Nested Queries

- In the atomic conditions of the **`where`** clause one can also use a **`select`** clause (which must appear in parentheses).

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16:  SELECT  E.Fname, E.Lname
      FROM    EMPLOYEE AS E
      WHERE   E.Ssn IN ( SELECT  D.Essn
                         FROM    DEPENDENT AS D
                         WHERE   E.Fname = D.Dependent_name
                                 AND E.Sex = D.Sex );
```

Nested Queries

In particular, in atomic conditions one can have:

- comparisons of an attribute (or several attributes) with the result of a subquery

- existential quantification **existential quantifier** (∃).

  an existential quantifier = condition F is TRUE if there exists some tuple that makes F TRUE.

Nested Queries (Example)

"Name and income of Frank's father"

```
select  f.name, f.income
from    person f, fatherChild fc
where   f.name = fc.father and fc.child = 'Frank'
```

```
select  f.name, f.income
from    person f
where   f.name = (select fc.father
                  from   fatherChild fc
                  where  fc.child = 'Frank')
```

An example of the first usage scenario:

the Where clause comparing f.name with the result of the nested SELECT

Nested Queries: Operators

In the **`where`** clause, the result of a nested query can be related to other values by way of several **operators**:

- equality and other comparisons such as >, < …
  (the result of the nested query must be unique)
- if it is not certain that the result of the nested query is unique, the nested query can be preceded by one of the keywords:
  - ***`any`***: *true, if the comparison is true for **at least one** of the result tuples of the nested query (e.g., > any, < any)*
  - ***`all`***: *true, if the comparison is true for **all** the result tuples of the nested query (e.g., >all, <all)*
- the operator **`in`**, which is equivalent to **`=any`**
- the operator **`not in`**, which is equivalent to **`<>all`**
- the operator **`exists`**

Nested Queries: Example

"Name and income of the fathers of persons who earn more than 20k"

```
select distinct f.name, f.income
from   person f, fatherChild fc, person c
where  f.name = fc.father and
       fc.child = c.name and c.income > 20
```

```
select f.name, f.income
from   person f
where  f.name = any
            (select fc.father
             from   fatherChild fc, person c
             where  fc.child = c.name and
                    c.income > 20)
```

Nested Queries: Example

"Name and income of the fathers of persons who earn more than 20k"

```
select f.name, f.income
from   person f
where  f.name = any
           (select fc.father
            from   fatherChild fc, person c
            where  fc.child = c.name and
                   c.income > 20)
```

```
 name  | income
-------+--------
 Greg  |     40
 Frank |     20
(2 rows)
```

```
nuttdb=# select f.name, f.income from person f;
 name  | income
-------+--------
 Andy  |     21
 Rob   |     15
 Mary  |     42
 Anne  |     35
 Phil  |     30
 Greg  |     40
 Frank |     20
 Kim   |     41
 Mike  |     21
 Lisa  |     87
(10 rows)
```

```
nuttdb=# select fc.father from fatherchild fc, person c
nuttdb-# where fc.child = c.name and c.income > 20;
 father
--------
 Greg
 Greg
 Frank
(3 rows)
```

Nested Queries: Example

Name and income of the fathers of persons who earn more than 20k.

```
select f.name, f.income
from   person f
where  f.name in (select fc.father
                  from   fatherChild fc, person c
                  where  fc.child = c.name
                         and c.income > 20)
```

"in" equals to "= any"

```
select f.name, f.income
from   person f
where  f.name in (select fc.father
                  from  fatherChild fc
                  where fc child in (select c.name
                                     from   person c
                                     where c.income > 20)
                  )
```

fathers of persons who earn more than 20k

persons who earn more than 20k

Nested Queries: Comments

The nested formulation of a query is sometimes executed less efficiently than an equivalent unnested formulation (due to limitations of the query optimizer).

The nested formulation is sometimes more *readable*.

Nested Queries: Example with `all`

"Persons who have an income that is higher than the income of all persons younger than 30"

Nested Queries: Example with `all`

"Persons who have an income that is higher than the income of all persons younger than 30"

