

“Who is the borrower whose id is 12345?”

σ borrower
borrower_id = 12345

```
select *  
  from borrower  
 where borrower_id = '12345'
```

“List the names of all borrowers”

```
 $\pi$       borrower  
last_name  
first_name
```

```
select last_name, first_name  
       from borrower
```

The rows happen to come out in alphabetic order of last name in our example database, because that happens to be the way they were inserted into the database. In SQL, we could get a different order by requesting it. Relational Algebra incorporates no such provision, since relations are sets.

```
select last_name, first_name  
       from borrower  
       order by borrower_id
```

“What is the title of the book whose call number is QA76.093?”

π σ book
title call_number = QA76.093

```
select title
      from book
     where call_number = 'QA76.093'
```

Because Relational Algebra relations are sets, the project operation can result in eliminating rows if two or more rows happen to agree on the attributes being projected. In SQL, this behavior is not the default, but can be requested explicitly using "distinct".

Compare the results of :

```
select author  
from book
```

and

```
select distinct author  
from book
```

“List the titles of all books that are currently checked out”

π
title (checked_out |X| book)

-which is equivalent to:

π σ (checked_out X book)
title checked_out.call_number =
 book.call_number

(Some SQL implementations do not provide natural join. The second Relational Algebra syntax above corresponds to the way one would do natural join on such systems)

All of the following are SQL equivalents - but not all implementations support all syntaxes:

```
select title
  from checked_out natural join book
```

or

```
select title
  from checked_out, book
 where checked_out.call_number =
        book.call_number
```

or

```
select title
  from checked_out join book on
        checked_out.call_number =
        book.call_number
```

“List the names of all borrowers having one or more books overdue”

π last_name,
first_name

σ (checked_out |X| borrower)
date_due < today

or

π last_name,
first_name

σ (checked_out X borrower)
checked_out.borrower_id =
borrower.borrower_id
^ date_due < today

```
select last_name, first_name
       from checked_out natural join borrower
       where date_due < '2002-11-12'
```

or

```
select last_name, first_name
       from checked_out, borrower
       where checked_out.borrower_id = borrower.borrower_id
             and date_due < '2002-11-12'
```

or

```
select last_name, first_name
       from checked_out join borrower
       on checked_out.borrower_id = borrower.borrower_id
       where date_due < '2002-11-12'
```

(Date selected to produce interesting results based on values in the table)

“List the names of all employees who earn more than their supervisor”

π	σ	$(\rho \text{ employee } X \rho \text{ employee})$	
e.last_name	e.supervisor_ssn =	e	s
e.first_name	s.employee_ssn ^		
	e.salary > s.salary		

(Natural join cannot be used here, because we are joining based on columns with different names in the two versions of the table)

```
select e.last_name, e.first_name
       from employee as e join employee as s
          on e.supervisor_ssn = s.ssn
       where e.salary > s.salary
```

“List the names of all people connected with the library - whether borrowers, employees, or both.”

$(\pi \text{ borrower}) \cup (\pi \text{ employee})$
last_name, last_name,
first_name first_name

(select last_name, first_name
from borrower)
union
(select last_name, first_name
from employee)

Contrast results of the above with

```
(select last_name, first_name
  from borrower)
union all
(select last_name, first_name
  from employee)
```

“List the names of all borrowers who are not employees.”

$(\pi \text{ borrower}) - (\pi \text{ employee})$
last_name, last_name,
first_name first_name

(select last_name, first_name
from borrower)
except
(select last_name, first_name
from employee)

“List all books needed as course reserves that are currently checked out to someone”

$$(\pi_{\text{reserve_book}}) \cap (\pi_{\text{checked_out}})$$

call_number call_number

```
(select call_number from reserve_book)
intersect
(select call_number from checked_out)
```

“List the names of all employees together with their supervisor’s name.”

π	σ	$(\rho \text{ employee } X \rho \text{ employee})$	
e.last_name	e.supervisor_ssn =	e	s
e.first_name	s.ssn		
s.last_name			
s.first_name			

```
select e.last_name, e.first_name,  
       s.last_name, s.first_name  
from employee as e join employee as s  
on e.supervisor_ssn = s.ssn
```

“List the call numbers of all overdue books, together with the number of days they are overdue”

π σ checked_out
call_number date_due < today
today - date_due

```
select call_number, '2002-11-12' - date_due  
from checked_out  
where date_due < '2002-11-12'
```

(Date selected to produce interesting results based on values in the table)

“What is the average salary of all employees?”

G employee

average(salary)

```
select avg(salary)
  from employee
```

“Print a list of borrower ids and the number of books each has out”

G checked_out

borrower_id count(call_number)

```
select borrower_id, count(*)  
  from checked_out  
 group by borrower_id
```

The following variant of the above is easily expressed in SQL,
though awkward in Relational Algebra:

“Print a list of borrower ids and the number of books each has out,
but only for borrowers who have at least two books out”

```
select borrower_id, count(*)  
  from checked_out  
  group by borrower_id  
  having count(*) >= 2
```

“List the titles of all books, together with the borrower id of the person (if any) who has the book out.”

```
 $\pi$       book  $\bowtie$  checked_out  
title  
borrower_id
```

```
select title, borrower_id  
       from book left outer join checked_out on  
       book.call_number=checked_out.call_number
```

Revisiting “List the names of all employees together with their supervisor’s name.” to include Aardvark in the result. (He was not included in the original query because he has no supervisor)

$$\pi_{\substack{e.\text{last_name} \\ e.\text{first_name} \\ s.\text{last_name} \\ s.\text{first_name}}} \left(\rho_{\text{employee}} \text{ e } \bowtie_{\substack{e.\text{supervisor_ssn} = \\ s.\text{ssn}}} \rho_{\text{employee}} \text{ s} \right)$$

```
select e.last_name, e.first_name,
       s.last_name, s.first_name
from employee as e left outer join
      employee as s
on e.supervisor_ssn = s.ssn
```

The following operations are not supported by many versions of SQL

- Natural join
- Division

Both can be synthesized from other operations where needed

- Assume tables A and Z have schemes (a, b, c) and (c, d, e) - with attribute c in common

then $A \bowtie B$ is

```
select a, b, A.c, d, e
      from A join Z on A.c = Z.c
```

- Division gets really messy! (but it can be done)