

Retrieving Records with the USING Clause

```
SELECT order_id, order_status, customer_id, cust_first_name  
FROM orders JOIN customers  
USING (customer_id);
```

	ORDER_ID	ORDER_STATUS	CUSTOMER_ID	CUST_FIRST_NAME
1	2458	0	101	Constantin
2	2447	8	101	Constantin
3	2413	5	101	Constantin
4	2430	8	101	Constantin
5	2397	1	102	Harrison
6	2432	10	102	Harrison
7	2414	8	102	Harrison
8	2431	1	102	Harrison
...				
64	2448	5	145	DeSouza
65	2379	8	146	Ella

6-1

Retrieving Records with the USING Clause

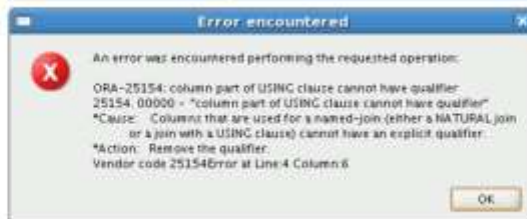
In the example in the slide, the `CUSTOMER_ID` columns in the `ORDERS` and `CUSTOMERS` tables are joined and thus the `CUST_FIRST_NAME` of the customer who placed each order is shown.

Using Table Aliases with the USING Clause

Do not qualify a column that is used in the USING clause.

If the same column is used elsewhere in the SQL statement, do not alias it.

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d
USING (location_id)
WHERE  d.location_id = 1400;
```



6-2

Using Table Aliases with the USING clause

When joining with the USING clause, you cannot qualify a column that is used in the USING clause itself. Furthermore, if that column is used anywhere in the SQL statement, you cannot alias it. For example, in the query mentioned in the slide, you should not alias the `location_id` column in the WHERE clause because the column is used in the USING clause.

The columns that are referenced in the USING clause should not have a qualifier (table name or alias) anywhere in the SQL statement. For example, the following statement is valid:

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d USING
(location_id)
WHERE  location_id = 1400;
```

The columns that are common in both the tables, but not used in the USING clause, must be prefixed with a table alias; otherwise, you get the “column ambiguously defined” error.

In the following statement, `manager_id` is present in both the `employees` and `departments` table; if `manager_id` is not prefixed with a table alias, it gives a “column ambiguously defined” error.

The following statement is valid:

```
SELECT first_name, d.department_name, d.manager_id  
FROM   employees e JOIN departments d USING  
       (department_id)  
WHERE  department_id = 50;
```

Creating Joins with the ON Clause

The join condition for the natural join is basically an equijoin of all columns with the same name.

Use the ON clause to specify arbitrary conditions or specify columns to join.

The join condition is separated from other search conditions.

The ON clause makes code easy to understand.

6-3

Creating Joins with the ON Clause

Use the ON clause to specify a join condition. With this, you can specify join conditions separate from any search or filter conditions in the WHERE clause.

Retrieving Records with the ON Clause

```
SELECT e.order_status, e.customer_id, e.order_id,  
       d.order_id, d.quantity  
FROM   orders e JOIN order_items d  
ON     (e.order_id = d.order_id);
```

	ORDER_STATUS	CUSTOMER_ID	ORDER_ID	ORDER_ID_1	QUANTITY
1	8	104	2355	2355	200
2	5	105	2356	2356	38
3	5	108	2357	2357	140
4	2	105	2358	2358	9
5	9	106	2359	2359	1
6	8	108	2361	2361	180
7	4	109	2362	2362	200
8	0	144	2363	2363	9
9	4	145	2364	2364	6

...

6-4

Retrieving Records with the ON Clause

In this example, the `ORDER_ID` columns in the `ORDERS` and `ORDER_ITEMS` table are joined using the `ON` clause. Whenever an `order_id` in the `ORDERS` table equals a `order ID` in the `ORDER_ITEMS` table, the row is returned. The table alias is necessary to qualify the matching `column_names`.

You can also use the `ON` clause to join columns that have different names. The parenthesis around the joined columns, as in the example in the slide, `(e.order_id = d.order_id)` is optional. So, even `ON e.order_id = d.order_id` will work.

