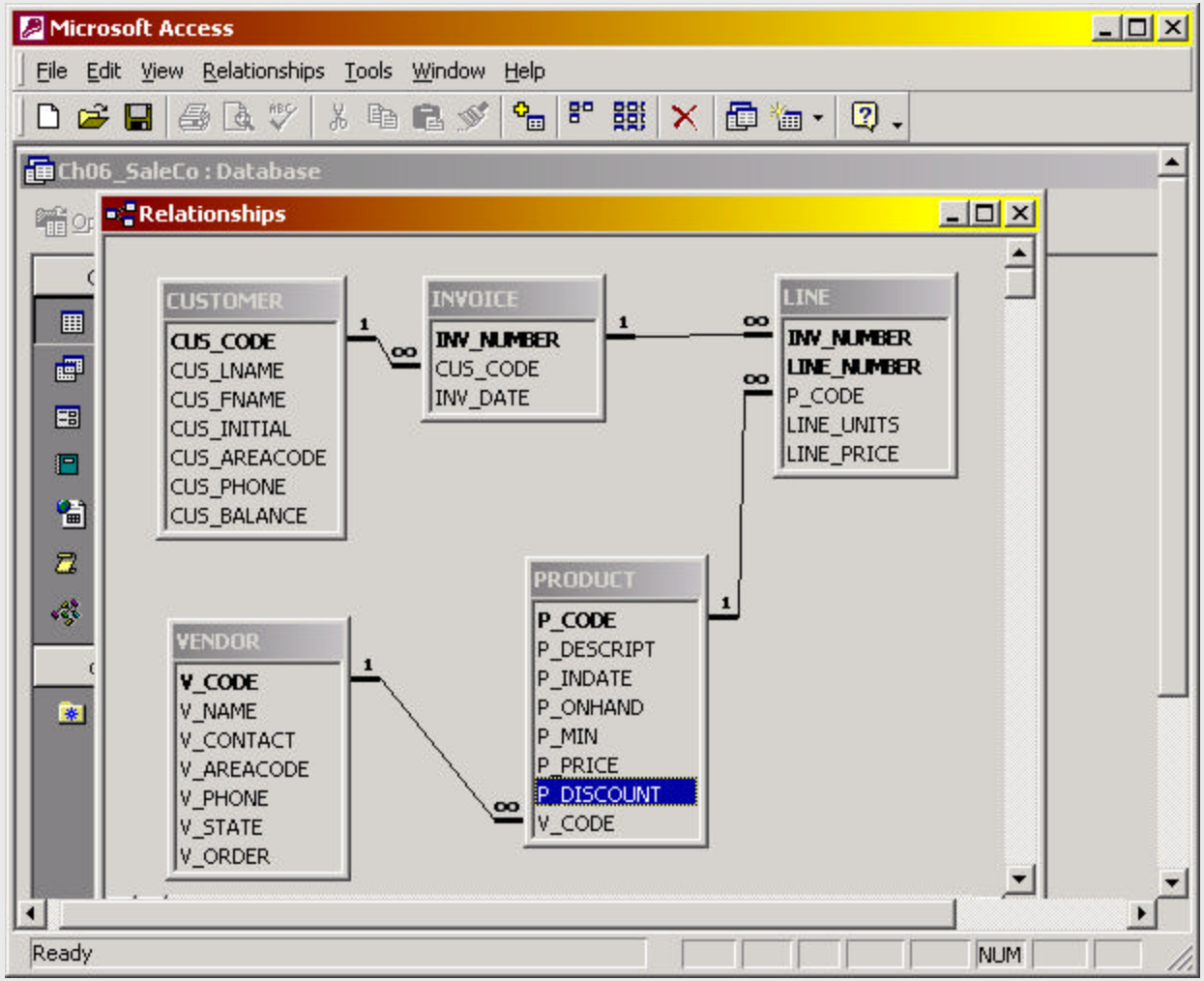
COP 4710: Database Systems
Spring 2004

-Day 20 – March 22, 2004 –
Introduction to SQL – Part 4

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CC1 211, 823-2790
http://www.cs.ucf.edu/courses/cop4710/spr2004

School of Electrical Engineering and Computer Science
University of Central Florida

An Example Database



SQL Join Operations

- The SQL join operations merge rows from two tables and returns the rows that:
  1. Have common values in common columns (natural join) or,
  2. Meet a given join condition (equality or inequality) or,
  3. Have common values in common columns or have no matching values (outer join).
- We've already examined the basic form of an SQL join which occurs when two tables are listed in the FROM clause and the WHERE clause specifies the join condition.
- An example of this basic form of the join is shown on the next page.

