COMP9311: Database Systems

Term 3 2022

Week 4 (SQL)

By Helen Paik, CSE UNSW

Textbook: Chapters 6 and 7

Disclaimer: the course materials are sourced from

- previous offerings of COMP9311 and COMP3311
- Prof. Werner Nutt on Introduction to Database Systems (http://www.inf.unibz.it/~nutt/Teaching/IDBs1011/)

MotherChild

mother	child
Lisa	Mary
Lisa	Greg
Anne	Kim
Anne	Phil
Mary	Andy
Mary	Rob

FatherChild

father	child
Steve	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob

Person

name	age	income
Andy	27	21
Rob	25	15
Mary	55	42
Anne	50	35
Phil	26	30
Greg	50	40
Frank	60	20
Kim	30	41
Mike	85	35
Lisa	75	87

Selection and Projection

Name and income of persons that are less than 30:

$$\pi_{\text{name, income}}(\sigma_{\text{age<30}}(\text{Person}))$$

```
select name, income
from   person
where  age < 30
```

name	income
Andy	21
Rob	15
Phil	30

Naming Conventions and Renaming

To avoid ambiguities, every attribute name has two components: *RelationName.AttributeName*

When there is no ambiguity, one can drop the initial component: *RelationName*.

```
select person.name, person.income
from   person
where  person.age < 30

can be written as:

select name, income
from   person
where  age < 30
```

and also for (re-naming attributes and relations)

```
select p.name as Pname,  p.income as income
from   person p
where  p.age < 30
```

Query 1

“From the table `person`, compute a new table by selecting only the persons with an income between 20 and 30, and adding an attribute called `income doubled’ that has, for every tuple, double the value of `income`.

Show the result of the query”

Person

name	age	income

Query 1: Solution

```
select name, age, income,
       (income * 2) as income-doubled
from   person
where  income >= 20 and income <= 30
```

name	age	income	income-doubled
Andy	27	21	42
Phil	26	30	60
Frank	60	20	40

Expressions in the Target List

```
select income/4 as quarterlyIncome
from   person
where  name = ‘Greg'
```

Complex Conditions in the “where” Clause

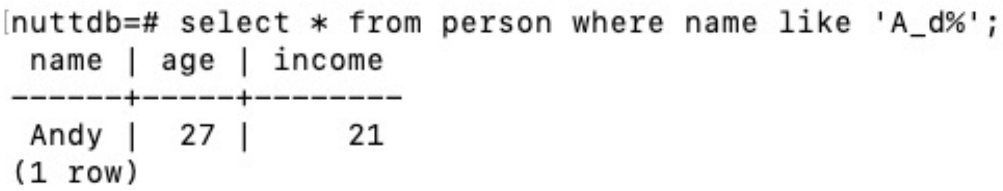
```
select *
from   person
where  income > 25
       and (age < 30 or age > 60)
```

The “like” Condition

The persons having a name that starts with '**A**' and has a '**d**' as the third letter:

```
select *
from   person
where  name like 'A_d%‘
```

- ‘_‘ matches a single letter
- ‘%‘ matches a string



```
[nuttdb=# select * from person where name like 'A_d%';
 name | age | income
------+-----+--------
 Andy |  27 |     21
(1 row)
```

Query 2

“From the table **`employee`**, calculate a new table by selecting only employees from the branches whose name start with ‘L’ and salary is less than 50, projecting the data on the attribute **`empNo`**, **`salary,  branch`** and adding an attribute that has, for every tuple, twice the value of the attribute **`salary`**.

Show the result of the query on the following table”

Employee

empNo	surname	branch	salary
7309	Black	York	55
5998	Black	Glasgow	64
9553	Brown	London	44
5698	Brown	London	64

Query 2

```
select empNo, branch, salary,
       salary*2 as doubleSal
from   employee
where  branch like ‘L%’
         and salary < 50
```

Employee

empNo	branch	salary	doubleSal
9553	London	44	88

Selection, Projection, and Join

Using `select` statements with a single relation in the `from` clause we can realise:

- selections,
- projections,
- renamings

Joins (and Cartesian products) are realised by using two or more relations in the `from` clause

SQL and Relational Algebra (cntd)

