Joining Tables

## Prerequisite SQL

* create table toys (
* toy\_id integer,
* toy\_name varchar2(20),
* toy\_colour varchar2(10)
* );
* create table bricks (
* brick\_id integer,
* brick\_colour varchar2(10),
* brick\_shape varchar2(10)
* );
* insert into toys values ( 1, 'Miss Snuggles', 'pink' ) ;
* insert into toys values ( 2, 'Cuteasaurus', 'blue' ) ;
* insert into toys values ( 3, 'Baby Turtle', 'green' ) ;
* insert into bricks values ( 2, 'blue', 'cube' );
* insert into bricks values ( 3, 'green', 'cube' );
* insert into bricks values ( 4, 'blue', 'pyramid' );

commit;

* Module1

## Introduction

You can combine rows from two tables with a join. This tutorial explains the join methods using these two tables:

select \* from toys;

select \* from bricks;

* Module2

## Join Syntax: Oracle vs. ANSI

Oracle Database has two syntaxes for joining tables. The proprietary Oracle method. And the ANSI standard way.

Oracle syntax joins tables in the where clause. ANSI style has a separate join clause. This tutorial will show both methods.

We recommend you use ANSI syntax. This clearly separates the join and filter clauses. This can make your query easier to read, particularly with outer joins. But which approach you use is largely a matter of personal preference.

Whatever you do, choose one style and stick with it!

* Module3

## Cross Joins

A cross join returns every row from the first table matched to every row in the second. This will always return the Cartesian product the two table's rows. I.e. the number of rows in the first table times the number in the second.

Toys and bricks both store three rows. So cross joining them returns 3 \* 3 = 9 rows.

To cross join tables using Oracle syntax, simply list the tables in the from clause:

select \*

from toys, bricks;

Using ANSI style, type cross join between the tables you want to combine:

select \*

from toys

cross join bricks;

It's rare you'll use cross joins in your code. If you need to use cross joins often it's a sign something is wrong with your data model! If you do need a cross join, ANSI syntax makes it clear this is your intent. And not that you forgot your join clause!

* Module4

## Inner Joins

An inner join (or just join) links two tables. It compares values in one or more columns from each. It only returns rows which match the join conditions in both tables.

The simplest join checks if the values in a column from one table equal the values in a column from the other.

For example, you can join toys and bricks where the toy\_id equals the brick\_id. Only the ids 2 & 3 are in both tables. And there is only one row for each value in each table. So this join returns two rows:

select \*

from toys, bricks

where toy\_id = brick\_id;

Using the ANSI way, the join criteria go in the on clause.

select \*

from toys

inner join bricks

on toy\_id = brick\_id;

An inner join can return any number of rows from zero to the Cartesian product of the two tables.

If you join on a column that has repeated values, you'll get many copies of the matching rows in the joined table. For example, there are two blue bricks. The toy Cuteasaurus is blue. So joining the tables on colour returns two copies of the row for Cuteasaurus:

select \*

from toys

join bricks

on toy\_colour = brick\_colour;

Note that the keyword inner is optional.

You can also join on inequalities. For example, to return all the rows that have different colours, write:

select \*

from toys

join bricks

on toy\_colour <> brick\_colour;

Note there are no pink bricks. So this returns a copy of the pink toy (Miss Snuggles) for each row in bricks. So Miss Snuggles appears three times.

* Module5

## Try It!

Complete the query below to:

* + Inner join toys and bricks
  + Where the toy\_id is greater than the brick\_id

select \*

from toys

join bricks

/\* TODO \*/

The query should return the following row:

**TOY\_ID TOY\_NAME TOY\_COLOUR BRICK\_ID BRICK\_COLOUR BRICK\_SHAPE**

3 Baby Turtle green 2 blue cube

* Module6

## Outer Joins

An outer join returns all the rows from one table along with the matching rows from the other. Rows without a matching entry in the outer table return null for the outer table's columns.

An outer join can either be left or right. The direction defines which side of the join the database preserves the rows for.

The following left joins toys and bricks on their IDs. Toys is on the left side of the join. So the database returns all its rows. You also get the rows from bricks which have the same id as these:

select \*

from toys

left outer join bricks

on toy\_id = brick\_id;

To outer join with Oracle syntax use the (+) operator. This goes after the columns of the table you want to optionally include. So the following is the same as the previous query:

select \*

from toys, bricks

where toy\_id = brick\_id (+);

To return all the rows from bricks, you can switch from a left join to a right. Or change the order of the tables in your query:

select \*

from toys

right join bricks

on toy\_id = brick\_id;

select \*

from bricks

left join toys

on toy\_id = brick\_id;

To do the same with Oracle syntax, move the plus to the columns of the toys table:

select \*

from toys, bricks

where toy\_id (+) = brick\_id;

An outer join will always return all the rows from the preserved table. So the number of rows returned varies from the size of the preserved table to the Cartesian product of the tables.

* Module7

## Filtering Joins

You can filter the rows returned by a join in your where clause. For example, the following joins the tables on their id. Then filters the result to only show those combinations that have a green brick:

select \*

from toys

join bricks

on toy\_id = brick\_id

where brick\_colour = 'green';

But you must take care when doing this with outer joins!

Say you want to return all the toy rows. And any brick rows with a matching id and the colour green. So you expect your results to include the three rows from toys. But only brick details for brick\_id = 3:

**TOY\_ID TOY\_NAME TOY\_COLOUR BRICK\_ID BRICK\_COLOUR BRICK\_SHAPE**

3 Baby Turtle green 3 green cube

1 Miss Snuggles pink <null> <null> <null>

2 Cuteasaurus blue <null> <null> <null>

If you filter the outer joined table in your where clause, you'll convert the query to an inner join. And only get one row:

select \*

from toys

left join bricks

on toy\_id = brick\_id

where brick\_colour = 'green';

select \*

from toys, bricks

where toy\_id (+) = brick\_id

and brick\_colour = 'green';

To fix this, you must place the filtering criteria for the outer joined table in the join clause. And in Oracle style, add (+) to all the outer joined table's columns:

select \*

from toys

left join bricks

on toy\_id = brick\_id

and brick\_colour = 'green';

select \*

from toys, bricks

where toy\_id = brick\_id (+)

and brick\_colour (+) = 'green';

* Module8

## Try It!

Complete the query below to return:

* + All the rows from bricks
  + Any rows in toys with toy\_id equal to the brick\_id and the toy\_colour is blue

select \*

from bricks

left join toys

/\* TODO \*/

The query should return the following rows:

**BRICK\_ID BRICK\_COLOUR BRICK\_SHAPE TOY\_ID TOY\_NAME TOY\_COLOUR**

2 blue cube 2 Cuteasaurus blue

3 green cube <null> <null> <null>

4 blue pyramid <null> <null> <null>

* Module9

## Full Outer Joins

Sometimes you may want to join two tables to find the matching rows. But also include any unmatched rows from both tables. I.e. a "double outer" join. This is known as a full (outer) join.

You do this using ANSI syntax like so:

select \*

from toys

full join bricks

on toy\_id = brick\_id;

Writing a full outer join in Oracle syntax is a little clumsy. You need two outer join queries. One for each table. Then to combine the results of these using union all:

select \*

from toys, bricks

where toy\_id = brick\_id (+)

union all

select \*

from toys, bricks

where toy\_id (+) = brick\_id

and toy\_id is null;

Like cross joins, it's rare to use this. But it's useful to have in your SQL toolkit!