Combining Tables

Cartesian Product

Multiple tables in the FROM clause

There may be instances when a query requires information from two or more tables to be collated in some way, for example, to find out in which locations the employees work.

This would require the information in the EMP table to be combined with information from the DEPT table.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
405	MARCH	ADMIN	938	13/06/1997	18000		2
535	BYRNE	SALES	734	15/08/1997	26000	300	3
557	BELL	SALES	734	26/03/2000	22500	500	3
602	BIRD	MANAGER	875	31/10/1997	39750		2
690	AHMAD	SALES	734	05/12/1997	22500	1400	3
734	COX	MANAGER	875	11/06/2002	38500		3
818	POLLARD	MANAGER	875	14/05/2000	34500		1
824	REES	ANALYST	602	05/03/2000	40000		2
875	PARKER	PRESIDENT		09/07/2002	60000		1
880	TURNER	SALES	734	04/06/20			
912	HAYES	ADMIN	824	04/06/20			
936	CASSY	ADMIN	734	23/07/20			
938	GIBSON	ANALYST	602	05/12/1997	40000		2
970	BLACK	ADMIN	818	21/11/1997	23000		1

EMP table containing information about employees

DEPTNO	DNAME	LOC
1	ACCOUNTING	LONDON
2	RESEARCH	YORK
3	SALES	BIRMINGHAM
4	OPERATIONS	LEEDS

DEPT table containing information about department locations

To combine two or more tables together, SQL implements the equivalent of the mathematical operation *Cartesian Product*. In Relational Algebra the Cartesian Product of these two tables would be shown as

$$EMP \times DEPT$$

This is written in SQL by listing the tables to be connected in the *FROM* clause, separated by commas.

A multi-table FROM statement is of the form:

```
from tablename, tablename, tablename …
```

A query to combine the EMP and DEPT tables would look like this:

```
select *
from emp, dept;
```

This would produce the following results

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	emp.DEPTNO	dept.DEPTNO	DNAME	LOC
405	MARCH	ADMIN	938	13/06/1997	18000		2	1	ACCOUNTIN	LONDON
405	MARCH	ADMIN	938	13/06/1997	18000		2	2	RESEARCH	YORK
405	MARCH	ADMIN	938	13/06/1997	18000		2	3	SALES	BIRMINGHAM
405	MARCH	ADMIN	938	13/06/1997	18000		2	4	OPERATIONS	LEEDS
936	CASSY	ADMIN	734	23/07/2002	19500		3	1	ACCOUNTIN	LONDON
936	CASSY	ADMIN	734	23/07/2002	19500		3	2	RESEARCH	YORK
936	CASSY	ADMIN	734	23/07/2002	19500		3	3	SALES	BIRMINGHAM
936	CASSY	ADMIN	734	23/07/2002	19500		3	4	OPERATIONS	LEEDS
912	HAYES	ADMIN	824	04/06/2001	21000		2	1	ACCOUNTIN	LONDON
912	HAYES	ADMIN	824	04/06/2001	21000		2	2	RESEARCH	YORK
912	HAYES	ADMIN	824	04/06/2001	21000		2	3	SALES	BIRMINGHAM
912	HAYES	ADMIN	824	04/06/2001	21000		2	4	OPERATIONS	LEEDS
557	BELL	SALES	734	26/03/2000	22500	500	3	1	ACCOUNTIN	LONDON
557	BELL	SALES	734	26/03/2000	22500	500	3	2	RESEARCH	YORK

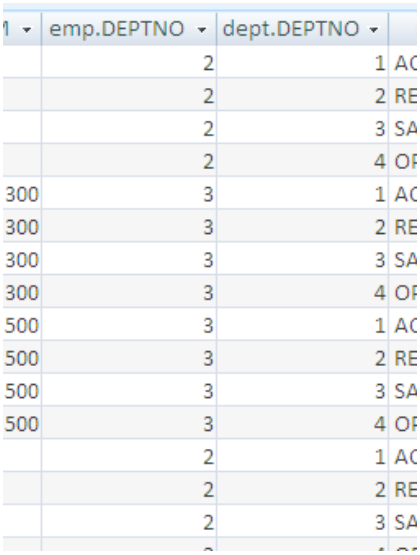
With many more records not shown.

