Which students attend which lecture? Output student names and lecture titles.

```
select name, title
from students, attends, lectures
where students.reg-id = attends.reg-id and
     attends.id = lectures.id
```

alternative formulation using **tuple variables** that are associated to relations:

```
select s.name, l.title
from students as s, attends as a, lectures as l
where s.reg-id = a.reg-id and
     a.id = l.id
```

- relationship to the tuple relational calculus observable: a variable is bound to
tuples of a relation

Determine the names of all university employees, i.e., the names of all professors and all assistants.

```
(select name
from assistants)
union
(select name
from professors)
```
Find all professors that are not involved in teaching.

```sql
select name
from professors
where pers-id not in (select held_by from lectures)
```

- operator `in` tests for set membership

Find the students with the largest number of semesters.

```sql
select name
from students
where sem >= all (select sem from students)
```

- With the keyword `all` a comparison is performed with all elements of the set that is parameter of `all`.
- An arbitrary comparison operator can be used with `all` and `some`.
  - `all` does not have the functionality of an forall quantifier, since only a comparison of a value with a set can be expressed.
- rather similarity to an aggregate function
- A condition with `some` is satisfied if it is satisfied for at least one element of the set.
**Existential quantifier \texttt{exists}**

- This operator checks whether a set of tuples specified by a subquery is empty. For a non-empty set the \texttt{exists} operator yields \textit{true}, otherwise \textit{false}. For the operator \texttt{not exists} it is just vice versa.

- Operator \texttt{exists} corresponds to the existential quantifier of the relational calculus.

- example: Which professors do not hold lectures?

  ```sql
  select name 
  from professors 
  where not exists (select * from lectures where held_by = pers-id)
  ```
The renaming operator

- application of the **as**-clause
- use:
  - Two relations in the **from** clause have attributes with the same name, which would appear in the result relation without renaming.
  - If an arithmetic expression is used in the **select** clause, the result attribute does not have a name.
  - explicit change of an attribute name
example: Which credit line do the customers still have?

```sql
select name, credit-line – debit as rest
from customers
```

**Tuple variables**

- A tuple variable in SQL is bound to a relation.
- Tuple variables are defined in the `from` clause by means of the `as` clause.
- in particular useful in order to compare two tuples of the same relation
- example: Which students attend which lectures?

```sql
select s.name, v.title
from students as s, attend as a, lectures as l
where s.reg-id = a.reg-id and a.id = l.id
```

**String operations**

- search patterns are described by
  - a percent sign (%): this represents any substring
  - an underscore (_): this represents any character
- distinction between upper and lower case
- String patterns in SQL are expressed with the aid of the `like` operator.
- example: Find all students with names Meier, Maier, Meyer, etc.
  ```sql
  select reg-id
  from students
  where name like "M__er"
  ```

Set operations

- schema compliant relations as operands
- `union, except, intersect`: operands and results are sets of tuples, elimination of duplicates
- `union all, except all, intersect all`: operands and results are multi-relations
- number of duplicates for multi-relations $R$ and $S$ ($F(R, x)$ describes the frequency of tuple $x$ in table $R$):
  - $\forall x : F(R \text{ union all } S, x) = F(R, x) + F(S, x)$
  - $\forall x : F(R \text{ except all } S, x) = \text{if } F(R, x) \geq F(S, x) \text{ then } F(R, x) - F(S, x) \text{ else } 0$
  - $\forall x : F(R \text{ intersect all } S, x) = \min(F(R, x), F(S, x))$