SQL Join Operations (cont.)

```
SELECT P_CODE, P_DESCRIPT, P_PRICE, V_NAME
   FROM PRODUCT, VENDOR
   WHERE PRODUCT.V_CODE = VENDOR.V_CODE;
```

- The FROM clause indicates which tables are to be joined. If three or more tables are specified, the join operation takes place two tables at a time, starting from left to right.
- The join condition is specified in the WHERE clause. In the example, a natural join is effected on the attribute V_CODE.
- The SQL join syntax shown above is sometimes referred to as an "old-style" join.
- The tables on pages 16 and 17, summarize the SQL join operations.

SQL Cross Join Operation

- A cross join in SQL is equivalent to a Cartesian product in standard relational algebra. The cross join syntax is:

```
SELECT column-list
    FROM table1, table2;
```

old style syntax

```
SELECT column-list
    FROM table1 CROSS JOIN table2;
```

new style syntax

SQL Natural Join Operation

- The natural join syntax is:

```
SELECT column-list
    FROM table1 NATURAL JOIN  table2;
```

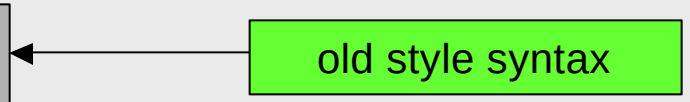
new style syntax

- The natural join will perform the following tasks:
  - Determine the common attribute(s) by looking for attributes with identical names and compatible data types.
  - Select only the rows with common values in the common attribute(s).
  - If there are no common attributes, return the cross join of the two tables.

SQL Natural Join Operation (cont.)

- The syntax for the old-style natural join is:

```
SELECT column-list
    FROM table1,  table2
    WHERE table1.C1 = table2.C2;
```



- One important difference between the natural join and the "old-style" syntax is that the natural join does not require the use of a table qualifier for the common attributes. The two SELECT statements shown on the next page are equivalent.

SQL Natural Join Operation (cont.)

```
SELECT CUS_NUM, CUS_LNAME,
          INV_NUMBER, INV_DATE
  FROM  CUSTOMER, INVOICE
  WHERE  CUSTOMER.CUS_NUM = INVOICE. CUS_NUM;
```

old style syntax

```
SELECT CUS_NUM, CUS_LNAME,
          INV_NUMBER, INV_DATE
  FROM  CUSTOMER NATURAL JOIN INVOICE;
```

old style syntax

Join With Using Clause

- A second way to express a join is through the USING keyword. This query will return only the rows with matching values in the column indicated in the USING clause. The column listed in the USING clause must appear in both tables.

old style syntax

- The syntax is:

```
SELECT column-list
    FROM table1 JOIN  table2 USING (common-column);
```


Join With Using Clause (cont.)

- An example:

```
SELECT  INV_NUMBER, P_CODE, P_DESCRIPT, LINE_UNITS,
        LINE_PRICE
  FROM  INVOICE JOIN LINE USING (INV_NUMBER)
        JOIN PRODUCT USING (P_CODE);
```

- As was the case with the natural join command, the JOIN USING does not required the use of qualified names (qualified table names). In fact, Oracle 9i will return an error if you specify the table name in the USING clause.

Join On Clause

- Both the NATURAL JOIN and the JOIN USING commands use common attribute names in joining tables.
- Another way to express a join when the tables have no common attribute names is to use the JOIN ON operand. This query will return only the rows that meet the indicated condition. The join condition will typically include an equality comparison expression of two columns. The columns may or may not share the same name, but must obviously have comparable data types.
- The syntax is:

```
SELECT column-list
    FROM table1 JOIN  table2 ON join-condition;
```

Join On Clause (cont.)

- An example:

```
SELECT INVOICE.INV_NUMBER, P_CODE, P_DESCRIPT, LINE_UNITS, LINE_PRICE
  FROM INVOICE JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER
       JOIN PRODUCT ON LINE.P_CODE = PRODUCT.P_CODE;
```

- Notice in the example query, that unlike the NATURAL JOIN and the JOIN USING operation, the JOIN ON clause requires the use of table qualifiers for the common attributes. If you do not specify the table qualifier you will get a "column ambiguously defined" error message.

- Keep in mind that the JOIN ON syntax allows you to perform a join even when the tables do not share a common attribute name.

Join On Clause (cont.)

