SQL Notes—Part 2

SELECT … FROM … WHERE can express projection, joins, and selection

But in addition to those SQL also provides specialized constructs to support:

1. Aggregates
2. Explicit Joins, and
3. Set Union, and
4. Set Difference from Relational Algebra.

Explicit Join

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

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

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

UNION

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
```

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

union all will keep duplicates

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

Result of

the Union

father	child
Greg	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob
Lisa	Mary
Lisa	Greg
Anne	Kim
Anne	Phil
Mary	Andy
Mary	Rob

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
```

For set difference: Replace ‘union’ by ‘except’

Set Difference: **except**

Find children who have a father but not a mother:

```
select  child as childname
from FatherChild
        except
select child from Mothedhild
```

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

Queries from Hw

customer(customer-name, street, city)

branch(branch-name, city)

account(customer-name, branch-name, account-number)

(*c) Find the branches that do not have any accounts.*

(d) Find the customer names who do not have any account in the 'Region12' branch.

(e) Find the customer names who have accounts in all the branches located in 'Los Angeles'.

(f) Find the customer names who have only one account.

To express these in SQL, we need nested subqueries

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

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')
```

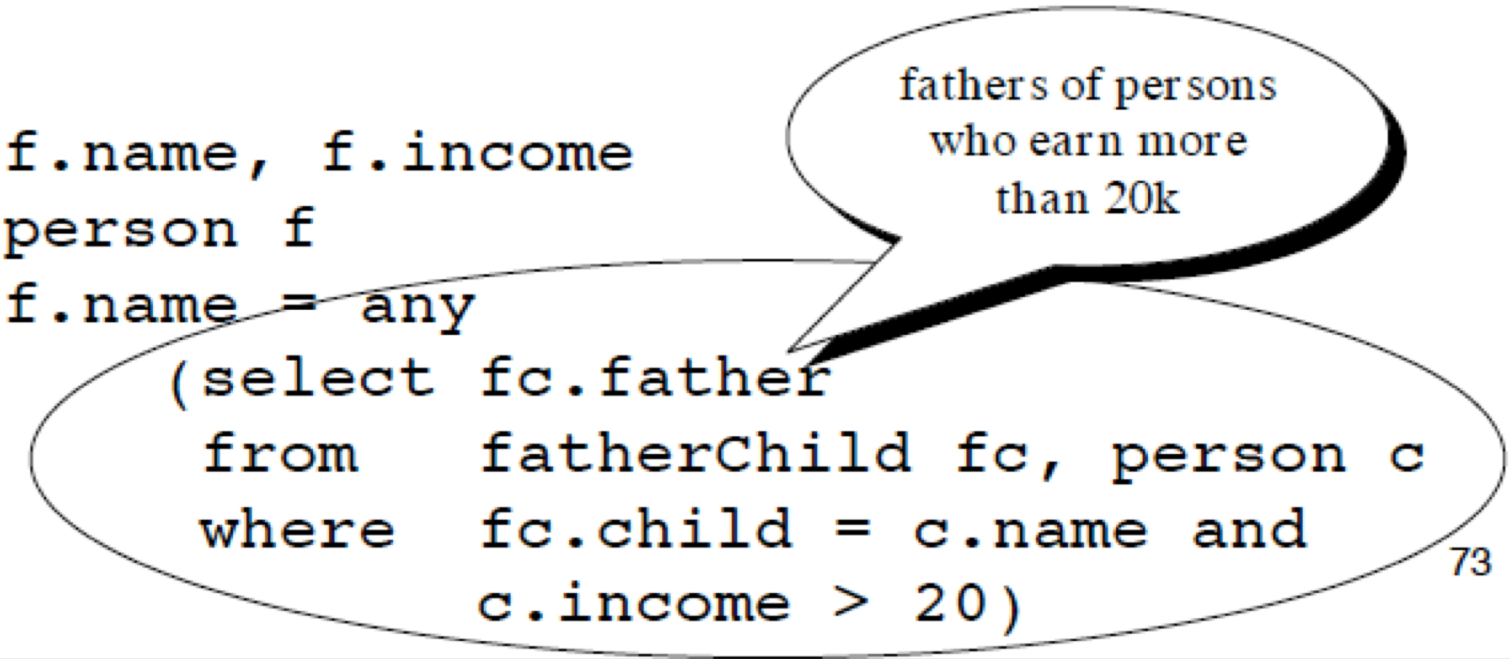
Instead of = (i.e., equal to any) we could have used IN (i.e. in the set)

Nested Queries: Example

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

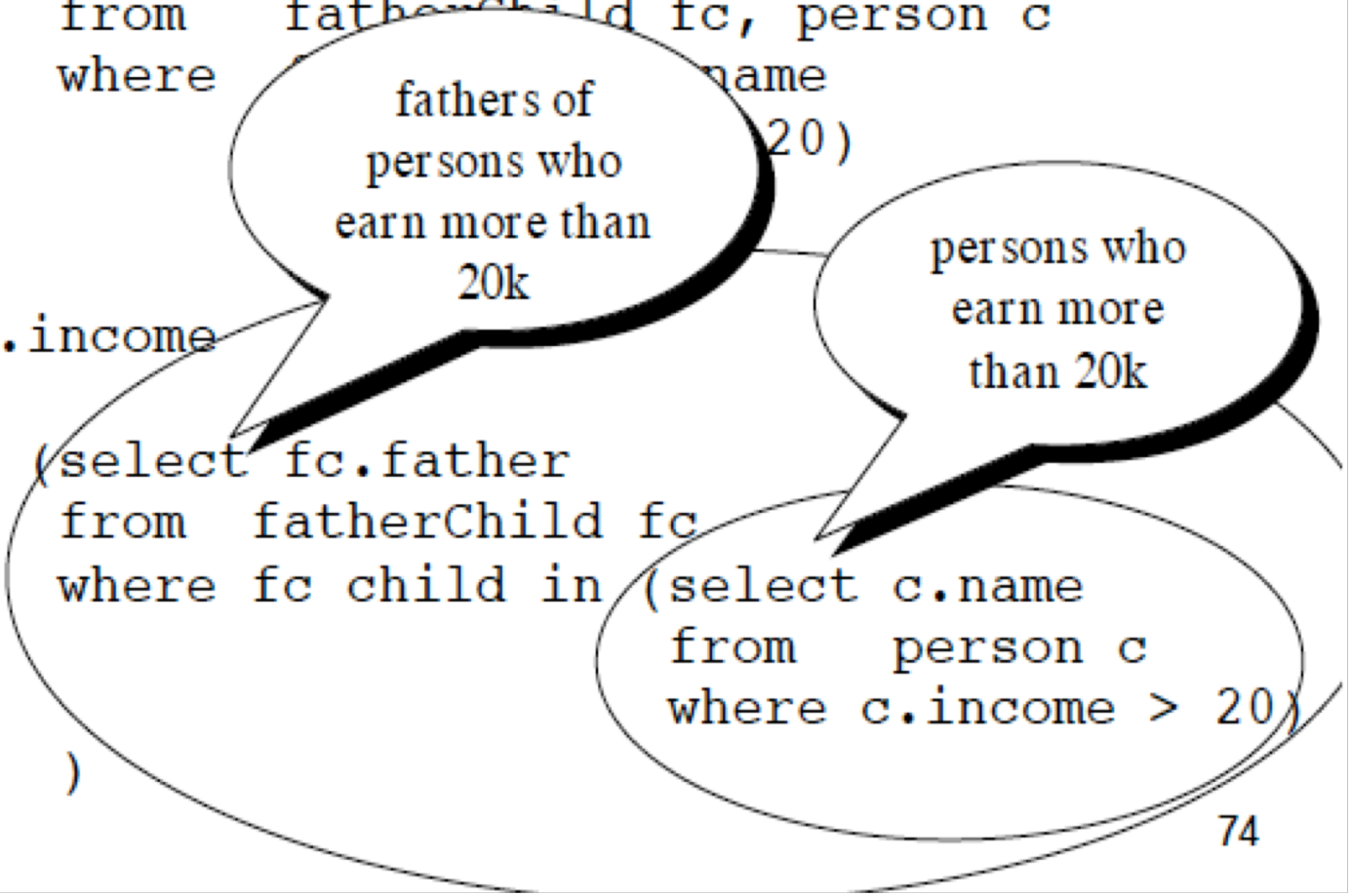
```
select distinct f.name, f.income
from   person f, fatherChild fc, person c
where  f.name = fc.father and
       fc.child = c.name and c.income > 20
```

```
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)
```



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  f.name
                                    20)
```




```
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)
                  )
```

Using nested subqueries to express joins does does not bring obvious benefits, in general.

