With an inner join between tables we define how the tables are to be related to each other. With a Cartesian product we join two tables but without any attempt to relate the tables to each other. Most of the time a Cartesian product is not what you want to use, but over the semester we will find places where the Cartesian product is helpful. The Cartesian product forms the logical basis for all of the joins but you generally do not want a Cartesian product in your query.

1. The Cartesian Product or Cross Join

Assume we have the following tables and rows. The SQL is in the demo

Z_EM_Dept

D_ID	D_Name
100	Manufacturing
150	Accounting
200	Marketing
250	Research

Z_EM_Emp

E_ID	E_Name	D_ID
1	Jones	150
2	Martin	150
3	Gates	250
4	Anders	100
5	Bossy	
6	Perkins	

Z_EM_EmpProj

P_ID	E_ID
ORDB-10	3
ORDB-10	5
Q4-SALES	2
Q4-SALES	4
ORDB-10	2
Q4-SALES	5

Suppose I want to display the name of every employee who is assigned to a department and the name of that department. You should be thing that this is done with an inner join- and that would be correct.

Demo 01: Inner join.

```
select
  D.d_name, D.d_id
, E.d_id, E.e_id, E.e_name
from a_testbed.z_em_dept  D
join a_testbed.z_em_emp   E  on D.d_id = E.d_id
;
+---------------+------+------+------+--------+
| d_name        | d_id | d_id | e_id | e_name |
+---------------+------+------+------+--------+
| Accounting    |  150 |  150 |    1 | Jones  |
| Accounting    |  150 |  150 |    2 | Martin |
| Research      |  250 |  250 |    3 | Gates  |
| Manufacturing |  100 |  100 |    4 | Anders |
+---------------+------+------+------+--------+
```

The result contain the department id twice- once from the department table and once from the employee table.

Employee id 1 works in dept 150 which is the Accounting department.

But suppose you were in a hurry and you made an error. In the From clause you listed the two tables with a comma between them and skipped the On clause.

Demo 02: Getting a Cartesian product by default.

```
select
  z_em_dept.d_name, z_em_dept.d_id
, z_em_emp.d_id, z_em_emp.e_id,  z_em_emp.e_name
from a_testbed.z_em_dept
```

```
, a_testbed.z_em_emp;
+---------------+------+------+------+---------+
| d_name        | d_id | d_id | e_id | e_name  |
+---------------+------+------+------+---------+
| Manufacturing |  100 |  150 |    1 | Jones   |
| Accounting    |  150 |  150 |    1 | Jones   |
| Marketing     |  200 |  150 |    1 | Jones   |
| Research      |  250 |  150 |    1 | Jones   |
| Manufacturing |  100 |  150 |    2 | Martin  |
| Accounting    |  150 |  150 |    2 | Martin  |
| Marketing     |  200 |  150 |    2 | Martin  |
| Research      |  250 |  150 |    2 | Martin  |
| Manufacturing |  100 |  250 |    3 | Gates   |
| Accounting    |  150 |  250 |    3 | Gates   |
| Marketing     |  200 |  250 |    3 | Gates   |
| Research      |  250 |  250 |    3 | Gates   |
| Manufacturing |  100 |  150 |    4 | Anders  |
| Accounting    |  150 |  150 |    4 | Anders  |
| Marketing     |  200 |  150 |    4 | Anders  |
| Research      |  250 |  150 |    4 | Anders  |
| Manufacturing |  100 | NULL |    5 | Bossy   |
| Accounting    |  150 | NULL |    5 | Bossy   |
| Marketing     |  200 | NULL |    5 | Bossy   |
| Research      |  250 | NULL |    5 | Bossy   |
| Manufacturing |  100 | NULL |    6 | Perkins |
| Accounting    |  150 | NULL |    6 | Perkins |
| Marketing     |  200 | NULL |    6 | Perkins |
| Research      |  250 | NULL |    6 | Perkins |
+---------------+------+------+------+---------+
24 rows in set (0.00 sec)
```

You get 24 rows in the result and it looks like employee 1 works in Manufacturing and in Accounting and in Marketing and in Research. But it looks like employee 2 also works in all of those departments- In fact every employee seems to work in every department including Bossy who is assigned to no department at all.

This is a legal query- it runs without error and produces a result; but the result is not very meaningful- unless you wanted to see a table of all possible assignments of employees to all departments.

