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.

In MySQL table aliases may be case specific depending on your system. My test system is not case specific for table names and table aliases, so if a demo does not work and your system is case specific, check for a case issue.

Demo 01: A single table select without an explicit table aliases.
You can run this query from any database since the From clause specifies the database for the table.

```
select dept_id, dept_name
from employee.departments;
+---------+-----------------+
| dept_id | dept_name       |
+---------+-----------------+
|      10 | Administration  |
|      20 | Marketing       |
|      30 | Development     |
|      35 | Cloud Computing |
|      80 | Sales           |
|      90 | Shipping        |
|      95 | Logistics       |
|     210 | IT Support      |
|     215 | IT Support      |
+---------+-----------------+
```

Demo 02: You could also use these versions of the query

```
select dp.dept_id, dp.dept_name
from employee.departments dp;

select dept_id, dept_name
from employee.departments dp;
```

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.

The use as of key word AS is optional with a table alias. (The use of AS with a table alias is not allowed with some dbms; you might want to get in the habit of skipping As for table aliases.)

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 employee.employees
INNER JOIN  employee.departments USING (dept_id)
limit 10;
+--------+----------+----------------+
| emp_id | Employee | dept_name      |
+--------+----------+----------------+
|    100 | King     | Administration |
|    201 | Harts    | Marketing      |
|    101 | Koch     | Development    |
|    108 | Green    | Development    |
|    109 | Fiet     | Development    |
|    110 | Chen     | Development    |
|    203 | Mays     | Development    |
|    204 | King     | Development    |
|    205 | Higgs    | Development    |
|    206 | Geitz    | Development    |
+--------+----------+----------------+
```

Discussion:

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

We do not need to qualify the column names in common. 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.

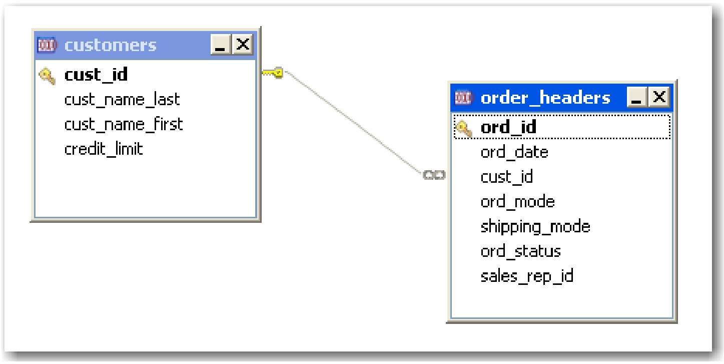
Another thing to be aware of: if we have any department with no employees, the inner join will not display that department. To get those rows we would need an outer join which we will discuss soon.

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

```
select customer_id
 , customer_name_last as "Customer"
 , order_id
from customer.customers
inner join orderEntry.orderHeaders using( customer_id )
```

```
order by customer_id
limit 10 ;
+-------------+----------+----------+
| customer_id | Customer | order_id |
+-------------+----------+----------+
|      400300 | McGold   |      378 |
|      401250 | Morse    |      106 |
|      401250 | Morse    |      113 |
|      401250 | Morse    |      119 |
|      401250 | Morse    |      301 |
|      401250 | Morse    |      506 |
|      401250 | Morse    |      552 |
|      401890 | Northrep |      112 |
|      401890 | Northrep |      519 |
|      402100 | Morise   |      114 |
+-------------+----------+----------+
```

The following is a graphic of a two table join.



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 orders table to form a virtual table, which is then joined to the order details 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.

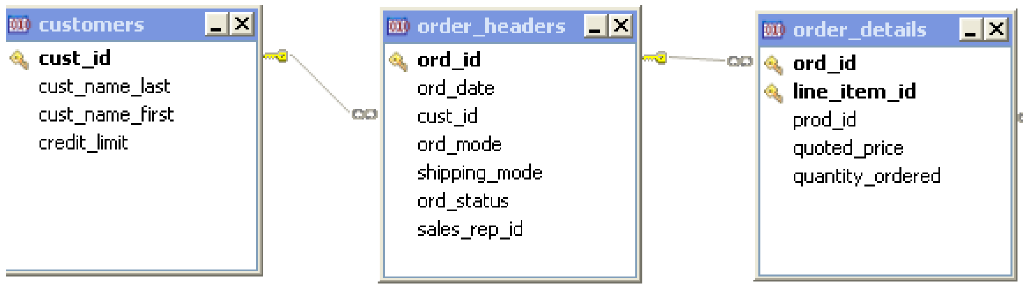
The key word Join and Inner Join both define an inner join. I will skip the work Inner in the following queries to make these easier to read. There is another type of join called an outer join we will discuss later.

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 customer_id
 , customer_name_last as "Customer"
 , order_id
 , prod_id
from customer.customers
inner join orderEntry.orderHeaders using( customer_id )
inner join orderEntry.orderDetails using( order_id )
order by customer_id
 , order_id
 limit 10 ;
```

```
+-------------+----------+----------+---------+
| customer_id | Customer | order_id | prod_id |
+-------------+----------+----------+---------+
|      400300 | McGold   |      378 |    1120 |
|      400300 | McGold   |      378 |    1125 |
|      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 |    1020 |
+-------------+----------+----------+---------+
```