Activity: Try this query out and verify that it performs as suggested

In fact, the Cartesian Product merges *every* row from *every* table together – the EMP table contains 14 records, the DEPT table contains 4 records, hence the final query results in 14*4 = 56 records. The database cannot assume (or deduce) how the table should be connected.

To do this an extra term must be added to the WHERE clause, specifying which two (or more) columns should contain values that are to be compared in some way (typically using equality '=').

With this example, it is clear that there are two columns that should be matched to produce the correct results – the EMP table contains a DEPTNO column, the DEPT table contains a DEPTNO. Only those rows which contain the same value should be retained.



	emp.DEPTNO	dept.DEPTNO	
	2	1	AC
	2	2	RE
	2	3	SA
	2	4	OP
300	3	1	AC
300	3	2	RE
300	3	3	SA
300	3	4	OP
500	3	1	AC
500	3	2	RE
500	3	3	SA
500	3	4	OP
	2	1	AC
	2	2	RE
	2	3	SA

A correct version of the SELECT clause for this example would look like this:

```
select *
from dept, emp
where emp.deptno=dept.deptno;
```

Notice how we have to refer to the different DEPTNO attributes using the **`tablename.columnname`** syntax. This is required as the two tables share a same named column heading DEPTNO.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	emp.DEPTNO	dept.DEPTNO	DNAME	LOC
405	MARCH	ADMIN	938	13/06/1997	18000		2	2	RESEARCH	YORK
936	CASSY	ADMIN	734	23/07/2002	19500		3	3	SALES	BIRMINGHAM
912	HAYES	ADMIN	824	04/06/2001	21000		2	2	RESEARCH	YORK
557	BELL	SALES	734	26/03/2000	22500	500	3	3	SALES	BIRMINGHAM
690	AHMAD	SALES	734	05/12/1997	22500	1400	3	3	SALES	BIRMINGHAM
970	BLACK	ADMIN	818	21/11/1997	23000		1	1	ACCOUNTIN	LONDON
880	TURNER	SALES	734	04/06/2001	25000	0	3	3	SALES	BIRMINGHAM
535	BYRNE	SALES	734	15/08/1997	26000	300	3	3	SALES	BIRMINGHAM
818	POLLARD	MANAG	875	14/05/2000	34500		1	1	ACCOUNTIN	LONDON
734	COX	MANAG	875	11/06/2002	38500		3	3	SALES	BIRMINGHAM
602	BIRD	MANAG	875	31/10/1997	39750		2	2	RESEARCH	YORK
824	REES	ANALY	602	05/03/2000	40000		2	2	RESEARCH	YORK
938	GIBSON	ANALY	602	05/12/1997	40000		2	2	RESEARCH	YORK
875	PARKER	PRESID		09/07/2002	60000		1	1	ACCOUNTIN	LONDON

Here it is clear from the two DEPTNO columns that the records have been matched in the correct sequence. In Relational Algebra this would be written as a combination of Selection and Cartesian Product:

$$\sigma_{emp.deptno = dept.deptno}\left(EMP \times DEPT\right)$$

As this operation is performed so frequently, it has its own name - a cartesian product with an appropriate selection operation is called an **Inner Join.**

Activity: Try out the refined query and verify that it produces this result.

To join *three* tables it is normally necessary to specify *two* join conditions, *four* tables usually require *three* join conditions etc.