```
select name
from   person
where  income >= all (select income
                      from   person
                      where  age < 30)
```

```
 name
------
 Mary
 Anne
 Phil
 Greg
 Kim
 Lisa
(6 rows)
```

```
nuttdb=# select name, income, age from person;
 name  | income | age
-------+--------+-----
 Andy  |     21 |  27
 Rob   |     15 |  25
 Mary  |     42 |  55
 Anne  |     35 |  50
 Phil  |     30 |  26
 Greg  |     40 |  50
 Frank |     20 |  60
 Kim   |     41 |  30
 Mike  |     21 |  85
 Lisa  |     87 |  75
(10 rows)
```

```
nuttdb=# select income from person where age < 30;
 income
--------
     21
     15
     30
(3 rows)
```

Equivalent Formulation with `max`

"Persons who have an income that is higher than the income of all persons younger than 30"

```
 name
------
 Mary
 Anne
 Phil
 Greg
 Kim
 Lisa
(6 rows)
```

```
select name
from   person
where  income >= (select max(income)
                  from   person
                  where  age < 30)
```

```
nuttdb=# select name, income, age from person;
  name   | income | age
---------+--------+-----
  Andy   |     21 |  27
  Rob    |     15 |  25
  Mary   |     42 |  55
  Anne   |     35 |  50
  Phil   |     30 |  26
  Greg   |     40 |  50
  Frank  |     20 |  60
  Kim    |     41 |  30
  Mike   |     21 |  85
  Lisa   |     87 |  75
(10 rows)
```

```
nuttdb=# select max(income)
nuttdb-# from person
nuttdb-# where age < 30
nuttdb-# ;
 max
-----
  30
(1 row)
```

Nested Queries: Example with `exists`

An expression with the operator **`exists`** is true if the result of the subquery is **not empty**.

***Example*:** "Persons with at least one child"

```
select p.name, p.age, p.income
from   person p
where  exists (select *
               from  fatherChild fc
               where fc.father = p.name)
       or
       exists (select *
               from  motherChild mc
               where mc.mother = p.name)
```

*Note: the attribute **`name`** refers to the table in the outer **`from`** clause.*

Nesting, Union, and "or"

The query for "persons with at least one child" can also be expressed as a union:

```
select p.name, p.age, p.income
from   person p, fatherChild fc
where  fc.father = p.name
union
select p.name, p.age, p.income
from   person p, motherChild mc
where  mc.mother = p.name
```

Does the following query with "or" return the same answers?

```
select distinct p.name, p.age, p.income
from   person p, fatherChild fc, motherChild mc
where  fc.father = p.name
         or mc.mother = p.name
```

Nested Queries and Negation

All the queries with nesting in the previous examples are equivalent to some unnested query. So, what's the point of nesting?

***Example*: "Persons without a child"**

```
select *
from   person p
where  not exists (select *
                   from  fatherChild fc
                   where fc.father = p.name)
       and
       not exists (select *
                   from  motherChild mc
                   where mc.mother = p.name)
```

This cannot be expressed equivalently as a "`select from where`" query … (join? union?)

Query 8 – nested queries

"Name and age of the mothers all of whose children are at least 18"

Approach 1: Subquery with **`all`**

Approach 2: Subquery with **`min`**

Approach 3: Subquery with **`not exists`**

Query 8 – Solution with `all`

"Name and age of the mothers all of whose children are at least 18"

```
select m.name, m.age
from   person m join motherChild mc
                on m.name = mc.mother
where  18 =< all (select c0.age
                  from   motherChild mc0 join person c0
                                  on mc0.child = c0.name
                  where  mc0.mother = mc.mother)
```

Query 8: Solution with `min`

"Name and age of the mothers all of whose children are at least 18"

```
select m.name, m.age
from   person m join motherChild mc
                  on m.name = mc.mother
where  18 =< (select min(c0.age)
                from   motherChild mc0 join person c0
                                     on mc0.child = c0.name
                where  mc0.mother = mc.mother)
```

"Name and age of mothers where the minimal age of their children is greater or equal 18"

Query 8: Solution with `not exists`

"Name and age of the mothers all of whose children are at least 18"