What you have here is a Cartesian product of two tables; it contains each row in the first table associated with each of the rows in the second table. There is no join on matching attribute values. Therefore, we get each of the 4 department rows matched with each of the 6 employee rows for a total of 24 rows.

Demo 03: Specifying the cross join gives the same results but makes it obvious that we are intentionally doing a Cartesian product.

```
select z_em_dept.d_name, z_em_dept.d_id
, z_em_emp.d_id,    z_em_emp.e_id, z_em_emp.e_name
from a_testbed.z_em_dept
CROSS JOIN a_testbed.z_em_emp;
```

Demo 04: If we added in the project table with no joins, then we get 144 rows returned.

```
select z_em_dept.d_name, z_em_dept.d_id
, z_em_emp.d_id, z_em_emp.e_id, z_em_emp.e_name
, p_id
from a_testbed.z_em_dept
CROSS JOIN a_testbed.z_em_emp
CROSS JOIN a_testbed.z_em_EmpProj;
```

```
+---------------+------+------+------+---------+----------+
| d_name        | d_id | d_id | e_id | e_name  | p_id     |
+---------------+------+------+------+---------+----------+
| Manufacturing |  100 |  150 |    1 | Jones   | ORDB-10  |
| Accounting    |  150 |  150 |    1 | Jones   | ORDB-10  |
| Marketing     |  200 |  150 |    1 | Jones   | ORDB-10  |
| Research      |  250 |  150 |    1 | Jones   | ORDB-10  |
| Manufacturing |  100 |  150 |    2 | Martin  | ORDB-10  |
| Accounting    |  150 |  150 |    2 | Martin  | ORDB-10  |
| Marketing     |  200 |  150 |    2 | Martin  | ORDB-10  |
| Research      |  250 |  150 |    2 | Martin  | ORDB-10  |
| Manufacturing |  100 |  250 |    3 | Gates   | ORDB-10  |
| Accounting    |  150 |  250 |    3 | Gates   | ORDB-10  |
| Marketing     |  200 |  250 |    3 | Gates   | ORDB-10  |
| Research      |  250 |  250 |    3 | Gates   | ORDB-10  |
| Manufacturing |  100 |  150 |    4 | Anders  | ORDB-10  |
| Accounting    |  150 |  150 |    4 | Anders  | ORDB-10  |
| Marketing     |  200 |  150 |    4 | Anders  | ORDB-10  |
| Research      |  250 |  150 |    4 | Anders  | ORDB-10  |
| Manufacturing |  100 | NULL |    5 | Bossy   | ORDB-10  |
| Accounting    |  150 | NULL |    5 | Bossy   | ORDB-10  |
| Marketing     |  200 | NULL |    5 | Bossy   | ORDB-10  |
| Research      |  250 | NULL |    5 | Bossy   | ORDB-10  |
. . .  rows omitted
```

144 rows would take a bit more than two pages to display- that is a lot of rows and these are tiny tables.

Suppose you did a cross join of the vets clients table (15 rows) with the animal table (32 rows) The result would have 480 rows. And every client would be associated with every animal; That is unacceptable.

Now consider adding the exam headers table (32 rows) to the From clause The result would now have 15,360 rows. Of those rows only 32 are meaningful (in the sense that they matching up a client to an animal to an exam).

The number of rows in the result of a Cartesian product is the product of the number of rows in each table.

Suppose our clinic had 300 clients with an average of 2 animals each and that there were an average of 2 exams per animal each year.

Our tables would have 300 rows for clients, 600 rows for animals and 1200 rows for exams. If we wanted to display exam header data for last year we would expect about 1200 rows in the result table. With a Cartesian product we would get 216,000,000 rows in the result

300*600*1200 = 216000000

Sometimes there is a value in intentionally doing a Cartesian product of two tables, but most of the time a query with a Cartesian product has been miscoded. If your query result has a lot more rows than you expected, check your joins carefully.

Error Alert

The following is not actually a Cartesian product but it is an error I see occasionally which also results in too many rows. Note the join in the ON phrase. The join test is d.d_id = d.d_id which is always true.

```
select
  d.d_name, d.d_id
, e.d_id, e.e_id, e.e_name
from a_testbed.z_em_dept d
join a_testbed.z_em_emp  e on D.d_id = d.d_id;
```