However nesting combined with constructs such as ALL, NOT EXIST, and IN allows us to express queries that cannot be expressed without these new constructs.

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 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.mother = c0.name
                  where  mc0.mother = mc.mother)
```

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 *
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.

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.

Why?

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.mother = c0.name
              where  mc0.mother = mc.mother and
                     c0.age < 18)
```

What about childless mothers? Note difference w.r.t. natural language.

Replace **mc0.mother=c0.name** by **mc0.chikd=c0.name**

Exercise 9

"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`

Equivalent formulation with max

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

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

Is this always equivalent to the previous query that used ALL?

Name and age of mothers whose children are all >=18: using min

```
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.mother = c0.name
              where  mc0.mother = mc.mother)
```

The EXISTS construct is also supported… but it is not as useful as NOT 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 *
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.

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)
```

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

Nested Queries: Comments

- Visibility rules:
  - it is not possible to refer to a variable defined in a block Inside ~~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

More on Visibility = Variable Scope

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

SET DIFF and NESTING

employee(name, lastname, dept)

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)
```

"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)
```

Can we do it with not exist?

Nested Queries: 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)
```

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

Explicit Join

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

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

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

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

Joins and Information Preservation by Nulls

Find the Mother and Father of each child

MotherChild

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

FatherChild

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

$MotherChild \bowtie FatherChild$

With join: Unmatched values are lost from both left (i.e., motherChild) and right (i.e., fatherChild)

But with we can use NULL values to fill in for the missing value: outerjoins

(Lisa, Mary, Null) *left-outer join*

…..

(Null, Steve, Frank) *left-outer join*

Outer Join: Examples

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

Unmatched values are lost from both left (i.e., motherChild) and right (i.e., fatherChild)

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

Values on the left are preserved, since unmatched ones are padded with null values-

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

Values on the right are preserved, since unmatched ones are padded with null values

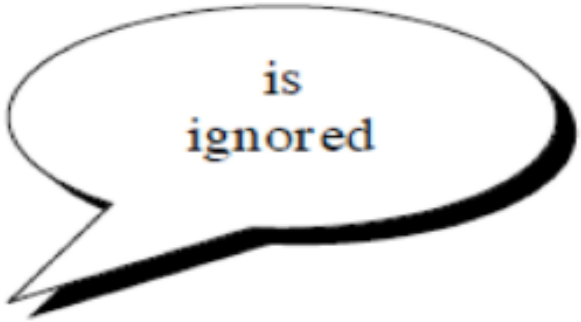
```
select fatherChild.child, father, mother
from   motherChild full outer join fatherChild
       on motherChild.child = fatherChild.child
```

Values on both right and Left are preserved, since unmatched ones are padded with null values

Null Values

Select name from person where income <100

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

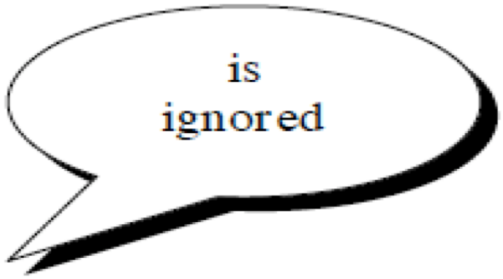


Three-valued logic: True, False and Logical Null

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



meanIncome
34

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

SQL Queries from Hw by outerjoins

Customer(customer-name, street, city)

Branch(branch-name, city)

Account(customer-name, branch-name, account-number)

%*the join of Customer and Account has the following attributes*
(customer-name, street, city, customer-name, branch-name, account-number)

c) Find the branches that do not have any account.

```
SELECT   customer-name
FROM     Customer  LEFT OUTERJOIN  Account  ON
         Customer.customer-mame=Account.customer-name
WHERE    ISNULL(account-number )
```

Queries from Hw by Outerjoins

Customer(customer-name, street, city)

Branch(branch-name, city)

Account(customer-name, branch-name, account-number)

(***c) Find the branches that do not have any accounts.***

```
SELECT    customer-name
FROM      Customer  LEFT OUTERJOIN  Account
          ON Customer.customer-mame=Account.customer-name
HAVING    COUNT(account-number)=  0
```

But COUNT(*) will not work because this counts the number of tuple occurrences, independent of their values.