Temporary labels may be specified in the *from* clause to save repeating the full table names repeatedly in the query:

```
select *
from emp e, dept d
where e.deptno=d.deptno;
```

Other Boolean operators to specify a further subset of the selection can be appended using *and*

```
select *
from emp e, dept d
where e.deptno=d.deptno
and d.dname="Accounting";
```

Inner Join and Cartesian Product examples

Activity: Type these example queries in and verify that they produce similar results

Examples

1 Display Employee Name, Department No and Location where Employees work

```
SELECT ename, emp.deptno, loc
from emp, dept
where emp.deptno=dept.deptno;
```

Note that the department number must be prefixed with the tablename **emp** – this is to clarify which department number is included in the output.

ename	emp.deptno	loc
MARCH	2	YORK
CASSY	3	BIRMINGHAM
HAYES	2	YORK
BELL	3	BIRMINGHAM
AHMAD	3	BIRMINGHAM
BLACK	1	LONDON
TURNER	3	BIRMINGHAM
BYRNE	3	BIRMINGHAM
POLLARD	1	LONDON
COX	3	BIRMINGHAM
BIRD	2	YORK
REES	2	YORK
GIBSON	2	YORK
PARKER	1	LONDON

In Relational Algebra:

$$\Pi_{ename,emp.deptno,loc}\left(\sigma_{emp.deptno=dept.deptno}\left(EMP \times DEPT\right)\right)$$

2 Find out where the President is based

```
SELECT ename,emp.deptno,loc
from emp, dept
where emp.deptno=dept.deptno
and job = "President";
```

ename	Deptno	loc
PARKER	1	LONDON

In Relational Algebra:

$$\Pi_{ename,emp.deptno,loc}\left(\sigma_{\substack{emp.deptno=dept.deptno \\ and\ \ job="Pr esident"}}\left(EMP \times DEPT\right)\right)$$

3 Find all the employees in the sales department

```
SELECT ename,emp.deptno,loc
from emp, dept
where emp.deptno=dept.deptno
and dname="Sales";
```

ename	deptno	loc
CASSY	3	BIRMINGHAM
BELL	3	BIRMINGHAM
AHMAD	3	BIRMINGHAM
TURNER	3	BIRMINGHAM
BYRNE	3	BIRMINGHAM
COX	3	BIRMINGHAM

In Relational Algebra:

$$\Pi_{ename,emp.deptno,loc}\left(\sigma_{\substack{emp.deptno=dept.deptno \\ and\ dname="Sales"}}\left(EMP \times DEPT\right)\right)$$

Exercises – Inner Join, Selection and Projection

In the following exercises, the query must be specified to produce the suggested result. There are spaces for you to write the SQL query and Relational Algebra (where appropriate). Use the AS command to get correct column headings in SQL.

1. Display the employee name and location of employment

SQL:

Ename	loc
MARCH	YORK
CASSY	BIRMINGHAM
HAYES	YORK
BELL	BIRMINGHAM
AHMAD	BIRMINGHAM
BLACK	LONDON
TURNER	BIRMINGHAM
BYRNE	BIRMINGHAM
POLLARD	LONDON
COX	BIRMINGHAM
BIRD	YORK
REES	YORK
GIBSON	YORK
PARKER	LONDON

In Relational Algebra:

2. Display the Employee name and salary for employees working in York

SQL:

Ename	sal
MARCH	18000
HAYES	21000
BIRD	39750
REES	40000
GIBSON	40000

In Relational Algebra:

3. Display the Employee name and job for everyone working in research

SQL:

ename	job
MARCH	ADMIN
HAYES	ADMIN
BIRD	MANAGER
REES	ANALYST
GIBSON	ANALYST

In Relational Algebra:

4. Display the Employee name and salary grade for every employee

SQL:

Ename	grade
MARCH	1
CASSY	1
HAYES	1
BELL	2
AHMAD	2
BLACK	2
TURNER	3
BYRNE	3
POLLARD	4
COX	4
BIRD	4
REES	4
GIBSON	4
PARKER	5

In Relational Algebra:

Hint: most joins are equi-joins (in other words some attribute in the first table will be matched with some attribute in the second table). This example demonstrates a different type of join that involves the use of *less than* and *greater than*.