```
select m.name, m.age
from   person m join motherChild mc
                  on m.name = mc.mother
where  not exists
              (select *
               from   motherChild mc0 join person c0
                             on mc0.child = c0.name
               where  mc0.mother = mc.mother and
                      c0.age < 18)
```

Name and age of mothers who don't have a child that is younger than 18.

Nested Queries: Comments

Visibility rules:

- it is not possible to refer to a variable defined in a block below the current block
- if an attribute name is not qualified with a variable or table name, it is assumed that it refers to the “closest” variable or table with that attribute

In each block, one can refer to variables defined in the same block or in surrounding blocks

Semantics: the inner query is executed for every tuple of the outer query

Nested Queries: Visibility

Persons having at least one child.

```
select *
from   person
where  exists (select *
               from  fatherChild
               where father = name)
       or
       exists (select *
               from  motherChild
               where mother = name)
```

```
 name   | age | income
--------+-----+--------
```

```
 father | child
--------+-------
```

```
 mother | child
--------+-------
```

The attribute `name` refers to the table `person` in the outer `from` clause.

More on Visibility

Note: This query is incorrect:

```
select *
from  employee
where dept in (select name
               from  department D1
               where name = 'Production')
      or
      dept in (select name
               from  department D2
               where D2.city = D1.city)
```

employee	name	lastName	dept

department	name	address	city

Visibility: Variables in Internal Blocks

Name and income of the fathers of persons who earn more than 20k, **showing also the income of the child**.

```
select distinct f.name, f.income, c.income
from   person f, fatherChild, person c
where  f.name = fc.father and fc.child = c.name
       and c.income > 20
```

In this case, the “intuitive” nested query is incorrect:

```
select name, income, c.income
from   person
where  name in (select father
                from fatherChild
                where child in (select name
                                from person c
                                where c.income > 20))
```

Correlated Subqueries

It may be necessary to **use in inner blocks** variables that are **defined in outer blocks**. In this case one talks about **correlated** subqueries.

Example: The fathers all of whose children earn strictly more than 20k.

```
select distinct fc.father
from   fatherChild fc
where  not exists (select *
                   from  fatherChild fc0, person c
                   where fc.father = fc0.father
                         and fc0.child = c.name
                         and c.income <= 20)
```

Query 10. Correlated Subqueries

"Name and age of mothers who have a child whose age differs less than 20 years from their own age"

```
select m.name, m.age
from   person m, motherChild mc
where  m.name = mc.mother and
       mc.child in (select c.name
                    from   person c
                    where  m.age - c.age < 20)
```

Question: Intersection

Can one express intersection by way of nesting?

```
select name from employee
  intersection
select lastName as name from employee
```

employee	name	lastName	dept

Intersection by Way of Nesting

```
select name from employee
  intersection
select lastName as name from employee
```

```
select name
from  employee
where name in (select lastName
               from   employee)
```

```
select name
from  employee e
where exists (select *
              from   employee
              where  lastName = e.name)
```

Intersection Without Nesting

Is it possible to express intersection without nesting?

```
select name from employee
  intersection
select lastName as name from employee
```

```
select en.name
from   employee en, employee eln
where  en.name = eln.lastName
```

Query 11

Can one express set difference by way of nesting?

```
select name from employee
  except
select lastName as name from employee
```

Query 11 (Solution 1)

Can one express set difference by way of nesting?

```
select name from employee
  except
select lastName as name from employee
```

```
select name
from  employee
where name not in (select lastName
                   from employee)
```

Query 11 - (Solution 2)

Can one express set difference by way of nesting?

```
select name from employee
  except
select lastName as name from employee
```

```
select name
from   employee e
where  not exists (select *
                   from  employee
                   where lastName = e.name)
```

Query 12: Nesting and Functions

"The person (or the persons) that *have the highest income*"

```
select *
from  person
where income = (select max(income)
                from   person)
```

Or:

```
select *
from  person
where income >= all (select income
                     from   person)
```

Conditions on Several Attributes

The persons which have a unique combination of age and income

(that is, persons for whom the pair (age, income) is different from the corresponding pairs of all other persons).