- For example, to general a list of all employees with the manager's name you can use the recursive query shown below which utilizes the JOIN ON clause.

```
SELECT E.EMP_MGR, M.EMP_LNAME, E.EMP_NUM, E.EMP_LNAME
   FROM EMP E JOIN EMP M ON E.EMP_MGR = M.EMP_NUM
   ORDER BY E.EMP_MGR;
```


Outer Joins

- We saw the forms for the LEFT OUTER JOIN and the RIGHT OUTER JOIN in the previous set of notes.
- There is also a FULL OUTER JOIN operation in SQL. A full outer join returns not only the rows matching the join condition (that is, rows with matching values in the common column(s)), but also all the rows with unmatched values in either side table.
- The syntax of a full outer join is:

```
SELECT column-list
    FROM table1 FULL [OUTER] JOIN  table2 ON join-condition;
```

Outer Joins (cont.)

- The following example will list the product code, vendor code, and vendor name for all products and include all the product rows (products without matching vendors) and also all vendor rows (vendors without matching products):

```
SELECT  P_CODE, VENDOR.V_CODE, V_NAME
    FROM VENDOR FULL OUTER JOIN PRODUCT
            ON VENDOR.V_CODE = PRODUCT.V_CODE;
```


Summary of SQL JOIN Operations

Join Classification	Join Type	SQL Syntax Example	Description
Cross	CROSS JOIN	SELECT * FROM T1, T2;	Old style. Returns the Cartesian product of T1 and T2
		SELECT * FROM T1 CROSS JOIN T2;	New style. Returns the Cartesian product of T1 and T2.
Inner	Old Style JOIN	SELECT * FROM T1, T2 WHERE T1.C1 = T2.C1	Returns only the rows that meet the join condition in the WHERE clause – old style. Only rows with matching values are selected.
	NATURAL JOIN	SELECT * FROM T1 NATURAL JOIN T2	Returns only the rows with matching values in the matching columns. The matching columns must have the same names and similar data types.
	JOIN USING	SELECT * FROM T1 JOIN T2 USING (C1)	Returns only the rows with matching values in the columns indicated in the USING clause.
	JOIN ON	SELECT * FROM T1 JOIN T2 ON T1.C1 = T2.C1	Returns only the rows that meet the join condition indicated in the ON clause.

Summary of SQL JOIN Operations (cont.)

Join Classification	Join Type	SQL Syntax Example	Description
Outer	LEFT JOIN	SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1= T2.C1	Returns rows with matching values and includes all rows from the left table (T1) with unmatched values.
	RIGHT JOIN	SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1= T2.C1	Returns rows with matching values and includes all rows from the right table (T2) with unmatched values.
	FULL JOIN	SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.C1= T2.C1	Returns rows with matching values and includes all rows from both tables (T1 and T2) with unmatched values.

Subqueries and Correlated Queries

- The use of joins allows a RDBMS go get information from two or more tables. The data from the tables is processed simultaneously.
- It is often necessary to process data based on other processed data. Suppose, for example, that you want to generate a list of vendors who provide products. (Recall that not all vendors in the VENDOR table have provided products – some of them are only potential vendors.)
- The following query will accomplish our task:

```
SELECT  V_CODE, V_NAME
    FROM VENDOR
    WHERE V_CODE NOT IN (SELECT V_CODE    FROM PRODUCT);
```

Subqueries and Correlated Queries (cont.)

- A subquery is a query (SELECT statement) inside a query.
- A subquery is normally expressed inside parentheses.
- The first query in the SQL statement is known as the outer query.
- The second query in the SQL statement is known as the inner query.
- The inner query is executed first.
- The output of the inner query is used as the input for the outer query.
- The entire SQL statement is sometimes referred to as a nested query.

Subqueries and Correlated Queries (cont.)

- A subquery can return:

1. One single value (one column and one row). This subquery can be used anywhere a single value is expected. For example, in the right side of a comparison expression.

2. A list of values (one column and multiple rows). This type of subquery can be used anywhere a list of values is expected. For example, when using the IN clause.

3. A virtual table (multi-column, multi-row set of values). This type of subquery can be used anywhere a table is expected. For example, in the FROM clause.

4. No value at all, i.e., NULL. In such cases, the output of the outer query may result in an error or null empty set, depending on where the subquery is used (in a comparison, an expression, or a table set).

