LEKCE 6
Vkládání poddotazů do dotazů

Poddotaz (subquery) je dotaz, jehož výsledky se předají jako argument jinému dotazu. Díky poddotazům můžete svázat několik dotazů dohromady. Na konci této lekce budete schopni provádět následující:

- sestavovat poddotazy,
- používat ve svých poddotazech klíčová slova `EXISTS`, `ANY` a `ALL`,
- sestavovat a používat korelované poddotazy.

V této lekci budeme pracovat s tabulkami `PART` a `ORDERS`. K vytvoření a naplnění těchto tabulek proveďte prosím následující činnosti ve svém databázovém systému MySQL. Databázi `kuba` v následujícím příkladu nahraďte názvem vámi vytvořené databáze, do níž chcete tabulky umístit:

Vstup/výstup ▼

```
mysql> use kuba;
Database changed
mysql> show tables;
+----------------+
| Tables_in_kuba |
+----------------+
| characters     |
| checks         |
| orders         |
| orgchart       |
| part           |
| teamstats      |
+----------------+
6 rows in set (0.00 sec)
```

Pro příklady v této kapitole budete potřebovat tabulky `PART` a `ORDERS`. Pokud je dosud nemáte, zde je kód pro jejich vytvoření a naplnění:

Vstup ▼

```
create table part
(partnum     numeric(10)   not null,
 description varchar(20)   not null,
 price       decimal(10,2) not null);
```

```
create table orders
(orderedon date,
 name      varchar(16) not null,
 partnum   numeric(10) not null,
 quantity  numeric(10) not null,
 remarks   varchar(30) not null);

insert into part values
('54', 'Pedály', '542.50');
insert into part values
('42', 'Sedla', '245.00');
insert into part values
('46', 'Pneu', '152.50');
insert into part values
('23', 'Horské kolo', '3504.50');
insert into part values
('76', 'Silniční kolo', '5300.00');
insert into part values
('10', 'Dvojkolo', '12000.00');

insert into orders values
('2006-03-15', 'Mega Kola', '23', '6', 'Zaplaceno');
insert into orders values
('2006-03-19', 'Mega Kola', '76', '3', 'Zaplaceno');
insert into orders values
('2006-09-02', 'Mega Kola', '10', '1', 'Zaplaceno');
insert into orders values
('2006-06-30', 'Mega Kola', '42', '8', 'Zaplaceno');
insert into orders values
('2006-06-30', 'CykloSpec', '54', '10', 'Zaplaceno');
insert into orders values
('2006-05-30', 'CykloSpec', '23', '8', 'Zaplaceno');
insert into orders values
('2006-01-17', 'CykloSpec', '76', '11', 'Zaplaceno');
insert into orders values
('2006-01-17', 'LX Obchůdek', '76', '5', 'Zaplaceno');
insert into orders values
('2006-06-01', 'LX Obchůdek', '10', '3', 'Zaplaceno');
insert into orders values
('2006-06-01', 'Cyklo ABC', '10', '1', 'Zaplaceno');
insert into orders values
('2006-07-01', 'Cyklo ABC', '76', '4', 'Zaplaceno');
insert into orders values
('2006-07-01', 'Cyklo ABC', '46', '14', 'Zaplaceno');
insert into orders values
('2006-07-11', 'Cyklo Franta', '76', '14', 'Zaplaceno');
```

POZNÁMKA

Příklady v této lekci jsou pro databázový systém MySQL. Ujistěte se, že používáte verzi databázového systému MySQL 4.1 nebo vyšší, protože nižší verze nepodporují poddotazy.

Sestavujeme poddotazy

Jednoduše řečeno, poddotazy vám umožňují svázat výslednou sadu jednoho dotazu s jiným. Obecná syntaxe vypadá takto:

Syntaxe ▼

```
SELECT *
FROM tabulka1
WHERE tabulka1.nejaky_sloupec =
(SELECT jiny_sloupec
 FROM tabulka2
 WHERE jiny_sloupec = nejaka_hodnota)
```

Všimněte si, jak je druhý dotaz vnořen do prvního. Podívejme se na aktuální obsah tabulek, které budeme používat při konstrukci příkladů z reálného světa:

