Practice class script 6

# Content

* How to include SELECT as table into other SELECT statement
* CREATE VIEW
* Scalar subqueries
* Set operations. EXISTS, SOME, ALL

We will use the same “Pizzeria ” tables and data as in class 5

# How to combine multiple SELECT statements and why it is necessary?

## Union rows of two SELECT statements. (Practical class 3, 4)

select 'income', sum(order\_amount) from orders where status = 'DONE'

union all

select 'salaries', sum(salary) from employment where end\_date is null

Number and type of columns should be the same

## SELECT as a “table”

Result of one SELECT statement can be used as table in other statement. In following select:

SELECT order\_id, sum(list\_price) price, sum(cost) cost

FROM order\_items JOIN products ON product\_id = products.id

GROUP BY order\_id

… if you put it in the brackets and at the end give the name it can be used as table

SELECT \* FROM orders join

(SELECT order\_id, sum(list\_price) price, sum(cost) cost

FROM order\_items JOIN products ON product\_id = products.id

GROUP BY order\_id) order\_details

on orders.id = order\_details.order\_id

We can use subselect’s column names:

SELECT ID, delivery\_address, order\_amount, price

FROM orders left join

(SELECT order\_id, sum(list\_price) price, sum(cost) cost

FROM order\_items JOIN products ON product\_id = products.id

GROUP BY order\_id) order\_details

on orders.id = order\_details.order\_id

where price > 10

Already created non trivial subselect can be reused in other selects.

Aggregate functions can be used step by step. For example, we have SELECT statement that selects number of deliveries

select delivery\_address, count(\*) dcount

from orders group by delivery\_address

We can create max number of deliveries:

select max(dcount) from

(select delivery\_address, count(\*) dcount

from orders group by delivery\_address) counts

## VIEW – saved SELECT

To create view: CREATE VIEW [name] AS SELECT …  
View can be used as ordinary table   
SELECT \* FROM orders join

(SELECT order\_id, sum(list\_price) cena, sum(cost) cost

FROM order\_items JOIN products ON product\_id = products.id

GROUP BY order\_id) order\_details

on orders.id = order\_details.order\_id

It can be written as

create view order\_details as

SELECT order\_id, sum(list\_price) price, sum(cost) cost

FROM order\_items JOIN products ON product\_id = products.id

GROUP BY order\_id

SELECT \* FROM orders join order\_details

on orders.id = order\_details.order\_id

Only restrictions – all columns should have names and ORDER BY is no allowed

Usage:

* To reuse the same logic (SELECT)
* Version compatibility when table structure is changed (view “old\_table”, that actually SELECT data from “new\_table”)
* Access rights (user sees only data selected by view)

## Scalar subselects

Scalar SELECT is SELECT that returns only one row with one column. It can be used in expressions as value.  
Max order amount:

select max(order\_amount) from orders

Max order (or orders if amounts are the same)

select \* from orders

where order\_amount = (select max(order\_amount) from orders)

Average delivery time:

select avg(delivery\_minutes) from orders

Column where calculated each order delivery time as percentage from average delivery time:

select \*, 100.0\*delivery\_minutes/(select avg(delivery\_minutes) from orders) procents\_from\_average from orders

## Subselects refering main select

This finds all orders and minimal delivery time in one select.

select \*, (select min(delivery\_minutes) from orders) best\_time

from orders

These two “FROM ORDERS” are actually two SELECTs from table orders.

This SELCT finds for each order also minimal delivery time to that address.

select \*,

(select min(delivery\_minutes) from orders o2

where o2.delivery\_address = orders.delivery\_address) best\_time

from orders

## Set comparison

Subselects can be used together with set operators.

select distinct courier from orders  
select \* from employees where ID in (4,5,6)  
select \* from employees where ID in (select distinct courier from orders)

Operator EXISTS – true if subselect returns at least one row. Slow deliverers:

select \* from employees

where exists

(select \* from orders

where delivery\_minutes>40

and courier = employees.id)

The same table can be used more than once, but you should rename it.  
All deliveries to addresses where was slow deliveries.

select \* from orders

where exists

(select \* from orders o2

where delivery\_minutes>40

and orders.delivery\_address = o2.delivery\_address)

Deliveries to addresses where never was slow deliveries:

select \* from orders

where not exists

(select \* from orders o2

where delivery\_minutes>40

and orders.delivery\_address = o2.delivery\_address)

Fastest deliveries – delivery time is less or equal than all other deliveries time. NULL should be excluded!

select \* from orders

where delivery\_minutes <=

ALL(select delivery\_minutes from orders where delivery\_minutes is not null)

Delivieries that was not the fastest:

select \* from orders

where delivery\_minutes > SOME(select delivery\_minutes from orders)

Equivalent SELECT with aggregate function.

select \* from orders

where delivery\_minutes > (select min(delivery\_minutes) from orders)

# Excercises

SELECT orders with discount (where order\_amount is not equal with order items price sum)

SELECT orders for most productive courier

Find addresses that never ordered soft drink (product.id = 103)

Find full order details of the orders that contains pineapple pizza (id=101)

SELECT employees who do not delivered any order

# Solutions for first three excercises

SELECT orders with discount (where order\_amount is not equal with order items price sum)

select \* from

(SELECT order\_id, sum(list\_price) cena, sum(cost) izmaksas

FROM order\_items JOIN products ON product\_id = products.id

GROUP BY order\_id) order\_details

join orders on orders.id = order\_id

where cena <> order\_amount

It is equivalent to VIEW order\_details created before, that allow to do it easier

select \* from order\_details

join orders on orders.id = order\_id

where cena <> order\_amount

“SELECT orders for most productive courier” – there could be more than one such courier

select courier, count(\*) ccount

from orders

group by courier

This find “productivity” of couriers

select max(ccount) from

(select courier, count(\*) ccount from orders group by courier) ccounts

This finds largest number of deliveries by using previous select

select courier from orders group by courier

having count(\*) >= (select max(ccount)

from (select courier, count(\*) ccount

from orders group by courier) ccounts)

Previous SELECT is used as subselect to find courier(s) with largest count of deliveries

select \* from orders

where courier in

(select courier from orders group by courier

having count(\*) >= (select max(skaits)

from (select courier, count(\*) skaits

from orders group by courier) skaiti))

Previous SELECTS is used to find deliveries for couriers with largest deliveries count   
Sometimes it is necessary to create SELECT with many (in this case 4) levels of SELECTs. It is allowed and sometimes necessary.

Find addresses that never ordered soft drink (product.id = 103) – two different solutions are possible

select distinct delivery\_address from orders

where delivery\_address not in

(select distinct delivery\_address

from order\_items

join orders on orders.id = order\_id

where product\_id = 103)

select distinct delivery\_address from orders

where not exists

( select \*

from orders o2 join order\_items on o2.id = order\_id

where o2.delivery\_address = orders.delivery\_address   
 and product\_id=103)