WHERE Subqueries

- The most common type of subquery uses an inner SELECT subquery on the right hand side of a WHERE comparison expression.
- For example, to find all products with a price greater than or equal to the average product price, the following query would be needed:

```
SELECT P_CODE, P_PRICE
  FROM PRODUCT
  WHERE P_PRICE >= (SELECT AVG(P_PRICE)
                    FROM PRODUCT);
```


WHERE Subqueries (cont.)

- Subqueries can also be used in combination with joins.
- The query below lists all the customers that ordered the product “Claw hammer”.

```
SELECT DISTINCT CUS_CODE, CUS_LNAME, CUYS_FNAME
   FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
                 JOIN LINE USING (INV_NUMBER)
                 JOIN PRODUCT USING (P_CODE)
    WHERE P_CODE = (SELECT P_CODE
                    FROM PRODUCT
                     WHERE P_DESCRIPT = “Claw hammer”);
```


WHERE Subqueries (cont.)

- Notice that the previous query could have been written as:

```
SELECT DISTINCT CUS_CODE, CUS_LNAME, CUYS_FNAME
   FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
                 JOIN LINE USING (INV_NUMBER)
                 JOIN PRODUCT USING (P_CODE)
    WHERE  P_DESCRIPT = 'Claw hammer');
```

- However, what would happen if two or more product descriptions contain the string "Claw hammer"?
  - You would get an error message because only a single value is expected on the right hand side of this expression.

IN Subqueries

- To handle the problem we just saw, the IN operand must be used.
- The query below lists all the customers that ordered any kind of hammer or saw.

```
SELECT DISTINCT CUS_CODE, CUS_LNAME, CUYS_FNAME
   FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
                 JOIN LINE USING (INV_NUMBER)
                 JOIN PRODUCT USING (P_CODE)
    WHERE P_CODE IN (SELECT P_CODE
                     FROM PRODUCT
                      WHERE P_DESCRIPT LIKE '%hammer%'
                        OR P_DESCRIPT LIKE '%saw%');
```


HAVING Subqueries

- It is also possible to use subqueries with a HAVING clause.
- Recall that the HAVING clause is used to restrict the output of a GROUP BY query by applying a conditional criteria to the grouped rows.
- For example, the following query will list all products with the total quantity sold greater than the average quantity sold.

```
SELECT DISTINCT P_CODE, SUM(LINE_UNITS)
   FROM LINE
    GROUP BY P_CODE
     HAVING SUM(LINE_UNITS) > (SELECT AVG(LINE_UNITS)
                                FROM LINE);
```

Multi-row Subquery Operators: ANY and ALL

- The IN subquery uses an equality operator; that is, it only selects those rows that match at least one of the values in the list. What happens if you need to do an inequality comparison of one value to a list of values?

- For example, suppose you want to know what products have a product cost that is greater than all individual product costs for products provided by vendors from Florida.

```
SELECT P_CODE, P_ONHAND*P_PRICE
   FROM PRODUCT
    WHERE P_ONHAND*P_PRICE > ALL (SELECT P_ONHAND*P_PRICE
                                  FROM PRODUCT
                                  WHERE V_CODE IN (SELECT V_CODE
                                                   FROM VENDOR
                                                   WHERE V_STATE= 'FL'));
```


FROM Subqueries

- In all of the cases of subqueries we've seen so far, the subquery was part of a conditional expression and it always appeared on the right hand side of an expression. This is the case for WHERE, HAVING, and IN subqueries as well as for the ANY and ALL operators.
- Recall that the FROM clause specifies the table(s) from which the data will be drawn. Because the output of a SELECT statement is another table (or more precisely, a "virtual table"), you could use a SELECT subquery in the FROM clause.
- For example, suppose that you want to know all customers who have purchased products 13-Q2/P2 and 23109-HB. Since all product purchases are stored in the LINE table, it is easy to find out who purchased any given product just by searching the P_CODE attribute in the LINE table. However, in this case, you want to know all customers who purchased both, not just one.
- The query on the next page accomplishes this task.

FROM Subqueries (cont.)

```
SELECT DISTINCT CUSTOMER.CUS_CODE, CUSTOMER.LNAME
  FROM CUSTOMER, (SELECT INVOICE.CUS_CODE
                    FROM INVOICE NATURAL JOIN LINE
                    WHERE P_CODE = '13-Q2/P2') CP1,
                 (SELECT INVOICE.CUS_CODE
                    FROM INVOICE NATURAL JOIN LINE
                    WHERE P_CODE = '23109-HB') CP2
  WHERE CUSTOMER.CUS_CODE = CP1.CUS_CODE
                  AND CP1.CUS_CODE = CP2.CUS_CODE;
```