Vstup/výstup ▼

```
mysql> select * from part;
+---------+---------------+----------+
| partnum | description   | price    |
+---------+---------------+----------+
|      54 | Pedály        | 542.50   |
|      42 | Sedla         | 245.00   |
|      46 | Pneu          | 152.50   |
|      23 | Horské kolo   | 3504.50  |
|      76 | Silniční kolo | 5300.00  |
|      10 | Dvojkolo      | 12000.00 |
+---------+---------------+----------+
6 rows in set (0.04 sec)

mysql> select * from orders;
+------------+--------------+---------+----------+-----------+
| orderedon  | name         | partnum | quantity | remarks   |
+------------+--------------+---------+----------+-----------+
| 2006-03-15 | Mega Kola    |      23 |        6 | Zaplaceno |
| 2006-03-19 | Mega Kola    |      76 |        3 | Zaplaceno |
| 2006-09-02 | Mega Kola    |      10 |        1 | Zaplaceno |
| 2006-06-30 | Mega Kola    |      42 |        8 | Zaplaceno |
| 2006-06-30 | CykloSpec    |      54 |       10 | Zaplaceno |
| 2006-05-30 | CykloSpec    |      23 |        8 | Zaplaceno |
| 2006-01-17 | CykloSpec    |      76 |       11 | Zaplaceno |
| 2006-01-17 | LX Obchůdek  |      76 |        5 | Zaplaceno |
| 2006-06-01 | LX Obchůdek  |      10 |        3 | Zaplaceno |
| 2006-06-01 | Cyklo ABC    |      10 |        1 | Zaplaceno |
| 2006-07-01 | Cyklo ABC    |      76 |        4 | Zaplaceno |
| 2006-07-01 | Cyklo ABC    |      46 |       14 | Zaplaceno |
| 2006-07-11 | Cyklo Franta |      76 |       14 | Zaplaceno |
+------------+--------------+---------+----------+-----------+
13 rows in set (0.01 sec)
```

Tabulky sdílejí společné pole s názvem PARTNUM. Předpokládejme, že neznáme (nebo nechceme vědět) hodnotu pole PARTNUM, ale místo toho chceme pracovat s popisem položky. S využitím poddotazu můžeme napsat následující příkaz:

Vstup/výstup ▼

```
mysql> select *
    -> from orders where partnum =
    -> (select partnum
    ->  from part
    ->  where description like 'Silniční%');
+------------+--------------+---------+----------+-----------+
| orderedon  | name         | partnum | quantity | remarks   |
+------------+--------------+---------+----------+-----------+
| 2006-03-19 | Mega Kola    |      76 |        3 | Zaplaceno |
| 2006-01-17 | CykloSpec    |      76 |       11 | Zaplaceno |
| 2006-01-17 | LX Obchůdek  |      76 |        5 | Zaplaceno |
| 2006-07-01 | Cyklo ABC    |      76 |        4 | Zaplaceno |
| 2006-07-11 | Cyklo Franta |      76 |       14 | Zaplaceno |
+------------+--------------+---------+----------+-----------+
5 rows in set (0.00 sec)
```

Podívejme se nyní podrobně na princip poddotazů. K tomu nám poslouží výše uvedený dotaz, který si rozložíme na jednotlivé části:

Vstup/výstup ▼

```
mysql> select partnum
    -> from part
    -> where description like 'Silniční%';
+---------+
| partnum |
+---------+
|      76 |
+---------+
1 row in set (0.04 sec)
```

Analýza ▼

V příkladu pro databázový systém MySQL můžete vidět rozklad poddotazu. Vzhledem k tomu, že poddotaz je vždy uzavřen do závorek, vyhodnotí se jako první. Výsledná sada (76) je poté porovnána (testována na rovnost) se sloupcem PARTNUM tabulky ORDERS. Níže je uveden příklad výsledné sady z vnějšího dotazu:

Vstup/výstup ▼

```
mysql> select * from orders
    -> where partnum = 76;
+------------+--------------+---------+----------+-----------+
| orderedon  | name         | partnum | quantity | remarks   |
+------------+--------------+---------+----------+-----------+
| 2006-03-19 | Mega Kola    |      76 |        3 | Zaplaceno |
```

```
| 2006-01-17 | CykloSpec    |      76 |       11 | Zaplaceno |
| 2006-01-17 | LX Obchůdek  |      76 |        5 | Zaplaceno |
| 2006-07-01 | Cyklo ABC    |      76 |        4 | Zaplaceno |
| 2006-07-11 | Cyklo Franta |      76 |       14 | Zaplaceno |
+------------+--------------+---------+----------+-----------+
5 rows in set (0.00 sec)
```

Zde jsme již schopni do podmínky v naší klauzuli `WHERE` dosadit konkrétní hodnotu, kterou jsme získali pomocí poddotazu.