```
select *
from  person p
where (age,income) not in
                        (select age, income
                         from person
                         where name <> p.name)
```

SQL Views

A view is a table **whose instance is derived from other tables by a query.**

`create view` *ViewName* [**`(`***AttributeList***`)`**] **`as`** *SQLSelect*

Views are virtual tables: their instances (or parts of them) are only calculated when they are used (for instance in other queries).

Example:

```
create view AdminEmp(empNo,firstName,lastName,sal) as
  select EmpNo, firstName, lastName, salary
  from   employee
  where  dept = 'Administration' and
         salary > 10
```

Maximizing Aggregates

"Which age group has the highest total income?"

One solution is to use nesting in the **`having`** clause:

```
select age
from   person
group by age
having sum(income) >= all (select sum(income)
                           from   person
                           group by age)
```

Another solution is to create a view.

Solution with Views

```
create view ageIncome(age,sumIncome) as
    select age, sum(income)
    from   person
    group by age
```

```
nuttdb=# create view ageIncome(age, sumIncome) as
nuttdb-# select age, sum(income) from person
nuttdb-# group by age;
CREATE VIEW
nuttdb=# select * from ageIncome;
 age | sumincome
-----+-----------
  60 |        20
  26 |        30
  85 |        21
  30 |        41
  50 |        75
  75 |        87
  25 |        15
  55 |        42
  27 |        21
(9 rows)
```

```
select age
from   ageIncome
where  sumIncome = (select max(sumIncome)
                    from   ageIncome)
```

```
nuttdb=# select age from ageIncome
nuttdb-# where sumIncome = (select max(sumIncome) from ageIncome);
 age
-----
  75
(1 row)
```

Query 13

Among all companies based in George Street that sell red parts, which is the one with the least average price for red parts?

On the supplier and parts DB:

```
Supplier(sid, sname, address)
Part(pid, pname, colour)
Catalog(sid, pid, cost)
```

Query 13 (Solution)

Among all companies based in George Street that supply red parts, which is the one with the least average price for red parts?

```
create view RedPartCompGS(sid,name,avgCost) as
  select sid, name, avg(cost)
  from   supplier natural join catalog
                  natural join part
  where  address LIKE '%George St%' AND
         colour = 'red'
  group by sid, name
```

Query 13 (Solution, cntd)

Among all companies based in George Street that sell red parts, which is the one with *the least average price* for red parts?

```
select name
from   RedPartCompGS
where  avgCost = (select min(avgCost)
                  from   RedPartCompGS)
```

Views can be used in subqueries

```
select *
from   person
where  name in (select father from fatherChild);
```

With a view

```
create view father(name) as
select distinct father from fatherChild;

select *
from   person
where  name in (select name from father);
```

Inline Views: Views in the FROM Clause

An equivalent formulation (… showing a view appearing in JOIN)

```
select person.*
from   person, father
where  person.name = father.name;
```

where father is the view we saw previously.

If we need a view only once, we can define it in the FROM clause

```
select *
from   person,
       (select distinct father as name
        from   fatherChild) father
where  person.name = father.name;
```

Inline Views (Cntd)

Inline views can also take part in joins

```
select person.*
from   person
       natural join
       (select distinct father as name
        from   fatherChild) father;
```

Note: The inline view needs to be named, even if the name is never used.

Exercises …

Consider a database about suppliers and parts with the following schema:

```
Supplier(sid, sname, address)
Part(pid, pname, colour)
Catalog(sid, pid, cost)
```

Formulate the following queries in SQL:

Queries: Exercises (cntd)

1. Find the names of suppliers who supply some red part.
2. Find the IDs of suppliers who supply some red or green part
3. Find the Ids of suppliers who supply some red part and are based at 21 George Street
4. Find pairs of IDs such that for some part the supplier with the first ID charges more than the supplier with the second ID.
5. For each supplier, return the maximal and the average cost of the parts they offer.
6. List those red parts that on average cost no more than 30 Euro.
7. List the names of those red parts that are offered by at least three suppliers.
8. Suppliers that supply *only* red parts
9. Suppliers that supply *all* red parts