Given the relations: R1(A1,A2) and R2 (A3,A4),

```
select R1.A1, R2.A4
from   R1, R2
where  R1.A2 = R2.A3
```

corresponds to :

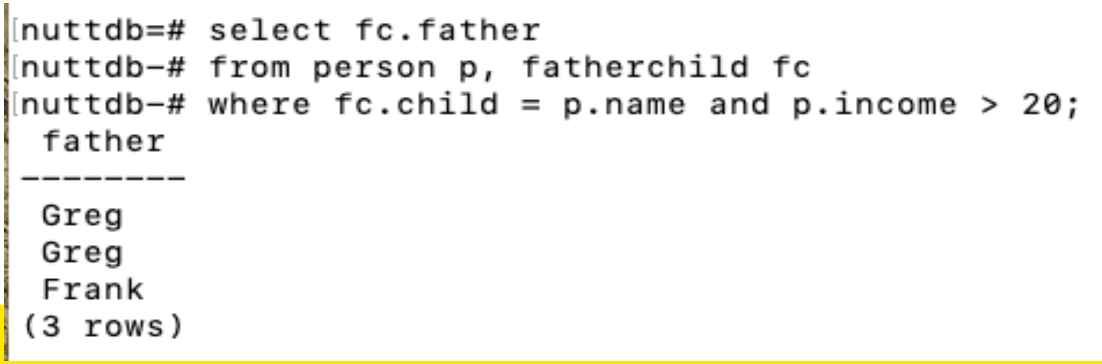
$$\pi_{A1,A4}\,(\sigma_{A2=A3}\,(R1\ \ x\ \ R2))$$

Query 3:

"The fathers of persons who earn more than 20K"

$\pi_{father}(FatherChild \bowtie_{child=name} \sigma_{income>20}(Person))$

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



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

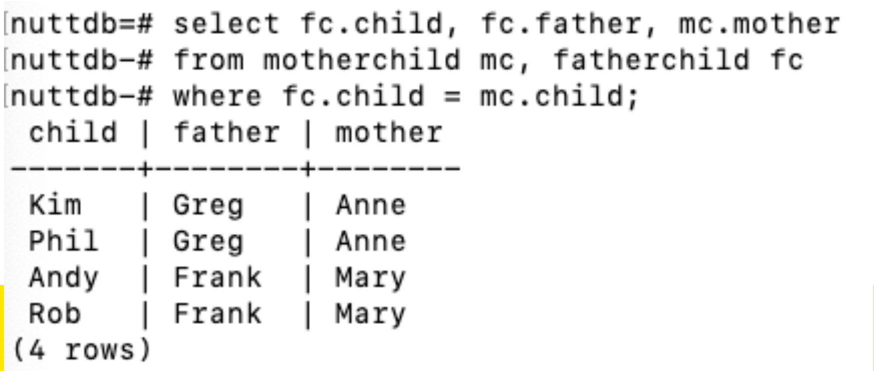
Query 4

“Father and mother of every person”

… can be calculated in relational algebra by means of a natural join

FatherChild $\bowtie$ MotherChild

```
select fc.child, fc.father, mc.mother
from   motherChild mc, fatherChild fc
where  fc.child = mc.child
```



```
[nuttdb=# select fc.child, fc.father, mc.mother
[nuttdb-# from motherchild mc, fatherchild fc
[nuttdb-# where fc.child = mc.child;
 child | father | mother
-------+--------+--------
 Kim   | Greg   | Anne
 Phil  | Greg   | Anne
 Andy  | Frank  | Mary
 Rob   | Frank  | Mary
(4 rows)
```

Query 5 Join and Other Operations

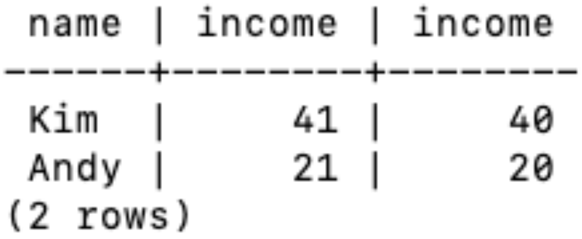
“Persons that earn more than their father,
showing name, income, and income of the father”

Write the query in SQL

Query 5.