Demo 06: four table join; inner join with USING

```
select customer_id
 , customer_name_last as "Customer"
 , order_id
 , prod_id
 , prod_name
from customer.customers
join orderEntry.orderHeaders using( customer_id )
join orderEntry.orderDetails using( order_id )
join product.products using( prod_id )
limit 10 ;
+-------------+----------+----------+---------+-----------------+
| customer_id | Customer | order_id | prod_id | prod_name       |
+-------------+----------+----------+---------+-----------------+
|      400300 | McGold   |      378 |    1120 | Washer          |
|      400300 | McGold   |      378 |    1125 | Dryer           |
|      401250 | Morse    |      106 |    1060 | Mountain bike   |
|      401250 | Morse    |      113 |    1080 | Cornpopper      |
|      401250 | Morse    |      119 |    1070 | Iron            |
|      401250 | Morse    |      301 |    1100 | Blender         |
|      401250 | Morse    |      552 |    2984 | B000000Y7L      |
|      401250 | Morse    |      552 |    2014 | B000005INR      |
|      401890 | Northrep |      112 |    1110 | Pancake griddle |
|      401890 | Northrep |      519 |    1020 | Dartboard       |
+-------------+----------+----------+---------+-----------------+
```

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

```
select customer_id
 , order_id
 , prod_id
 , prod_name
 , quoted_price
```

```
from customer.customers
join orderEntry.orderHeaders using( customer_id )
join orderEntry.orderDetails using( order_id )
join product.products using( prod_id )
join product.categories using( catg_id )
where catg_desc in( 'APPLIANCES' )
limit 10;
+-------------+----------+---------+-----------+--------------+
| customer_id | order_id | prod_id | prod_name | quoted_price |
+-------------+----------+---------+-----------+--------------+
|      402100 |      115 |    1120 | Washer    |       475.00 |
|      409030 |      130 |    1120 | Washer    |       500.00 |
|      903000 |      306 |    1120 | Washer    |       500.00 |
|      900300 |      307 |    1120 | Washer    |       450.00 |
|      400300 |      378 |    1120 | Washer    |       450.00 |
|      403010 |      118 |    1125 | Dryer     |       475.00 |
|      409030 |      130 |    1125 | Dryer     |       500.00 |
|      903000 |      306 |    1125 | Dryer     |       500.00 |
|      900300 |      307 |    1125 | Dryer     |       450.00 |
|      400300 |      378 |    1125 | Dryer     |       450.00 |
+-------------+----------+---------+-----------+--------------+
```

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 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.
It is not required to use table aliases -but it does make the query easier to read.

```
select order_id
 , customer_id
 , emp_id
 , name_last as "SalesRep"
from orderEntry.orderHeaders oh
join employee.employees em on oh.sales_rep_id = em.emp_id
limit 10 ;
+----------+-------------+--------+----------+
| order_id | customer_id | emp_id | SalesRep |
+----------+-------------+--------+----------+
|      112 |      401890 |    145 | Russ     |
|      130 |      409030 |    145 | Russ     |
|      312 |      903000 |    145 | Russ     |
|      405 |      408770 |    145 | Russ     |
|      505 |      403000 |    145 | Russ     |
|      540 |      404950 |    145 | Russ     |
|      105 |      403000 |    150 | Tuck     |
|      106 |      401250 |    150 | Tuck     |
|      107 |      403050 |    150 | Tuck     |
|      111 |      403000 |    150 | Tuck     |
+----------+-------------+--------+----------+
```