Když jsme začínali, tak jsme věděli jen to, že potřebujeme všechny řádky z tabulky `ORDERS`, které obsahují položku, jejíž popis začíná slovem „Silniční".

Díky poddotazu máme možnost získat data z obou tabulek, aniž bychom je museli jakkoli spojovat. Zde je příklad, v němž pro dosažení téhož výsledku používáme spojení tabulek:

Vstup/výstup ▼

```
mysql> select o.orderedon,
    ->        o.name,
    ->        o.partnum,
    ->        o.quantity,
    ->        o.remarks
    -> from orders o, part p
    -> where o.partnum = p.partnum
    ->   and p.description like 'Silniční%';
+------------+--------------+---------+----------+-----------+
| orderedon  | name         | partnum | quantity | remarks   |
+------------+--------------+---------+----------+-----------+
| 2006-03-19 | Mega Kola    |      76 |        3 | Zaplaceno |
| 2006-01-17 | CykloSpec    |      76 |       11 | Zaplaceno |
| 2006-01-17 | LX Obchůdek  |      76 |        5 | Zaplaceno |
| 2006-07-01 | Cyklo ABC    |      76 |        4 | Zaplaceno |
| 2006-07-11 | Cyklo Franta |      76 |       14 | Zaplaceno |
+------------+--------------+---------+----------+-----------+
5 rows in set (0.02 sec)
```

Ba co víc, pokud použijete principy, které jste se naučili v lekci 5, pak můžete sloupec `PARTNUM` ve výsledku rozšířit o sloupec `DESCRIPTION`, což přispěje k lepší čitelnosti výsledku:

Vstup/výstup ▼

```
mysql> select o.orderedon, o.partnum,
    ->        p.description,o.quantity,o.remarks,
    -> from orders o, part p
    -> where o.partnum=p.partnum
    -> and
    -> o.partnum =
    -> (select partnum
    ->  from part
    ->  where description like 'Silniční%');
```

```
+------------+----------+---------------+----------+-----------+
| orderedon  | partnum  | description   | quantity | remarks   |
+------------+----------+---------------+----------+-----------+
| 2006-03-19 |       76 | Silniční kolo |        3 | Zaplaceno |
| 2006-01-17 |       76 | Silniční kolo |       11 | Zaplaceno |
| 2006-01-17 |       76 | Silniční kolo |        5 | Zaplaceno |
| 2006-07-01 |       76 | Silniční kolo |        4 | Zaplaceno |
| 2006-07-11 |       76 | Silniční kolo |       14 | Zaplaceno |
+------------+----------+---------------+----------+-----------+
5 rows in set (0.02 sec)
```

První část dotazu je již více než známá:

```
SELECT O.ORDEREDON, O.PARTNUM,
       P.DESCRIPTION, O.QUANTITY, O.REMARKS
FROM ORDERS O, PART P
```

Zde pomocí aliasů `O` a `P` pro tabulky `ORDERS` a `PART` vybíráme pět sloupců, které nás zajímají. V tomto případě aliasy sloupců nepotřebujeme, protože každý z požadovaných sloupců má jedinečný název. Na druhou stranu jsme tak vytvořili poměrně dobře čitelný dotaz, což by později mohlo být mnohem obtížnější. První klauzule `WHERE` vypadá takto:

```
WHERE O.PARTNUM = P.PARTNUM
```

Jedná se o standardní tvar pro spojování tabulek `PART` a `ORDERS` uvedených v klauzuli `FROM`. Pokud bychom klauzuli `WHERE` nepoužili, obdrželi bychom všechny možné kombinace řádků těchto dvou tabulek. Další část obsahuje poddotaz.

```
AND
O.PARTNUM =
(SELECT PARTNUM
 FROM PART
 WHERE DESCRIPTION LIKE "Silniční%")
```

Tímto příkazem přidáváme upřesnění, které říká, že se pole `O.PARTNUM` musí rovnat výsledku našeho jednoduchého poddotazu. V něm hledáme všechna čísla položek, jejichž popis začíná slovem „Silniční“. Operátor `LIKE` nám šetří úhozy na klávesnici, protože díky němu nemusíme psát „Silniční kolo“. Jenže za okamžik se ukáže, že jsme tentokrát nezvolili příliš šťastně. Představte si, že by někdo v oddělení součástek přidal novou součástku s názvem „Silniční brzdy“.

