"Who is the borrower whose id is 12345?"

$$\sigma_{\text{borrower\_id} = 12345}\ \text{borrower}$$

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

“List the names of all borrowers”

$\pi_{\text{last\_name, first\_name}}$ borrower

```
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?”

$$\pi_{title}\ \sigma_{call\_number = QA76.093}\ book$$

```
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"

$$\pi_{title} (checked\_out \bowtie book)$$

-which is equivalent to:

$$\pi_{title} (checked\_out \, \vartheta_{checked\_out.call\_number = book.call\_number} \, book)$$

OR

$$\pi_{title} \, \sigma_{checked\_out.call\_number = book.call\_number} (checked\_out \times book)$$

(Some SQL implementations do not provide natural join. The second and third Relational Algebra syntaxes above correspond 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 join book on
            checked_out.call_number =
                          book.call_number
```

or

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

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

$$\pi_{\text{last\_name, first\_name}}\ \sigma_{\text{date\_due} < \text{today}}\ (\text{checked\_out} \bowtie \text{borrower})$$

```
select last_name, first_name
    from checked_out natural join borrower
    where date_due < current date;
```

or

```
select last_name, first_name
    from checked_out join borrower
      on checked_out.borrower_id = borrower.borrower_id
    where date_due < current date;
```

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

$$\pi_{e.last\_name,\ e.first\_name}\ \sigma_{e.salary > s.salary}\ (\rho_{e}\ employee\ \vartheta_{e.supervisor\_ssn = s.ssn}\ \rho_{s}\ employee)$$

(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."

$$(\pi_{\text{last\_name, first\_name}}\ \text{borrower}) \cup (\pi_{\text{last\_name, first\_name}}\ \text{employee})$$

```
     (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."

$$(\pi_{\text{last\_name, first\_name}}\ \text{borrower}) - (\pi_{\text{last\_name, first\_name}}\ \text{employee})$$

```
    (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”

$$(\pi_{\text{call\_number}}\ \text{reserve\_book}) \cap (\pi_{\text{call\_number}}\ \text{checked\_out})$$

```
     (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.”

$$\pi_{e.last\_name,\ e.first\_name,\ s.last\_name,\ s.first\_name}\ \sigma_{e.supervisor\_ssn = s.ssn}\ (\rho_{e}\ employee \times \rho_{s}\ employee)$$

```
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”

$$\pi_{\substack{\text{call\_number} \\ \text{today - date\_due}}} \; \sigma_{\text{date\_due < today}} \; \text{checked\_out}$$

```
select call_number, current date - date_due
     from checked_out
     where date_due < current date;
```

"What is the average salary of all employees?"

$$_{\text{average(salary)}}\mathcal{G} \quad \text{employee}$$

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

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

$$_{\text{borrower\_id}}\mathcal{G}_{\text{count(call\_number)}} \ \text{checked\_out}$$

```
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."

$\pi_{\text{title},\ \text{borrower\_id}}$ book ⟕ checked_out

```
select title, borrower_id
 from book natural left outer join checked_out
```

OR

```
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)

This is something that is easy to do in SQL, but somewhat awkward in Relational Algebra

```
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 by 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 |X| 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)