Table of Contents

One of the goals of a relational database is to reduce redundancy; we want to store descriptive data one time only. To do this we need to split data across several tables- so we have a Customer table and an Order table and an Order details table. As a consequence, we will often need to write queries that bring the data back together again from two or more tables. We will need to indicate in the query how these tables are related. We will first discuss table aliases which are commonly used when our query involves more than one table. Then we will look at the ANSI standard inner join syntax.

1. Qualified references

We know that each column in a table needs to have a unique name. But we can have the same identifier for columns in different tables. It is common, and good style, for the pk column in the parent table and the fk column in the child table to have the same name. For example, with the AltgeldMart tables, we have a column for the product id in both the products table and in the orderDetails table; in each of these tables the column name is prod_id. But we also store an employee identifier in the employees table and in the orderHeaders table; in the employees table it makes sense to call this attribute emp_id and it makes sense in the orderHeaders table to call this attribute sales_rep_id.

If we are joining two tables, we may have to qualify any reference to a column name which is the same in the two tables. (The exception is the joining column when we use the Using (col) syntax.) The format for a qualified name is `tblName.ColumnName`.

We can qualify all of the column names in the query. If this is a single table query, there is no need to do this. With a multi-table query, the query may be somewhat more efficient if we fully qualify the column names.

For most of the example in this discussion, I will not fully qualify all of the column names since the queries are easier to read without the qualification. The purpose of these demos is to show you techniques and syntax- and ease of reading is more important for this purpose than execution efficiency.

An **ambiguous reference** means that you have a column identifier in your query and the same identifier is used in more than one of the tables used in the query. Therefore the system does not know which column is being referenced. This is an error.

2. Using a Table Alias

The table aliases (correlation names) are alternate names for tables. The table alias is defined in the From clause of the SQL statement and is limited in scope to that statement. The table alias is not saved on the server. Once you establish a table alias in a query, you need to use that alias, not the table name, in the other clauses of that query.

Some types of queries require the use of table aliases since the same table is included in the query more than once. In other queries the use of table aliases is optional.

The use of table aliases is very common in SQL and you need to be aware of its use. However, poorly defined table aliases can make a query harder to read and understand.

The table alias is commonly a single letter but single letter names are often difficult to read and remember. You should not have to keep referring to the From clause to interpret the Select clause. Table aliases should be long enough to make them meaningful.

Demo 01: A single table select without an explicit table alias

```
select dept_id, dept_name
from emp_departments;
```

```
   DEPT_ID DEPT_NAME
---------- --------------------------------------------------
        10 Administration
        20 Marketing
        30 Development
        35 Cloud Computing
       210 IT Support
       215 IT Support
        80 Sales
        90 Shipping
        95 Logistics

 9 rows selected
```

Demo 02: A single table select with a table alias; this will give us the same output as the first query. The table alias is now used to refer to the table.

```
select dp.dept_id, dp.dept_name
from emp_departments dp;
```

The use of the key word As is not allowed with a table alias.

We could rewrite the second query as

```
select dept_id,dept_name
from emp_departments dp;
```

But we cannot use the following; once we set up the alias we cannot use the table name as a qualifier.

```
select prd_warehouses.warehouse_id
, prd_warehouses.loc_id
from prd_warehouses  wh;
```

The error message is ORA-00904: " EMPLOYEE"."DEPARTMENTS"."DEPT_NAME": invalid identifier

3. SQL inner joins

We can use several types of joins between tables. We start with inner joins. For a row to appear in the result returned by an inner join, there needs to be matching data in the joining columns in the two tables. There are several ways to implement the inner join.

These examples show the ANSI standard Condition Join syntax and the Column Name join syntax. Most dbms now support these syntax models.

The legacy comma join syntax is discussed later. **For assignments in this class you are required to use a syntax that does the join in the From clause**. In a job situation where you are reading and maintaining old code, you will need to understand the legacy syntax which expresses the joining condition as a criterion in the Where clause along with any filter criteria.