“Persons that earn more than their father,
showing name, income, and income of the father”

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

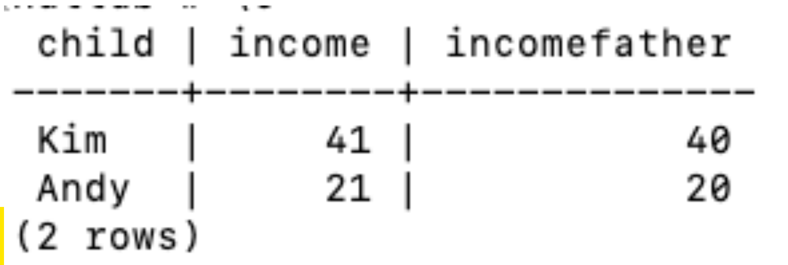


```
 name | income | income
------+--------+--------
 Kim  |     41 |     40
 Andy |     21 |     20
(2 rows)
```

select, with Renaming of the Result

For the persons that earn more than their father, show their name, income, and the income of the father

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



```
 child | income | incomefather
-------+--------+--------------
 Kim   |     41 |           40
 Andy  |     21 |           20
(2 rows)
```

Explicit Join

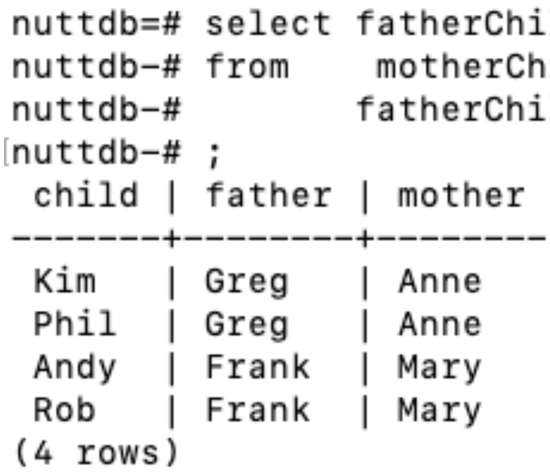
For every person, return the person, their father and their mother

```
select fatherChild.child, father, mother
from   motherChild join fatherChild on
       fatherChild.child = motherChild.child
```

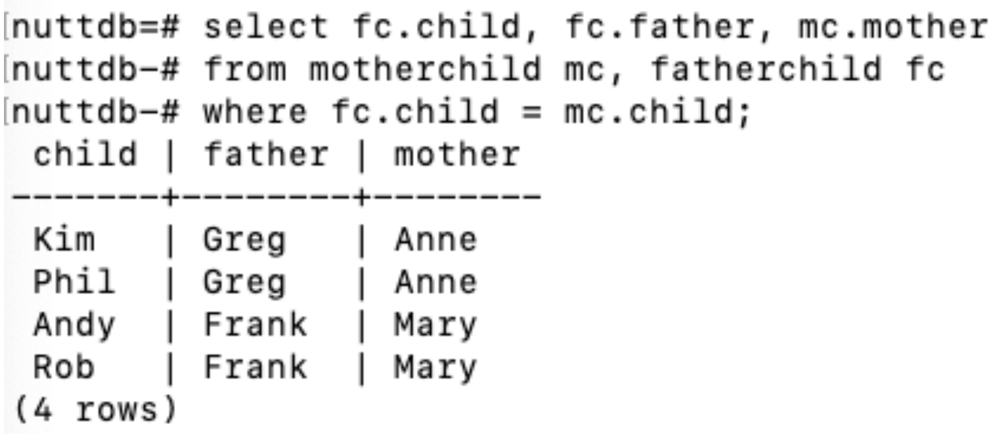
```
select …
from    Table { join Table on JoinCondition }, …
[ where OtherCondition ]
```

Explicit Join

For every person, return the person, their father and their mother



```
nuttdb=# select fatherChild.child, father, mother
nuttdb-# from    motherChild join fatherChild on
nuttdb-#         fatherChild.child = motherChild.child
nuttdb-# ;
 child | father | mother
-------+--------+--------
 Kim   | Greg   | Anne
 Phil  | Greg   | Anne
 Andy  | Frank  | Mary
 Rob   | Frank  | Mary
(4 rows)
```



```
nuttdb=# select fc.child, fc.father, mc.mother
nuttdb-# from motherchild mc, fatherchild fc
nuttdb-# where fc.child = mc.child;
 child | father | mother
-------+--------+--------
 Kim   | Greg   | Anne
 Phil  | Greg   | Anne
 Andy  | Frank  | Mary
 Rob   | Frank  | Mary
(4 rows)
```