Syntaxe pro přidání řádku se součástkou „Silniční brzdy“ do tabulky `PART` vypadá takto:

Vstup/výstup ▼

```
mysql> insert into part values
    -> (77,'Silniční brzdy',79.90);
Query OK, 1 row affected (0.00 sec)
```

Nová verze tabulky `PART` nyní vypadá následovně:

Vstup/výstup ▼

```
mysql> select * from part;
```

```
+---------+----------------+----------+
| partnum | description    | price    |
+---------+----------------+----------+
|      54 | Pedály         | 542.50   |
|      42 | Sedla          | 245.00   |
|      46 | Pneu           | 152.50   |
|      23 | Horské kolo    | 3504.50  |
|      76 | Silniční kolo  | 5300.00  |
|      10 | Dvojkolo       | 12000.00 |
|      77 | Silniční brzdy |    79.90 |
+---------+----------------+----------+
7 rows in set (0.00 sec)
```

Předpokládejme, že o této změně vůbec nevíme, a zkusme nyní spustit náš dotaz:

Vstup ▼

```
mysql> select o.orderedon, o.partnum,
    ->        p.description, o.quantity, o.remarks
    -> from orders o, part p
    -> where o.partnum = p.partnum
    -> and
    -> o.partnum =
    -> (select partnum
    ->  from part
    ->  where description like 'Silniční%');
```

Pokud jej zadáme, místo výsledků obdržíme následující chybové hlášení:

```
ERROR 1242 (21000): Subquery returns more than 1 row
```

Odpověď vámi používaného interpretu jazyka SQL se může malinko lišit. Podstatné ale je, že nevrátí žádné výsledky.

Vžijme se nyní do role interpretu jazyka SQL a pojďme zjistit, co se vlastně stalo. Nejdříve vyhodnotí poddotaz, takže vrátí následující výsledek:

Vstup/výstup ▼

```
mysql> select partnum
    -> from part
    -> where description like 'Silniční%';
+---------+
| partnum |
+---------+
|      76 |
|      77 |
+---------+
2 rows in set (0.00 sec)
```

Tento výsledek nyní vezmeme a aplikujeme na výraz `O.PARTNUM =`, což je zřejmě krok, který působí určitý problém.

Analýza ▼

Jak se může pole PARTNUM rovnat hodnotě 76 i 77? Něco takového měl na mysli interpret jazyka SQL, když vracel chybu. Při každém použití klauzule LIKE se otevíráme tomuto typu chyby. Jakmile kombinujeme výsledky relačního operátoru s jiným relačním operátorem (např. =, < >), pak musíme dbát na to, aby byl výsledek singulární. Náš příklad tedy můžeme opravit tak, že v dotazu nahradíme operátor LIKE operátorem =:

Vstup/výstup ▼

```
mysql> select o.orderedon, o.partnum,
    ->        p.description, o.quantity, o.remarks
    -> from orders o, part p
    -> where o.partnum=p.partnum
    -> and
    -> o.partnum=
    -> (select partnum
    ->  from part
    ->  where description = 'Silniční kolo');
+------------+---------+---------------+----------+-----------+
| orderedon  | partnum | description   | quantity | remarks   |
+------------+---------+---------------+----------+-----------+
| 2006-03-19 |      76 | Silniční kolo |        3 | Zaplaceno |
| 2006-01-17 |      76 | Silniční kolo |       11 | Zaplaceno |
| 2006-01-17 |      76 | Silniční kolo |        5 | Zaplaceno |
| 2006-07-01 |      76 | Silniční kolo |        4 | Zaplaceno |
| 2006-07-11 |      76 | Silniční kolo |       14 | Zaplaceno |
+------------+---------+---------------+----------+-----------+
5 rows in set (0.02 sec)
```

Tento poddotaz vrátí pouze jediný výsledek, takže v podmínce = bude jen jediná hodnota. Jak si můžeme být jisti, že poddotaz nevrátí více hodnot, když hledáme jen jedinou hodnotu?

Ze všeho nejlepší je nepoužívat operátor LIKE. Další možnost spočívá v zajištění jedinečnosti vyhledávacího pole při návrhu tabulky. Jste-li nedůvěřiví, pak můžete pomocí metody (popsané v předchozí lekci) pro spojení tabulky se sebou ověřit jedinečnost daného pole. Pokud si navrhujete tabulky sami (viz lekce 9) nebo důvěřujete osobě, která je navrhuje, pak můžete vyžadovat, aby měl sloupec, podle něhož vyhledáváte, jedinečné hodnoty. Kromě toho můžete použít jistou část jazyka SQL, která vrací pouze jedinou odpověď: agregační funkci.

