

## Null Values

```
select name
from employee
where salary is null;
also, is not null
```

Some rules for null values:

- think of *unknown* as a weaker *false*
- arithmetic ops (+, -, \*, /) give an unknown
- if there is null in boolean operation in the *predicate*, (i.e., one operand of **and**, **or**, **not**) the rules are:
  - **and**. *true and unknown = unknown, false and unknown = false, unknown and unknown = unknown*
  - **or**. *true or unknown = true, false or unknown = unknown, unknown or unknown = unknown*
  - **not**. *not of unknown is unknown.*
- For the select command, only those tuples in  $r_1 \times r_2 \times \dots \times r_n$  are in the result for which the predicate *P* evaluates to **true**. If **unknown** or **false**, then the tuple wouldn't appear in the result.
  - So, the query: **select \* from employee where salary < 2000;** will not show a tuple with null salary.
  - A difference from regular rule for null in arithmetic operations is in *aggregate* functions. So, **select sum(salary) from employee** will ignore tuples with null salary values.

## Adding foreign key

You can add a foreign key for a table attribute *after* the table has been created:

```
alter table employee
add foreign key (dept_id) references department;
```

## Nested subqueries

- a **select-from-where** expression inside a query.
  - For e.g.: departments with an employee who makes more than 3000:

```
select dept_name from department
where dept_id in
(select dept_id from employee where salary > 3000);
```

or

```
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:

```
select employee.name, department.dept_name
      from department, employee
where department.dept_id = employee.dept_id 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 possibility is:

```
select distinct T.name, T.salary
      from employee as T, employee as S
      where T.salary > S.salary
```

Another way is by set comparison:

```
select name, salary
      from employee
where salary > some (select salary
                     from employee);
```

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

Notice that Kyle is not included as comparison with null value leads to an unknown.

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:

```
select name, salary
      from employee
where salary > all (select salary
                   from employee
                   where title = 'WebDeveloper');
```

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);
```