Query 5 with explicit joins

“For the persons that earn more than their father, show their name, income, and the income of the father”

```
select c.name, c.income, f.income
from   person c
          join fatherChild fc on c.name = fc.child
          join person f on fc.father = f.name
where  c.income > f.income
```

An equivalent formulation without explicit join:

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

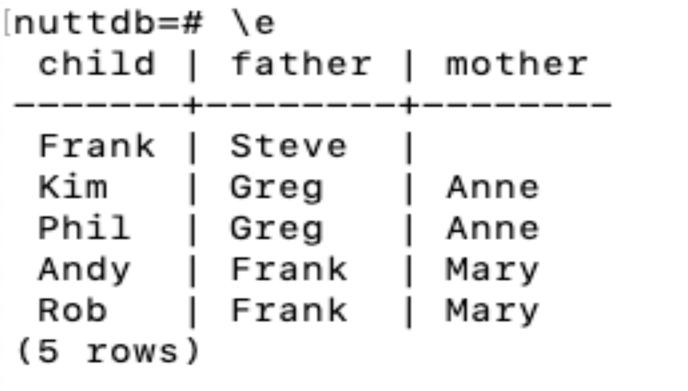
Outer Join

“For every person, return the father and, if known, the mother”

```
select fatherChild.child, father, mother
from   fatherChild left outer join motherChild
       on fatherChild.child = motherChild.child
```

Note: “`outer`” is optional

```
select fatherChild.child, father, mother
from   fatherChild left join motherChild
       on fatherChild.child = motherChild.child
```



```
nuttdb=# \e
 child | father | mother
-------+--------+--------
 Frank | Steve  |
 Kim   | Greg   | Anne
 Phil  | Greg   | Anne
 Andy  | Frank  | Mary
 Rob   | Frank  | Mary
(5 rows)
```

Ordering the Result: `order by`

“Return name and income of persons under thirty, in alphabetic order of the names”

```
select name, income
from   person
where  age < 30
order by  name
```

ascending order

```
select name, income
from   person
where  age < 30
order by  name desc
```

descending order

Ordering the Result: `order by`

```
select name, income
from   person
where  age < 30
```

name	income
Andy	21
Rob	15
Mary	42

```
select name, income
from   person
where  age < 30
order by name
```

name	income
Andy	21
Mary	42
Rob	15

Aggregate Operators

Among the expressions in the target list (i.e., projection list), we can also have expressions that calculate values based on a group of tuples:

- count, minimum (min), maximum (max), average (avg), sum

Example: How many children has Frank?

```
select count(*) as NumFranksChildren
from   fatherChild
where  father = 'Frank'
```

Results of `count`: Example

FatherChild

father	child
Steve	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob

NumFranksChildren
2

count and Null Values

```
select count(*)
from   person
```

Result = number of tuples
= 4

```
select count(income)
from   person
```

Result = number of values different from NULL
= 3

```
select count(distinct income)
from   person
```

Result = number of distinct values (excluding NULL)
= 2

Person

name	age	income
Andy	27	21
Rob	25	NULL
Mary	55	21
Anne	50	35

Aggregate Operators and Null Values

```
select avg(income) as meanIncome
from   person
```

Person

name	age	income
Andy	27	30
Rob	25	NULL
Mary	55	36
Anne	50	36

is
ignored

meanIncome
34

Aggregate Operators and the Projection List

An incorrect query (whose name should be returned?):

```
select name, max(income)
from   person
```

The projection list has to be **homogeneous**, for example:

```
select min(age), avg(income)
from   person
```

Aggregate Operators and Grouping

- Aggregation functions can be applied to partitions of the tuples of a relations

- To specify the partition of tuples, one uses the `group by` clause:

`group by` *attributeList*

Aggregate Operators and Grouping

The number of children of every father.

```
select father, count(*) as NumChildren
from   fatherChild
group by father
```

FatherChild

father	child
Steve	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob

father	NumChildren
Steve	1
Greg	2
Frank	2

Query 6: `group by`

“For each group of adult persons (age > 17) who have the same age, return the maximum income for every group and show the age”

Write the query in SQL!