Agregační funkce v poddotazech

Všechny agregační funkce – SUM, COUNT, MIN, MAX a AVG – vracejí jedinou hodnotu. K nalezení průměrné hodnoty objednávky můžete použít následující příkaz:

Vstup/výstup ▼

```
mysql> select avg(o.quantity * p.price)
    -> from orders o, part p
    -> where o.partnum = p.partnum
    -> ;
```

```
+--------------------------+
| avg(o.quantity * p.price) |
+--------------------------+
|              24206.384615 |
+--------------------------+
1 row in set (0.00 sec)
```

Tento příkaz vrací pouze jedinou hodnotu. Ke zjištění, které objednávky mají nadprůměrnou hodnotu, lze v poddotaze použít výše uvedený příkaz `SELECT`. Celý dotaz i s výsledkem vypadá takto:

Vstup/výstup ▼

```
mysql> select o.name, o.orderedon,
    -> o.quantity * p.price total
    -> from orders o, part p
    -> where o.partnum = p.partnum
    -> and
    -> o.quantity * p.price >
    -> (select avg(o.quantity * p.price)
    ->  from orders o, part p
    ->  where o.partnum = p.partnum);
+---------------+------------+----------+
| name          | orderon    | total    |
+---------------+------------+----------+
| CykloSpec     | 2006-05-30 | 28036.00 |
| CykloSpec     | 2006-01-17 | 58300.00 |
| LX Obchůdek   | 2006-01-17 | 26500.00 |
| LX Obchůdek   | 2006-06-01 | 36000.00 |
| Cyklo Franta  | 2006-07-11 | 74200.00 |
+---------------+------------+----------+
5 rows in set (0.02 sec)
```


Tento příklad obsahuje poněkud všední klauzule `SELECT`/`FROM`/`WHERE`:

```
SELECT O.NAME, O.ORDEREDON,
       O.QUANTITY * P.PRICE TOTAL
FROM ORDERS O, PART P
WHERE O.PARTNUM = P.PARTNUM
```

Tyto řádky představují běžný způsob spojování těchto dvou tabulek. Toto spojení je nezbytné, protože cena je v tabulce `PART` a množství v tabulce `ORDERS`. Klauzule `WHERE` zajišťuje, aby se spojily pouze související řádky. Dále jsme přidali následující poddotaz:

```
AND
O.QUANTITY * P.PRICE >
(SELECT AVG(O.QUANTITY * P.PRICE)
 FROM ORDERS O, PART P
 WHERE O.PARTNUM = P.PARTNUM)
```

Výše uvedená podmínka porovnává celkovou cenu každé objednávky s průměrem vypočítaným v poddotaze. Všimněte si, že spojení v poddotaze je nutné ze stejného důvodu jako v hlavním příkazu `SELECT`. Toto spojení má navíc úplně stejný tvar.

V poddotazech nejsou ukryty žádné tajnosti. Mají úplně stejnou syntaxi jako samostatné dotazy. Ve skutečnosti začíná většina poddotazů jako samostatné dotazy, které se po otestování výsledků začleňují jako poddotazy.

Vnořování poddotazů

Vnoření znamená vsazení poddotazu do jiného poddotazu.

Syntaxe ▼

```
SELECT * FROM neco WHERE (poddotaz1(poddotaz2(poddotaz3)));
```

Poddotazy lze vnořovat tak hluboko, jak jen vám dovoluje vámi používaná implementace jazyka SQL. Například k odeslání speciálních oznámení zákazníkům, kteří utratili více než průměrnou částku, lze využít data v tabulce `CUSTOMER`:

Vstup/výstup ▼

```
mysql> select *
    -> from customer;
+--------------+-----------------+-----------+-------+-----------+---------+
| name         | address         | town      | zip   | phone     | remarks |
+--------------+-----------------+-----------+-------+-----------+---------+
| Mega Kola    | Hačice 253      | Hačice    | 58702 | 581123456 | Nic     |
| CykloSpec    | Dolní 86        | Brno      | 45678 | 771654321 | Nic     |
| LX Obchůdek  | Smetanova 15    | Brno      | 54678 | 771333222 | Nic     |
| Cyklo ABC    | Jarní 6         | Prostějov | 56784 | 771111000 | Honza   |
| Cyklo Franta | Prostějovská 10 | Bedihošť  | 34567 | 771789456 | Nic     |
+--------------+-----------------+-----------+-------+-----------+---------+
5 rows in set (0.43 sec)
```