Note: When you use the Execute Statement icon to run the query, SQL Developer suffixes a `'_1'` to differentiate between the two `order_ids`.

Creating Three-Way Joins with the ON Clause

```
SELECT customer_id, unit_price, warehouse_id  
FROM orders e  
JOIN order_items d  
ON e.order_id = d.order_id  
JOIN inventories f  
ON d.product_id = f.product_id;
```

	CUSTOMER_ID	UNIT_PRICE	WAREHOUSE_ID
1	100	199.1	9
2	100	199.1	2
3	100	199.1	4
4	100	199.1	6
5	100	199.1	9
6	100	226.6	9
7	100	226.6	2
8	100	226.6	4
9	100	226.6	6
10	100	226.6	9
11	100	270.6	6
12	144	199.1	9

6-5

Creating Three-Way Joins with the ON Clause

A three-way join is a join of three tables. In SQL:1999-compliant syntax, joins are performed from left to right. So, the first join to be performed is `ORDERS` `JOIN` `ORDER_ITEMS`. The first join condition can reference columns in `ORDERS` and `ORDER_ITEMS` but cannot reference columns in `INVENTORIES`. The second join condition can reference columns from all three tables.

Note: The code example in the slide can also be accomplished with the `USING` clause:

```
SELECT e.customer_id, d.unit_price, f.warehouse  
FROM orders e  
JOIN order_items d  
USING (order_id)  
JOIN inventories f  
USING (product_id)
```

Applying Additional Conditions to a Join

Use the AND clause or the WHERE clause to apply additional conditions:

```
SELECT  e.order_status, e.customer_id, e.order_id,
        d.order_id, d.quantity
FROM    orders e JOIN order_items d
ON      (e.order_id = d.order_id)
AND     e.order_status = 0;
```

Or

```
SELECT  e.order_status, e.customer_id, e.order_id,
        d.order_id, d.quantity
FROM    orders e JOIN order_items d
ON      (e.order_id = d.order_id)
WHERE   e.order_status = 0;
```

68

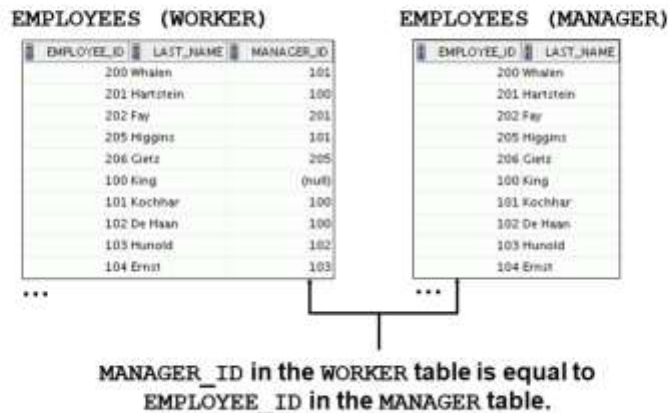
Applying Additional Conditions to a Join

You can apply additional conditions to the join.

The example shown performs a join on the `ORDERS` and `ORDER_ITEMS` tables and, in addition, displays only `ORDERS` which have an order status of 0. To add additional conditions to the `ON` clause, you can add `AND` clauses. Alternatively, you can use a `WHERE` clause to apply additional conditions.

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	174	Abel	80	80	2500
2	176	Taylor	80	80	2500

Joining a Table to Itself



6.7

Joining a Table to Itself

Sometimes you need to join a table to itself. To find the name of each employee's manager, you need to join the `EMPLOYEES` table to itself, or perform a self-join. For example, to find the name of Lorentz's manager, you need to:

Find Lorentz in the `EMPLOYEES` table by looking at the `LAST_NAME` column

Find the manager number for Lorentz by looking at the `MANAGER_ID` column.

Lorentz's manager number is 103.

Find the name of the manager with `EMPLOYEE_ID` 103 by looking at the `LAST_NAME` column. Hunold's employee number is 103, so Hunold is Lorentz's manager.

In this process, you look in the table twice. The first time you look in the table to find Lorentz in the `LAST_NAME` column and the `MANAGER_ID` value of 103.

The second time you look in the `EMPLOYEE_ID` column to find 103 and the `LAST_NAME` column to find Hunold.