Person

name	age	income

Query 6

“For each group of adult persons who have the same age, return the maximum income for every group and show the age”

```
select age, max(income)
from   person
where  age > 17
group by age
```

Grouping and Projection List

In a query that has a `group by` clause, **only** such attributes appear in the `group by` clause can appear in the projection list (except for aggregation functions)

Example: **Incorrect**: income of persons, grouped according to age

```
select age, income
from   person
group by age
```

The above is wrong … because there could exist several values for the same group.

Correct: average income of persons, grouped by age.

```
select age, avg(income)
from   person
group by age
```

`Make the attribute aggregate`

Grouping and Target List (cntd)

The syntactic restriction on the attributes in the select clause holds also for queries that would be semantically correct (i.e., for which there is only a single value of the attribute for every group).

Example: Fathers with their income and with the average income of their children.

Incorrect:

```
 select fc.father, avg(c.income), f.income
 from   person c join fatherChild fc on c.name=fc.child
                 join person f on fc.father=f.name
 group by fc.father
```

Correct:

```
 select fc.father, avg(c.income), f.income
 from   person c join fatherChild fc on c.name=fc.child
                 join person f on fc.father=f.name
 group by fc.father, f.income
```

Conditions on Groups (“having” clause)

It is also possible to **filter the groups** using selection conditions.
Clearly, the selection of groups differs from the selection of the tuples in the `where` clause: the tuples form the groups.

To filter the groups, the “having clause” is used.

The having clause must appear after the “`group by`”

Example: “Fathers whose children have an average income greater 25.”

```
select fc.father, avg(c.income)
from   person c join fatherChild fc
       on c.name = fc.child
group by fc.father
having avg(c.income) > 25
```

Query 7. `where` or `having`?

“Fathers whose children under age 30 have an average income greater 20”

Query 7.

“Fathers whose children under the age of 30 have an average income greater 20”

```
select father, avg(f.income)
from   person c join fatherChild fc
       on c.name = fc.child
where  c.age < 30
group by cf.father
having avg(c.income) > 20
```

Union, Intersection, and Difference

Within a `select` statement one cannot express unions.

An explicit construct is needed:

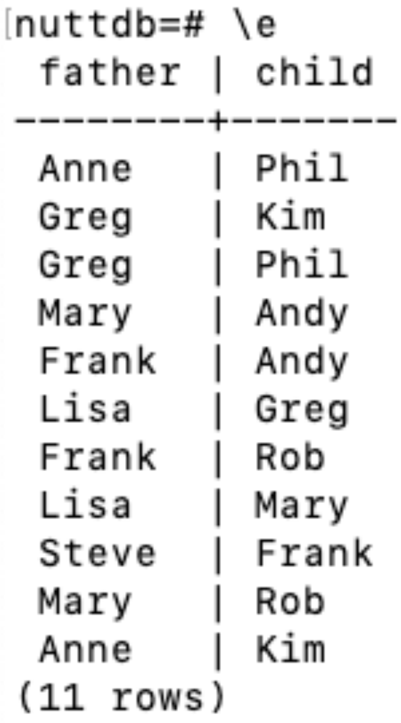
```
select ...
union [all]
select ...
```

With `union`, duplicates are eliminated (also those originating from projection).

With `union all` duplicates are kept.

Positional Notation of Attributes

```
select father, child
from   fatherChild
union
select mother, child
from   motherChild
```



```
[nuttdb=# \e
 father | child
--------+-------
 Anne   | Phil
 Greg   | Kim
 Greg   | Phil
 Mary   | Andy
 Frank  | Andy
 Lisa   | Greg
 Frank  | Rob
 Lisa   | Mary
 Steve  | Frank
 Mary   | Rob
 Anne   | Kim
(11 rows)
```

Which are the attribute names of the result?

Those of the first operand!

