```
select dept_name from department
where dept_id not in
(select dept_id from employee where salary > 3000);
```

The first query can be replicated by doing a (theta) join operation:

$$\texttt{select } employee.name,\ department.dept_name$$
$$\texttt{from } department, employee$$
$$\texttt{where} department.dept\_id = employee.dept\_id \texttt{ and } salary > 2000;$$

- Also enumerated sets:

```
select name, salary from employee
where title in ('CEO', 'Web Developer');
```

Set comparison

For the query "Employees with salary more than at least one other employee", one possiblity is:

$$\texttt{select distinct } T.name, T.salary$$
$$\texttt{from } employee \texttt{ as } T, employee \texttt{ as } S$$
$$\texttt{where } T.salary > S.salary$$

Another way is by set comparison:

$$\texttt{select } name, salary$$
$$\texttt{from } employee$$
$$\texttt{where } salary > \texttt{ some } (\texttt{select } salary$$
$$\texttt{from } employee);$$

Here, `some` implies *at least one*.

Notice that Kyle is not included as comparison with null value leads to an unknown.
Also, the sub-statement can only result in single attribute tuples so that salary can be compared with them.
Also: `< some`, `<= some`, `>= some`, `= some`, `<> some`.
Similarly we have variations for `all`. For example, the query "Employees who make more than all web developers" can be expressed in SQL as:

$$\texttt{select } name, salary$$
$$\texttt{from } employee$$
$$\texttt{where } salary > \texttt{ all } (\texttt{select } salary$$
$$\texttt{from } employee$$
$$\texttt{where } title = \texttt{'Web Developer'});$$

Another use of `all`: "The title whose employees make the most salary on an average"

```
select avg(salary), title
from employee
group by title having avg(salary) >= all (select avg(salary)
from employee group by title);
```

Views

- till now: logical model level
- Often we need to create a different *view* of data:
  - security
  - personalized relations that are more intuitive than the logical model

$$\texttt{create view } v \texttt{ as } < query\ expression >$$

Example:

```
create view CEOs as
select *
from employee
where title = 'CEO';
```

  - The view now acts as any other relation
  - Can query it, describe it, or drop it.
    - Can appear in any place a relation would if there is no update applied to it

```
create view expensive_employees as
select name, salary, dept_name
from employee, department
where employee.dept_id = department.dept_id and salary> 2500;
```

Can specify attribute names explicitly:

```
create view department_spending(dept_name, spending) as
select dept_name, sum(salary)
from employee, department
where employee.dept_id = department.dept_id group bydept_name;
```

- Views are computed whenever they are needed rather than when they are first created alone.

- When first created only the defn. of the view is stored.
- When the view appears in another query, it is evaluated then

- Some DBs actually store the content of a view at creation time, called *materialized views*, and *view maintenance* has to be done.
  - allows fast response for frequently used views
  - however issues of storage costs, added overhead for updates

  Views defined by other views.
  - For example,

    `create view` Top_CEOs `as`
    `select` $*$
    `from` $CEOs$
    `where` $salary > 2500$;
  - Can expand views

    `repeat`
    Find any view v_i in e_i
    Replace the view $v_i$by its expression
    `until` no more view relations are present in e_i

Modification

Deletion

- Can only delete a tuple/tuples. Cannot delete values for particular attibutes

  `delete from` r
  `where` P

  Deletes all tuples t in r for which $P(t)$ is true.
- `delete` operates on **one relation only**.
- `delete from r` deletes all tuples (should get a warning; not in MySQL!!).
- delete employees with `null` salary

  `delete from` *employee*
  `where` *salary* `is null`;
- delete employees from department 'Programming'

  `delete from` *employee*
  `where` *name* `in` (`select` *name*
  `from` *employee*, *department*
  `where` *employee.dept_id* = *department.dept_id* `and` *department.dept_name* = 'Programming';