“Who is the borrower whose id is 12345?”

\[
\sigma \text{ borrower} \\
\text{borrower\_id} = 12345
\]

\[
\text{select *} \\
\text{from borrower} \\
\text{where borrower\_id} = '12345';
\]
“List the names of all borrowers”

\[
\pi \text{ borrower } \\
\quad \text{ last_name } \\
\quad \text{ first_name }
\]

\[
\text{select last_name, first_name} \\
\quad \text{ from borrower;}
\]

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.

\[
\text{select last_name, first_name} \\
\quad \text{ from borrower } \\
\quad \text{ order by borrower_id;}
\]
“What is the title of the book whose call number is QA76.093?”

\[ \pi \sigma \text{book} \]
\[ \text{title} \quad \text{call_number} = \text{QA76.093} \]

```
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:

```sql
select author
  from book;
```

and

```sql
select distinct author
  from book;
```
“List the titles of all books that are currently checked out”

\[ \pi \text{title} \ (\text{checked\_out} \ |X| \ \text{book}) \]

-which is equivalent to:

\[ \pi \text{title} \ (\text{checked\_out} \ \vartheta \ \text{book}) \]
\[ \text{checked\_out.call\_number} = \text{book.call\_number} \]

OR

\[ \pi \ \sigma \ (\text{checked\_out} \ X \ \text{book}) \]
\[ \text{checked\_out.call\_number} = \text{book.call\_number} \]

(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:

```sql
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”

\[ \pi \sigma \ (\text{checked\_out} \ |X| \text{borrower}) \]
\[ \text{last\_name, date\_due} < \text{today} \]
\[ \text{first\_name} \]

\[ \text{select} \ \text{last\_name, first\_name} \]
\[ \text{from} \ \text{checked\_out} \ \text{natural join} \ \text{borrower} \]
\[ \text{where} \ \text{date\_due} < \text{current date} \]

or

\[ \text{select} \ \text{last\_name, first\_name} \]
\[ \text{from} \ \text{checked\_out} \ \text{join} \ \text{borrower} \]
\[ \text{on} \ \text{checked\_out.borrower\_id} = \text{borrower.borrower\_id} \]
\[ \text{where} \ \text{date\_due} < \text{current date} \]
“List the names of all employees who earn more than their supervisor”

\[ \pi e.\text{last\_name}, e.\text{first\_name} \sigma (\rho \text{employee} \theta \rho \text{employee}) e.\text{salary} > s.\text{salary} \]

(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.\text{last\_name}, e.\text{first\_name} from employee as e join employee as s on e.\text{supervisor\_ssn} = s.\text{ssn}
where e.\text{salary} > s.\text{salary};
```
“List the names of all people connected with the library - whether borrowers, employees, or both.”

\[(\pi \text{ borrower}) \cup (\pi \text{ employee})\]

\begin{align*}
&\text{last\_name,} \\
&\text{first\_name}
\end{align*}

\begin{align*}
&(\text{select last\_name, first\_name} \\
&\text{from borrower}) \\
\text{union} \\
&(\text{select last\_name, first\_name} \\
&\text{from employee});
\end{align*}
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.”

\[
\text{(π borrower)} - \text{(π employee)} \\
\text{last_name,} \quad \text{last_name,} \\
\text{first_name} \quad \text{first_name}
\]

\[
\text{(select last_name, first_name} \\
\text{from borrower)} \\
\text{except} \\
\text{(select last_name, first_name} \\
\text{from employee)};
\]
“List all books needed as course reserves that are currently checked out to someone”

\[
(\pi \text{ reserve\_book}) \cap (\pi \text{ checked\_out})
\]

\[
\begin{align*}
\text{call\_number} & \quad \text{call\_number} \\
\end{align*}
\]

\[
(\text{select call\_number from reserve\_book})
\]

\[
\text{intersect}
\]

\[
(\text{select call\_number from checked\_out});
\]
“List the names of all employees together with their supervisor’s name.”

\[
\pi \sigma \quad (\rho \text{employee X } \rho \text{employee})
\]

\[
e.\text{last\_name} \quad e.\text{supervisor\_ssn} = e \quad s
\]

\[
e.\text{first\_name} \quad s.\text{ssn}
\]

\[
s.\text{last\_name}
\]

\[
s.\text{first\_name}
\]

\[
\text{select e.\text{last\_name}, e.\text{first\_name}, s.\text{last\_name}, s.\text{first\_name}}
\]

\[
\text{from employee as e join employee as s}
\]

\[
\text{on e.\text{supervisor\_ssn} = s.\text{ssn};}
\]
“List the call numbers of all overdue books, together with the number of days they are overdue”

\[ \pi \sigma \text{ checked\_out} \]
\[ \text{call\_number} \quad \text{date\_due} < \text{today} \]
\[ \text{today} - \text{date\_due} \]

\[ \text{select call\_number, current date - date\_due} \]
\[ \text{from checked\_out} \]
\[ \text{where date\_due} < \text{current date}; \]
“What is the average salary of all employees?”

\[ G \text{ employee} \]

\[ \text{average(salary)} \]

\[ \text{select avg(salary)} \]
\[ \text{from employee;} \]
“Print a list of borrower ids and the number of books each has out”

\[
G \text{ checked\_out}
\]

\[
\begin{align*}
\text{borrower\_id} & \quad \text{count(call\_number)} \\
\end{align*}
\]

```
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”

```sql
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.”

\[
\pi \quad \text{book } \sqcap \exists! \text{X} \quad \text{checked_out}
\]
\[
\phantom{\pi} \text{title}
\phantom{\pi} \text{borrower_id}
\]

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 \(|\times|\) 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)