Tyto informace nyní zkombinujeme s malinko upravenou verzí dotazu, který jsme použili k vyhledání objednávek s nadprůměrnou částkou:

Vstup/výstup ▼

```
mysql> select all c.name, c.address, c.town, c.zip
    -> from customer c
    -> where c.name in
    -> (select o.name
    ->  from orders o, part p
    ->  where o.partnum = p.partnum
    ->  and
    ->  o.quantity * p.price >
    ->  (select avg(o.quantity * p.price)
    ->   from orders o, part p
    ->   where o.partnum = p.partnum));
+-------------+----------------+----------+-------+
| name        | address        | town     | zip   |
+-------------+----------------+----------+-------+
| CykloSpec   | Dolní 86       | Brno     | 45678 |
```

```
| LX Obchůdek  | Smetanova 15    | Brno     | 54678 |
| Cyklo Franta | Prostějovská 10 | Bedihošť | 34567 |
+--------------+-----------------+----------+-------+
3 rows in set (0.03 sec)
```

Zde je to, oč v tomto dotazu žádáme. V nejvnitřnějších závorkách se nachází známý příkaz:

```
SELECT AVG(O.QUANTINTY * P.PRICE)
FROM ORDERS O, PART P
WHERE O.PARTNUM = P.PARTNUM
```

Výsledek tohoto dotazu vstupuje do lehce upravené verze již dříve použité klauzule `SELECT`:

```
SELECT O.NAME
FROM ORDERS O, PART P
WHERE O.PARTNUM = P.PARTNUM
AND
O.QUANTINTY * P.PRICE >
(...)
```

Všimněte si, že klauzule `SELECT` byla upravena tak, aby vracela jediný sloupec `NAME`, který je ne náhodou společný s tabulkou `CUSTOMER`. Spuštěním tohoto samotného dotazu obdržíme následující výsledek:

Vstup/výstup ▼

```
mysql> select o.name
    -> from orders o, part p
    -> where o.partnum = p.partnum
    -> and
    -> o.quantity * p.price >
    -> (select avg(o.quantity * p.price)
    ->  from orders o, part p
    ->  where o.partnum = p.partnum);
+--------------+
| name         |
+--------------+
| CykloSpec    |
| CykloSpec    |
| LX Obchůdek  |
| LX Obchůdek  |
| Cyklo Franta |
+--------------+
5 rows in set (0.00 sec)
```

Před chvílí jsme strávili nějaký čas diskuzí nad tím, proč by vaše poddotazy měly vracet jen jedinou hodnotu. Důvod, proč byl tento dotaz schopen vrátit více než jednu hodnotu, bude za okamžik zcela zjevný.

Výše uvedené výsledky nakonec vstupují do příkazu:

```
SELECT C.NAME, C.ADDRESS, C.TOWN, C.ZIP
FROM CUSTOMER C
WHERE C.NAME IN
(...)
```

První dva řádky nejsou ničím zajímavé. Na třetím řádku se znovu setkáváme s klíčovým slovem `IN`, s nímž jsme naposledy pracovali v lekci 2. Klíčové slovo `IN` umožňuje používat víceřádkový výstup poddotazu. Jak si jistě pamatujete, hledá klíčové slovo `IN` shody v sadě hodnot uzavřené do závorek. V tomto případě obdržíme následující hodnoty:

```
CykloSpec
CykloSpec
LX Obchůdek
LX Obchůdek
Cyklo Franta
```

Tento poddotaz poskytuje podmínky, které dávají následující seznam adres:

```
+--------------+-----------------+----------+-------+
| name         | address         | town     | zip   |
+--------------+-----------------+----------+-------+
| CykloSpec    | Dolní 86        | Brno     | 45678 |
| LX Obchůdek  | Smetanova 15    | Brno     | 54678 |
| Cyklo Franta | Prostějovská 10 | Bedihošť | 34567 |
+--------------+-----------------+----------+-------+
```

Klíčové slovo `IN` se v poddotazech používá velice často. K porovnávání používá sadu hodnot, a proto nezpůsobí v interpretu jazyka SQL chybu.