4. ANSI inner join: Column Name Join (the USING clause)

Let's start with an example of an inner join between the employee table and the department table. We want to see the name of the employee (which is in the employee table) and the name of the department which is in the department table. If we described this situation to someone we might say that we are **joining** the data in these two tables and that we are **using** the department id to associate each employee row with the correct department row. We also want to display the department id. If someone asks us which department id we want- the one from the employee table or the one from the department table, we would reply that we don't care because they have to be the same value.

The ANSI syntax shown here models that way of talking about the join.

Demo 03: Inner join employees and their dept

```
select emp_id
, name_last as "Employee"
, dept_name
from emp_employees
INNER JOIN emp_departments USING (dept_id);
```

```
EMP_ID               Employee                 DEPT_NAME
-------------------- ------------------------ ---------------------------------
100                  King                     Administration
201                  Harts                    Marketing
205                  Higgs                    Development
108                  Green                    Development
101                  Koch                     Development
206                  Geitz                    Development
204                  King                     Development
110                  Chen                     Development
109                  Fiet                     Development
. . . .
```

Discussion:

With this syntax, the joining column must have the same name in both tables.

We do not qualify the column names in common- since we are really using a generated column that Oracle produced for us. We do need to put parentheses around the common column name in the USING clause.

We need to qualify any other column names that would cause an ambiguous reference.

All of the information about the joining of the tables is placed in the From clause. The From clause is supposed to indicate the data source and this syntax defines the data source.

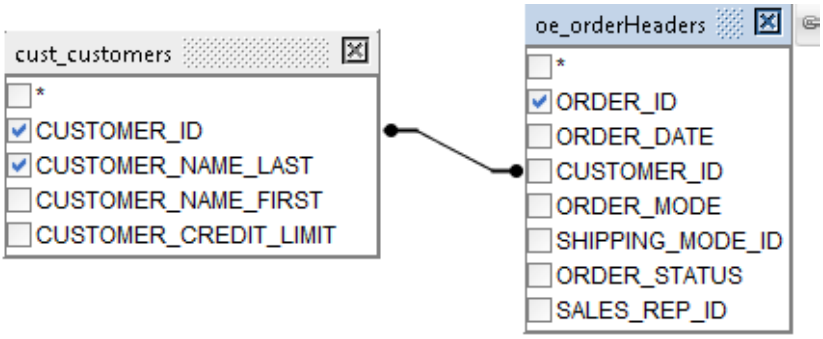
It might seem that the dbms could figure out which columns are the joining columns- particularly if we define the relationships when we create the table- but it doesn't.

Another thing to be aware of is if we have any department with no employees, the inner join will not display them. To get those rows we would need an outer join which we will discuss later.

Demo 04: Inner join with USING - show customers and their orders.

```
select cust_id
, oe_customers.cust_name_last as "Customer"
, oe_order_headers.ord_id
from oe_customers
INNER JOIN oe_order_headers USING (cust_id)
order by cust_id;
```

```
CUSTOMER_ID Customer                    ORDER_ID
----------- ------------------------- ----------
     400300 McGold                           378
     401250 Morse                            506
     401250 Morse                            301
     401250 Morse                            113
     401250 Morse                            106
     401250 Morse                            552
     401250 Morse                            119
. . .  rows omitted
```



We can join more than two tables- we just add them to the From clause in a logical order and set up the joining column for each pair of tables as we go. In the next query we get one row for each product a customer has ordered.

Tables are joined two at a time. The customer table is joined to the orderHeaders table to form a virtual table, which is then joined to the orderDetails table to form another virtual table used as the table expression. If you have a straight line inner join, start listing the tables with a table at one end of the path and then list the tables in order. Some people list the tables erratically which can make the joins harder to understand and error prone.

The key word Join and Inner Join both define an inner join. I will skip the work Inner in the following queries to make these easier to read. There is another type of join called an outer join we will discuss later.