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 cust_id column because we are not using the column name join and cust_id appears in more than one of these tables.

```
select cs.customer_id
 , customer_name_last as "Customer"
 , order_id
from customer.customers cs
join orderEntry.orderHeaders oh on cs.customer_id = oh.customer_id
limit 10 ;
+-------------+----------+----------+
| customer_id | Customer | order_id |
+-------------+----------+----------+
|      400300 | McGold   |      378 |
|      401250 | Morse    |      106 |
|      401250 | Morse    |      113 |
|      401250 | Morse    |      119 |
|      401250 | Morse    |      301 |
|      401250 | Morse    |      506 |
|      401250 | Morse    |      552 |
|      401890 | Northrep |      112 |
|      401890 | Northrep |      519 |
|      402100 | Morise   |      114 |
+-------------+----------+----------+
```

Demo 10: The only column in the select you are required to qualify is customer_id because it occurs in two tables in the From clause. But it is common and good style to qualify all of the columns when you have a join. This is easier to read with table aliases.

```
select cs.customer_id
 , cs.customer_name_last as "Customer"
 , oh.order_id
from customer.customers cs
join orderEntry.orderHeaders oh on cs.customer_id = oh.customer_id
limit 10 ;
```

You can combine both syntaxes in the same query.

Demo 11: Inner join with a USING and an ON clause- Show customers and their orders and their sales rep.

```
select cs.customer_id
 , cs.customer_name_last as "Customer"
 , oh.order_id
 , em.emp_id
 , em.name_last as "SalesRep"
from customer.customers cs
join orderEntry.orderHeaders oh using( customer_id )
join employee.employees em on oh.sales_rep_id = em.emp_id
limit 10 ;
+-------------+----------+----------+--------+----------+
| customer_id | Customer | order_id | emp_id | SalesRep |
+-------------+----------+----------+--------+----------+
|      400300 | McGold   |      378 |    150 | Tuck     |
|      401250 | Morse    |      106 |    150 | Tuck     |
|      401250 | Morse    |      113 |    150 | Tuck     |
|      401250 | Morse    |      119 |    155 | Hiller   |
|      401250 | Morse    |      301 |    150 | Tuck     |
|      401250 | Morse    |      506 |    150 | Tuck     |
|      401250 | Morse    |      552 |    150 | Tuck     |
|      401890 | Northrep |      112 |    145 | Russ     |
|      401890 | Northrep |      519 |    155 | Hiller   |
|      402100 | Morise   |      114 |    155 | Hiller   |
+-------------+----------+----------+--------+----------+
```

Demo 12: This is the five table join including a row filter 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 customer.customers cs
join orderEntry.orderHeaders oh on cs.customer_id = oh.customer_id
join orderEntry.orderDetails od on oh.order_id = od.order_id
join product.products pr     on od.prod_id = pr.prod_id
join product.categories ct   on pr.catg_id = ct.catg_id
where ct.catg_desc in( 'APPLIANCES' )
limit 10;
```

Demo 13: This is a style of query I often get from students. This is barely readable and has a logic error. How easy is it to see the error- without reformatting the query?

```
select customer.customers.customer_id, orderEntry.orderHeaders.order_id,
orderEntry.orderDetails.prod_id,
product.products.prod_name,orderEntry.orderDetails.quoted_price
from  customer.customers  join  orderEntry.orderHeaders on
customer.customers.customer_id = orderEntry.orderHeaders.customer_id join
orderEntry.orderDetails  on orderEntry.orderDetails.order_id =
orderEntry.orderHeaders.order_id
join  product.products  on orderEntry.orderDetails.prod_id =
orderEntry.orderDetails.prod_id join product.categories  on
product.categories.catg_id  = product.products.catg_id
where   product.categories.catg_desc in( 'APPLIANCES' )
limit 10;
```

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.