Poddotazy lze používat také s klauzulemi `GROUP BY` a `HAVING`. Podívejte se na následující dotaz:

Vstup/výstup ▼

```
mysql> select name, avg(quantity)
    -> from orders
    -> group by name
    -> having avg(quantity) >
    -> (select avg(quantity)
    ->  from orders);
+--------------+---------------+
| name         | avg(quantity) |
+--------------+---------------+
| Cyklo Franta |       14.0000 |
| CykloSpec    |        9.6667 |
+--------------+---------------+
2 rows in set (0.11 sec)
```

Prozkoumejme nyní tento dotaz tak, jak to provádí interpret jazyka SQL. Nejdříve se tedy podíváme na poddotaz:

Vstup/výstup ▼

```
mysql> select avg(quantity)
    -> from orders;
+---------------+
| avg(quantity) |
+---------------+
|        6.7692 |
+---------------+
1 row in set (0.00 sec)
```

Hlavní část dotazu vypadá sama o sobě takto:

Vstup/výstup ▼

```
mysql> select name, avg(quantity)
    -> from orders
    -> group by name
+--------------+---------------+
| name         | avg(quantity) |
+--------------+---------------+
| Cyklo ABC    |        6.3333 |
| Cyklo Franta |       14.0000 |
| CykloSpec    |        9.6667 |
| LX Obchůdek  |        4.0000 |
| Mega Kola    |        4.5000 |
+--------------+---------------+
5 rows in set (0.00 sec)
```

Při zkombinování s klauzulí `HAVING` vytvoří poddotaz dva řádky, které mají nadprůměrnou hodnotu v poli `QUANTITY`.

Vstup/výstup ▼

```
HAVING AVG(QUANTITY) >
(SELECT AVG(QUANTITY)
 FROM ORDERS)

NAME              AVG
------------  -------
CykloSpec      9.6667
Cyklo Franta  14.0000
```


Vnější reference s korelovanými poddotazy

Poddotazy, které jsme dosud napsali, jsou soběstačné. V žádném z nich nepoužíváme referenci z vnějšku poddotazu. *Korelované poddotazy* umožňují používat vnější referenci se zvláštními a současně zajímavými výsledky. Podívejte se na následující dotaz:

Vstup/výstup ▼

```
mysql> select *
    -> from orders o
    -> where 'Silniční kolo' =
    -> (select description
    ->  from part p
    ->  where p.partnum = o.partnum);
+------------+--------------+---------+----------+-----------+
| orderedon  | name         | partnum | quantity | remarks   |
+------------+--------------+---------+----------+-----------+
| 2006-03-19 | Mega Kola    |      76 |        3 | Zaplaceno |
| 2006-01-17 | CykloSpec    |      76 |       11 | Zaplaceno |
```

```
| 2006-01-17 | LX Obchůdek  |      76 |        5 | Zaplaceno |
| 2006-07-01 | Cyklo ABC    |      76 |        4 | Zaplaceno |
| 2006-07-11 | Cyklo Franta |      76 |       14 | Zaplaceno |
+------------+--------------+---------+----------+-----------+
5 rows in set (0.01 sec)
```

Tento dotaz se ve skutečnosti podobá následujícímu spojení:

Vstup/výstup ▼

```
mysql> select o.orderedon, o.name,
    ->        o.partnum, o.quantity, o.remarks
    -> from orders o, part p
    -> where p.partnum = o.partnum
    ->   and p.description = 'Silniční kolo';
+------------+--------------+---------+----------+-----------+
| orderedon  | name         | partnum | quantity | remarks   |
+------------+--------------+---------+----------+-----------+
| 2006-03-19 | Mega Kola    |      76 |        3 | Zaplaceno |
| 2006-01-17 | CykloSpec    |      76 |       11 | Zaplaceno |
| 2006-01-17 | LX Obchůdek  |      76 |        5 | Zaplaceno |
| 2006-07-01 | Cyklo ABC    |      76 |        4 | Zaplaceno |
| 2006-07-11 | Cyklo Franta |      76 |       14 | Zaplaceno |
+------------+--------------+---------+----------+-----------+
5 rows in set (0.00 sec)
```

Analýza ▼

Výsledky jsou naprosto stejné. Korelovaný poddotaz funguje podobně jako spojení. Korelace je ustavena použitím elementu z dotazu v poddotazu. V tomto příkladu jsme korelaci ustavili příkazem:

```
WHERE P.PARTNUM = O.PARTNUM
```