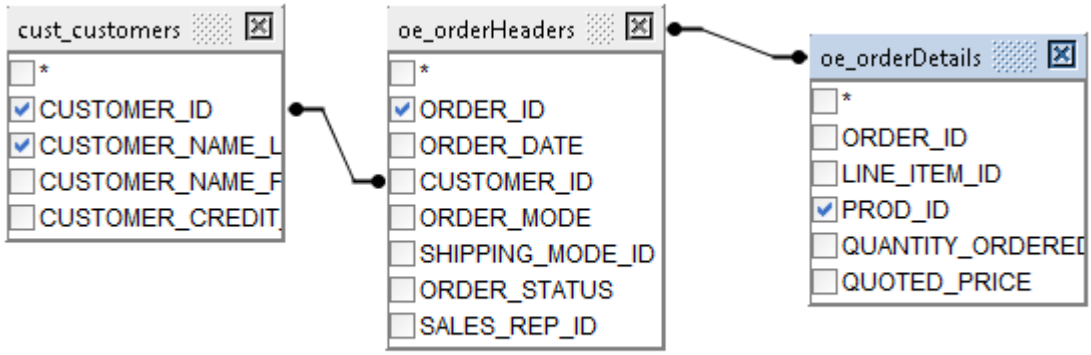
Class standards require that you start a new line for each Join keyword and keep the Using phrase on the same line as the table name. This style of SQL layout makes it easy to scan down the left edge of the query and see the tables involved.

Demo 05: three table join; inner join with column name join

```
select cust_id
, oe_customers.cust_name_last as "Customer"
, ord_id
, oe_order_details.prod_id
from oe_customers
JOIN oe_order_headers  USING (cust_id)
JOIN oe_order_details  USING (ord_id)
order by cust_id, ord_id;
```

```
CUSTOMER_ID Customer                    ORDER_ID    PROD_ID
----------- ------------------------- ---------- ----------
     400300 McGold                           378       1125
     400300 McGold                           378       1120
     401250 Morse                            106       1060
     401250 Morse                            113       1080
     401250 Morse                            119       1070
     401250 Morse                            301       1100
     401250 Morse                            552       2984
```

```
     401250 Morse                          552       2014
     401890 Northrep                       112       1110
     401890 Northrep                       519       1110
     401890 Northrep                       519       1020
. . .  rows omitted
```



Demo 06: four table join; inner join with USING

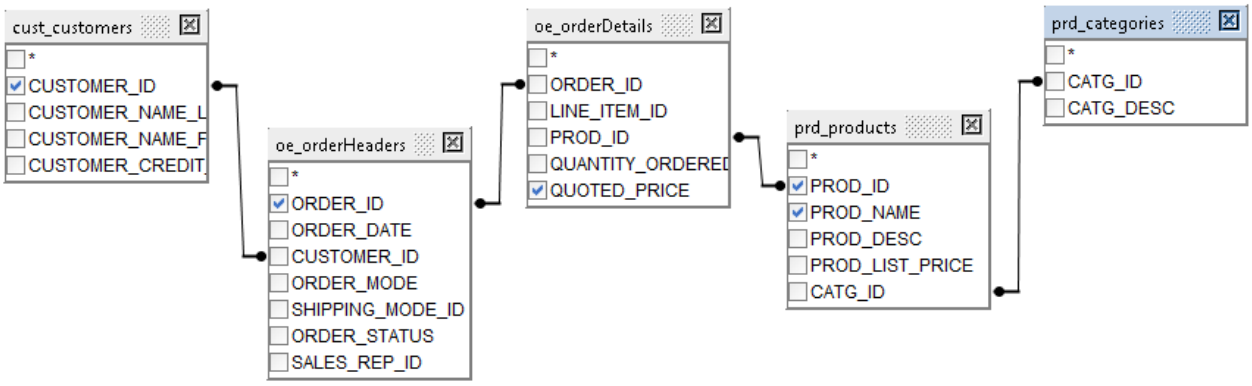
```
select customer_id
 , cust_customers.customer_name_last as "Customer"
 , order_id
 , prod_id
 , prd_products.prod_name
from cust_customers
join oe_orderHeaders using ( customer_id )
join oe_orderDetails using ( order_id )
join prd_products using ( prod_id ) ;
```

```
   CUST_ID Customer                    ORD_ID    PROD_ID PROD_NAME
---------- ------------------------ ---------- ---------- ------------
    403000 Williams                        808       1000 Hand Mixer
    903000 McGold                          313       1000 Hand Mixer
    404100 Button                          303       1000 Hand Mixer
    402100 Morise                          115       1000 Hand Mixer
    403000 Williams                        808       1010 Weights
    903000 McGold                          551       1010 Weights
    903000 McGold                          550       1010 Weights
    403000 Williams                        528       1010 Weights
    408770 Clay                            405       1010 Weights
. . .  rows omitted
```