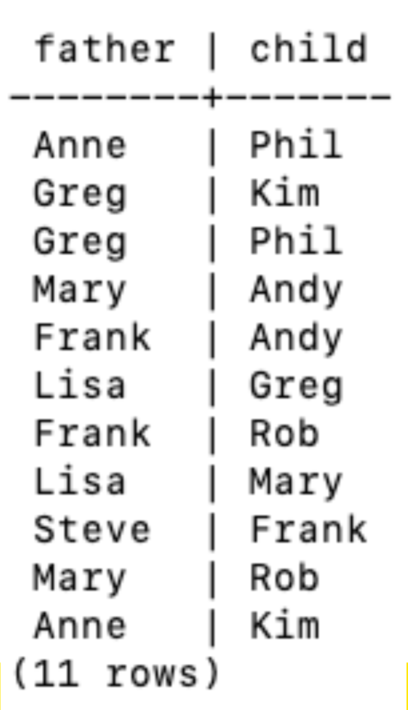
→ SQL matches attributes in the same position

→ SQL renames the attributes of the second operand

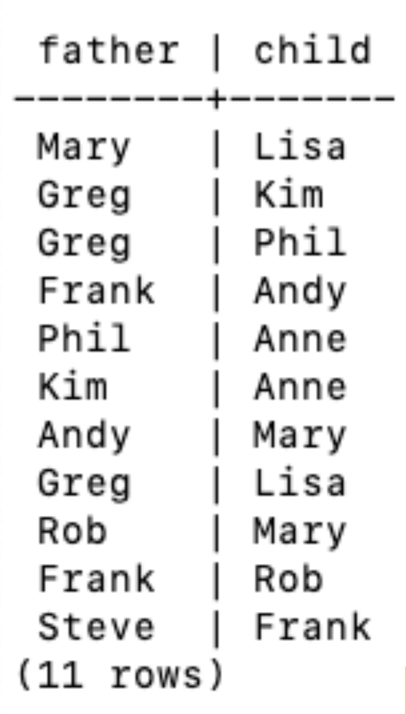
Positional Notation: Example

```
select father, child
from    fatherChild
union
select mother, child
from   motherChild
```

```
select father, child
from   fatherChild
union
select child, mother
from   motherChild
```



```
 father | child
--------+-------
 Anne   | Phil
 Greg   | Kim
 Greg   | Phil
 Mary   | Andy
 Frank  | Andy
 Lisa   | Greg
 Frank  | Rob
 Lisa   | Mary
 Steve  | Frank
 Mary   | Rob
 Anne   | Kim
(11 rows)
```



```
 father | child
--------+-------
 Mary   | Lisa
 Greg   | Kim
 Greg   | Phil
 Frank  | Andy
 Phil   | Anne
 Kim    | Anne
 Andy   | Mary
 Greg   | Lisa
 Rob    | Mary
 Frank  | Rob
 Steve  | Frank
(11 rows)
```

Positional Notation (cntd)

Renaming does not change anything:

```
select father as parent, child
from   fatherChild
union
select child, mother as parent
from   motherChild
```

Correct (if we want to treat fathers and mothers as parents):

```
select father as parent, child
from   fatherChild
union
select mother as parent, child
from   motherChild
```

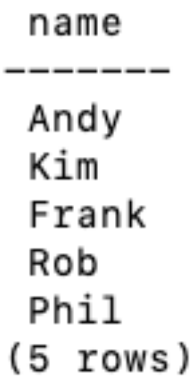
Difference

```
select name
from   person
except
select child as name
from   fatherChild
```

We will see that differences can also be expressed with nested `select` statements.

Intersection

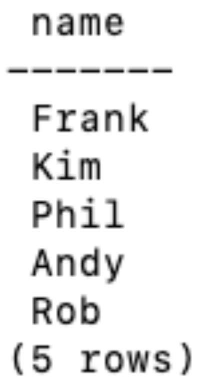
```
select name
from   person
intersect
select child as name
from   fatherChild
```



```
 name
-------
 Andy
 Kim
 Frank
 Rob
 Phil
(5 rows)
```

is equivalent to

```
select person.name
from   person, fatherChild
where  person.name = fatherChild.child
```



```
 name
-------
 Frank
 Kim
 Phil
 Andy
 Rob
(5 rows)
```