5. Display the employee name, grade and dept name for all staff who earn more than £25000

SQL:

Ename	grade	dname
BYRNE	3	SALES
POLLARD	4	ACCOUNTING
COX	4	SALES
BIRD	4	RESEARCH
REES	4	RESEARCH
GIBSON	4	RESEARCH
PARKER	5	ACCOUNTING

In Relational Algebra:

6. Display all the employees who either work in York or Leeds

SQL:

Ename
MARCH
BIRD
REES
HAYES
GIBSON

In Relational Algebra:

Hint: In the same way that 4+5*2 is 14 (rather than 18), precedence rules apply in logic, so AND operations are performed before OR operations. To get the correct output, you may need to add brackets around certain parts of the SQL expression

7. (HARD) Display every employee along with their manager

SQL:

employee	manager
MARCH	GIBSON
CASSY	COX
HAYES	REES
BELL	COX
AHMAD	COX
BLACK	POLLARD
TURNER	COX
BYRNE	COX
POLLARD	PARKER
COX	PARKER
BIRD	PARKER
REES	BIRD
GIBSON	BIRD

In Relational Algebra:

Hint: you can use the AS reserved word to create a temporary name for a table, i.e.

`…from emp as emptable …`

Outer Joins

Inner Joins Vs Outer Joins

When combining two tables an inner join is frequently performed to combine correct rows from the first table with correct rows from the second.

In SQL this is achieved by adding a condition that matches some shared attribute in some way (e.g. `emp.deptno=dept.deptno`, or `sal>=losal and sal<=hisal`)

Unfortunately the way that a join is implemented in SQL (a cartesian product, followed by a selection) may throw away extra rows which we actually require.

Example: Get a list of the department names with location, along with the employee names who work in them. This is to include departments which do not have any staff allocated to them at present.

A first (incorrect) attempt may look like this:

```
SELECT dname, loc, ename
from dept, emp
where emp.deptno=dept.deptno
order by dname
```

But running this query does not show the employees working in Operations (as there are currently no employees in the Operations department).

dname	loc	ename
ACCOUNTING	LONDON	BLACK
ACCOUNTING	LONDON	PARKER
ACCOUNTING	LONDON	POLLARD
RESEARCH	YORK	GIBSON
RESEARCH	YORK	HAYES
RESEARCH	YORK	REES
RESEARCH	YORK	BIRD
RESEARCH	YORK	MARCH
SALES	BIRMINGHAM	CASSY
SALES	BIRMINGHAM	TURNER
SALES	BIRMINGHAM	COX
SALES	BIRMINGHAM	AHMAD
SALES	BIRMINGHAM	BELL
SALES	BIRMINGHAM	BYRNE

In fact we need to ask for all rows that match, plus any rows from the *dept* table that don't have a match (i.e. the Operations department). In Relational Algebra this is called an outer join (to include rows from the Department table that don't match rows in the Employee table). In Access SQL *left* and *right* outer joins are achieved by using the following syntax:

```
FROM table1 [ LEFT | RIGHT ] JOIN table2
ON table1.field1 operator table2.field2
```

Where table1 and table 2 are the two tables being combined with the operator using some fields table1.field1 and table2.field2. The Left Outer Join includes extra rows from the relation on the 'left' of the join, the Right Outer Join includes rows from the 'right'.

```
SELECT dname, loc, ename
from dept left join emp
on emp.deptno=dept.deptno
```

Note that this can be different in other SQL implementations – Oracle SQL uses (+) next to the table in the where clause:

```
SELECT dname, loc, ename
from dept, emp
where emp.deptno=dept.deptno (+)
```

Exercises – Outer Join

1. Get all the employees and their manager, but include those staff that don't have a manager

2. Show all employees that don't manage anyone