Demo 07: five table join including a row filter for appliances.

```
select customer_id
 , order_id
 , prod_id
 , prd_products.prod_name
 , oe_orderDetails.quoted_price
from cust_customers
join oe_orderHeaders using ( customer_id )
join oe_orderDetails using ( order_id )
join prd_products using ( prod_id )
join prd_categories using ( catg_id )
where catg_desc in ( 'APPLIANCES' ) ;
```

CUSTOMER_ID	ORDER_ID	PROD_ID	PROD_NAME	QUOTED_PRICE
903000	306	1125	Dryer	500
903000	306	1120	Washer	500
900300	307	1125	Dryer	450
900300	307	1120	Washer	450
400300	378	1125	Dryer	450
400300	378	1120	Washer	450
403000	412	1130	Mini Freezer	149.99
409150	415	1125	Dryer	500
404100	605	1130	Mini Freezer	112.95
403000	109	1130	Mini Freezer	149.99
404950	110	1130	Mini Freezer	149.99
402100	114	1130	Mini Freezer	125
402100	115	1120	Washer	475
403010	118	1125	Dryer	475
409030	130	1125	Dryer	500
409030	130	1120	Washer	500
404100	2503	1130	Mini Freezer	149.99
404100	2504	1130	Mini Freezer	149.99
404100	2805	1130	Mini Freezer	112.95
403000	2509	1130	Mini Freezer	149.99
403000	4511	1130	Mini Freezer	149.99
409150	3518	1125	Dryer	500
409150	2218	1125	Dryer	500
409150	223	1130	Mini Freezer	148.99
409150	718	1125	Dryer	500
409150	523	1130	Mini Freezer	149.99
403000	529	1130	Mini Freezer	149.99



5. ANSI inner join: Condition Join (the ON clause)

Sometimes we need to join two tables that have different names for the joining column. In this example we are joining the order table to the employee table. In the employee table employees are identified with an employee_id. The order table uses the term sales_rep_id to refer to the same values.

Now we use the key word ON instead of Using and we list the two columns with an equality operator. We would not really have to qualify these columns, since they have different names, but it is common, and helpful, to do so.

Demo 08: two table join- inner join with ON clause. In this case, the joining columns have different names and you cannot have a USING clause.

```
select oe_orderHeaders.order_id
 , oe_orderHeaders.customer_id
 , emp_employees.emp_id
 , emp_employees.name_last as "SalesRep"
```

```
from oe_orderHeaders
join emp_employees on oe_orderHeaders.sales_rep_id = emp_employees.emp_id
order by oe_orderHeaders.order_id ;
```

ORD_ID	CUST_ID	EMP_ID	SalesRep
105	403000	150	Tuck
106	401250	150	Tuck
107	403050	150	Tuck
108	403000	155	Hiller
109	403000	155	Hiller
110	404950	155	Hiller
111	403000	150	Tuck
. . . rows omitted			

It is legal to use the On syntax when you are joining two tables which use the same column name for the joining column.

Demo 09: Inner join with ON clause - Show customers and their orders. Now you have to qualify the customer_id column because we are not using the column name join and cusomer_id appears in more than one of these tables.

```
select cust_customers.customer_id
 , cust_customers.customer_name_last as "Customer"
 , oe_orderHeaders.order_id
from cust_customers
join oe_orderHeaders on cust_customers.customer_id =
oe_orderHeaders.customer_id ;
```

CUST_ID	Customer	ORD_ID
400300	McGold	378
401250	Morse	506
401250	Morse	106
401250	Morse	113
401250	Morse	301
401250	Morse	119
. . . rows omitted		

You can combine both syntaxes in the same query.

