Inner Join Queries

Join Introduction

A join in its purest definition simply means combining data from more than one table. However, as you'll see throughout the next learning units, there are several ways tables can be joined. This unit concentrates on the inner join, where one table is joined (matched) to another table on a common column. An inner join is also referred to as an equality join, equijoin or simple join. A self-join, where a table is joined to itself, is a special type of inner join that will be discussed also.

Inner Join using WHERE clause: Traditional method

One of the most common ways to perform an inner join is to compare common columns on two tables in the WHERE clause. For example, if we wanted to know which orders were placed by which customers, it makes sense to match the customer numbers: WHERE Customers.Customer# = Orders.Customer#. Any data selected will be specific to a customer. Common columns exist throughout the JustLee Books database: Books and Publisher have PubID in common, BookAuthor and Author have AuthorID in common, just to name a few. To join more than 2 tables with the WHERE, you use the AND operator: WHERE Customers.Customer# = Orders.Customer# AND Orders.Order# = OrderItems.Order#

In general, anytime there is a relationship between two tables, a common column exists that can be used to join them together in a query. Most inner joins are done using the primary key/foreign key relationship. By studying the Entity Relationship Diagram (ERD) of the database, join operations are easier to understand. *Keep in mind, common columns do not have to have the same, but most the time they do.*

For a join to occur, multiple tables must be listed in the FROM clause: FROM Customers, Orders. Table names can be given an "alias", and the alias can be used in place of the table name for brevity. To create an alias, code a reference after the table name on the FROM clause: FROM Customers C. Then use the alias in place of the table name: WHERE C.Customer# =. Note: once assigned, an alias must be used in place of that table name for the entire SQL statement.

Since common columns tend to have the same name in multiple tables, they need to be "qualified." This means attaching the table name in front of the column so that there is no ambiguity (confusion) as to which table the data is coming from: Customers.Customer#. Qualifying needs to be done on the common column no matter which clause it is coded on – SELECT, WHERE, ORDER BY, etc. If Oracle gives you an ambiguity error, you know how to correct it. It is common for a query to not only have to join tables, but also satisfy some other condition as well. In our customer orders example above, what if we only wanted customers that were from Iowa? We would still do our join condition first, then add the state condition: WHERE Customers.Customer# = Orders.Customer# AND State = 'IA'. Notice that the state condition is in addition to, so the use of the AND operator is correct. If additional conditions are required, the use of parenthesis is critical to the correct evaluation of the WHERE clause and valid results returned. Back to our example, if we wanted customers in Iowa or Michigan, it would be: WHERE

Customers.Customer# = Orders.Customer# AND (State = 'IA' OR State = 'MI'). Coding it as: WHERE Customers.Customer# = Orders.Customer# AND State = 'IA' OR State = 'MI' would evaluate incorrectly due to the rules of logic.

If more than two tables are needed to satisfy a query, the process is the same. Just determine the tables required, the relationships between the tables, and then code the appropriate join condition:
FROM Customers C, Orders O, OrderItems OI WHERE C.Customer# = O.Customer# AND O.Order# = OI.Order#

Inner Join using JOIN Operator

Oracle supports the use of the JOIN operator as an alternative to joining with the WHERE clause. The JOIN operator is used in conjunction with the FROM clause when specifying tables. In effect, the "joining" is done prior to the WHERE clause, as opposed to during it. The JOIN operator has three options: NATURAL JOIN, JOIN USING, JOIN ON.

A NATURAL JOIN can only be used if the columns being "joined" have the same name and data type. Caution should be exercised with this approach, since two tables may have a column with the same name, but those columns are not related. Results could be unpredictable. For instance, say two tables have a LastName column, but LastName in one table is for customers, and LastName for the other table is for sales rep.

Example of a NATURAL JOIN: FROM Customers NATURAL JOIN Orders. This would "join" on the customer number column since a relationship exists between the tables on this column (one customer – many orders). Also no qualifiers can be used on the joining column. If more than 2 tables are necessary: FROM Customers NATURAL JOIN Orders NATURAL JOIN OrderItems.

Adding the USING clause allows a specific reference to the column to be used to join on. This prevents SQL from joining on same-name columns with no relationship. Using the previous example, the syntax would be: FROM Customers JOIN Orders USING (Cust#). If more than 2 tables are necessary: FROM Customers JOIN Orders USING (Cust#) JOIN OrderItems USING (Order#).

The USING clause works for a lot of situations, but sometimes tables have a common column, but that column does not have the same exact name. That's when the ON keyword comes in handy. Again back to the previous example, let's say we have Cust# on the Orders table, but CustNum on the Customers table. Then the query would have to use the ON operator as follows: FROM Customers C JOIN Orders O ON C.Cust# = O.CustNum. If more than 2 tables are necessary: FROM Customers C JOIN Orders O ON C.Cust# = O.CustNum JOIN OrderItems OI ON O.Order# = OI.Order# .Experienced SQL programmers like the use of JOIN ON because it simplifies the WHERE clause on complex queries.

NOTE: when joining tables on the FROM, no commas are ever used to separate the tables, as they are when joining on the WHERE.

Non-equality Joins

Sometimes it's handy to join tables where the columns being used don't have an exact match, but a column on one table falls within a range of values specified on another table; more of a "lookup" feature. This type of join can be implemented traditionally using the WHERE clause, or with the JOIN ON method. The BETWEEN operator is used on either method to specify the columns for the range. Examples are: FROM Books B, Promotion P WHERE B.Retail BETWEEN P.MinRetail AND P.MaxRetail or FROM Books B JOIN Promotion P ON B.Retail BETWEEN P.MinRetail AND P.MaxRetail.

Self-joins

At times a table must be joined to itself in order to solve a query request. I think of this as the table looking at itself in a mirror in order to see (reference) another column. As an example, assume each employee has a record on the employee table, and each employee has an employee number. Assume managers are also just another employee and have a record on the employee table. On the employee table, in addition to the employee number, there is a manager column that contains the employee number of the manager for each employee. If we wanted to figure out the manager's name, the employee table has to join to itself (look in the mirror) to solve this request. This might be written as: SELECT E2.Name FROM Employee E1, Employee E2 WHERE E1.Mgr# = E2.EmpNum. The JOIN ON method can be used also and would be: SELECT E2.Name FROM Employee E1 JOIN Employee E2 ON E1.Mgr# = E2.EmpNum