Demo 10: Inner join with a USING and an ON clause- Show customers and their orders and their sales rep.

```
select customer_id
 , cust_customers.customer_name_last as "Customer"
 , oe_orderHeaders.order_id
 , emp_employees.emp_id
 , emp_employees.name_last as "SalesRep"
from cust_customers
join oe_orderHeaders using ( customer_id )
join emp_employees on oe_orderHeaders.sales_rep_id = emp_employees.emp_id ;
```

CUST_ID	Customer	ORD_ID	EMP_ID	SalesRep
401250	Morse	119	155	Hiller
401250	Morse	552	150	Tuck
401250	Morse	301	150	Tuck
401250	Morse	506	150	Tuck
401250	Morse	113	150	Tuck
401250	Morse	106	150	Tuck
401890	Northrep	519	155	Hiller
. . . rows omitted				

Demo 11: Inner join Using syntax 1 & 2- Show customers and their orders and their sales rep.

```
select customer_id
 , cust_customers.customer_name_last as "Customer"
 , oe_orderHeaders.order_id
 , emp_employees.emp_id
 , emp_employees.name_last as "SalesRep"
 , emp_departments.dept_name
from cust_customers
join oe_orderHeaders using ( customer_id )
join emp_employees on oe_orderHeaders.sales_rep_id = emp_employees.emp_id
join emp_departments using ( dept_id )
order by customer_id
 , oe_orderHeaders.order_id ;
```

```
   CUST_ID Customer                    ORD_ID     EMP_ID SalesRep   DEPT_NAME
---------- ------------------------ ---------- ---------- ---------- ----------

    400300 McGold                          378        150 Tuck       Sales
    401250 Morse                           106        150 Tuck       Sales
    401250 Morse                           113        150 Tuck       Sales
    401250 Morse                           119        155 Hiller     Sales
    401250 Morse                           301        150 Tuck       Sales
    401250 Morse                           506        150 Tuck       Sales
    401250 Morse                           552        150 Tuck       Sales
    401890 Northrep                        112        145 Russ       Sales
    401890 Northrep                        519        155 Hiller     Sales
. . .  rows omitted
```

Demo 12: Rewriting the query with table aliases can make this easier to read.

```
select customer_id
 , cs.customer_name_last as "Customer"
 , oh.order_id
 , em.emp_id
 , em.name_last as "SalesRep"
 , dp.dept_name
from cust_customers cs
join oe_orderHeaders oh using ( customer_id )
join emp_employees em on oh.sales_rep_id = em.emp_id
join emp_departments dp using ( dept_id )
order by customer_id
 , oh.order_id ;
```

Demo 13: Five table join including a row filter for appliances using the condition join syntax. I find it easier to get all the joins written if I have a pattern. My pattern is to use 2 character table aliases. This reduces the problems with using O for the order headers table and then using D for the order details table.
When I use the condition join syntax I generally write the join as the prior table.col = this table.col.

```
select
  CS.customer_id
, OH.order_id
, OD.prod_id
, PR.prod_name
, OD.quoted_price
from cust_customers       CS
join oe_orderHeaders   OH  on CS.customer_id = OH.Customer_id
join oe_orderDetails   OD  on OH.order_id  = OD.order_id
join prd_products       PR  on OD.prod_id = PR.prod_id
join prd_categories     CT  on PR.catg_id = CT.catg_id
where CT.catg_desc in('APPLIANCES');
```

There is a small difference in using a Select * from tbl1 join tbl2 depending on whether you use the column name join and the condition join. The condition join show two copies on the joining column and the column name join shows only one column in the result with the joining column.

6. NATURAL JOIN

This join can be used if the columns in common have the same name. This will join on any and all columns having the same identifier in the two tables. Therefore it creates maintenance problems if attributes are renamed or additional columns are added to the tables. Suppose we had two tables such as a customer table and a salesrep table which should be joined on a salesrepID column. But perhaps we also have attributes named City and State in each of these tables. If we did a natural join, the join would be on all of these columns. We will not use the Natual Join syntax in this class.

If you specify a natural join on tables that do not have any column names in common, then you get a cross join.