Zde porovnáváme pole `P.PARTNUM` z tabulky uvnitř poddotazu a pole `O.PARTNUM` z tabulky vně dotazu. Vzhledem k tomu, že `O.PARTNUM` může mít na každém řádku odlišnou hodnotu, provede se korelovaný poddotaz pro každý řádek dotazu. V následujícím příkladu se každý řádek tabulky `ORDERS`:

Vstup/výstup ▼

```
mysql> select *
    -> from orders;
+------------+--------------+---------+----------+-----------+
| orderedon  | name         | partnum | quantity | remarks   |
+------------+--------------+---------+----------+-----------+
| 2006-03-15 | Mega Kola    |      23 |        6 | Zaplaceno |
| 2006-03-19 | Mega Kola    |      76 |        3 | Zaplaceno |
| 2006-09-02 | Mega Kola    |      10 |        1 | Zaplaceno |
| 2006-06-30 | Mega Kola    |      42 |        8 | Zaplaceno |
| 2006-06-30 | CykloSpec    |      54 |       10 | Zaplaceno |
| 2006-05-30 | CykloSpec    |      23 |        8 | Zaplaceno |
```

```
| 2006-01-17 | CykloSpec    |      76 |       11 | Zaplaceno |
| 2006-01-17 | LX Obchůdek  |      76 |        5 | Zaplaceno |
| 2006-06-01 | LX Obchůdek  |      10 |        3 | Zaplaceno |
| 2006-06-01 | Cyklo ABC    |      10 |        1 | Zaplaceno |
| 2006-07-01 | Cyklo ABC    |      76 |        4 | Zaplaceno |
| 2006-07-01 | Cyklo ABC    |      46 |       14 | Zaplaceno |
| 2006-07-11 | Cyklo Franta |      76 |       14 | Zaplaceno |
+------------+--------------+---------+----------+-----------+
13 rows in set (0.00 sec)
```

zpracuje podle kritéria poddotazu:

```
SELECT DESCRIPTION
FROM PART P
WHERE P.PARTNUM = O.PARTNUM
```

Tato operace vrátí popis (pole `DESCRIPTION`) každého řádku v tabulce `PART`, pro který platí `P.PARTNUM = O.PARTNUM`. Tyto popisy pak porovnáme pomocí klauzule `WHERE`:

```
WHERE 'Silniční kolo' =
```

Analýza ▼

Prozkoumává se každý řádek, a proto může mít poddotaz v korelovaném poddotazu více než jednu hodnotu. Nepokoušejte se ovšem vracet více sloupců nebo sloupce, které v kontextu klauzule `WHERE` nedávají smysl. Vrácené hodnoty totiž musí odpovídat operaci uvedené v klauzuli `WHERE`. Pokud bychom kupříkladu v právě provedeném dotazu vraceli pole `PRICE` a porovnávali jej s textem „Silniční kolo“, pak bychom obdrželi následující výsledek:

Vstup/výstup ▼

```
SQL> SELECT *
   2 FROM ORDERS O
   3 WHERE 'Silniční kolo' =
   4 (SELECT PRICE
   5  FROM PART P
   6  WHERE P.PARTNUM = O.PARTNUM);

conversion error from string "Silniční kolo"
```

Zde je další ukázka toho, co byste neměli dělat:

```
SELECT *
FROM ORDERS O
WHERE 'Silniční kolo' =
(SELECT *
 FROM PART P
 WHERE P.PARTNUM = O.PARTNUM)
```

Tento příkaz `SELECT` způsobí zásadní chybu. Interpret jazyka SQL prostě nedokáže korelovat všechny sloupce v tabulce `PART` s operátorem =.

Korelované poddotazy lze používat také v klauzulích `GROUP BY` a `HAVING`. V následujícím dotazu používáme korelovaný poddotaz ke zjištění průměrné hodnoty objednávky pro konkrétní součástku a tento průměr pak použijeme k odfiltrování celkových hodnot objednávek seskupených podle sloupce `PARTNUM`:

Vstup/výstup ▼

```
mysql> select o.partnum, sum(o.quantity*p.price), count(p.partnum)
    -> from orders o, part p
    -> where p.partnum = o.partnum
    -> group by o.partnum
    -> having sum(o.quantity*p.price) >
    -> (select avg(o1.quantity*p1.price)
    ->  from part p1, orders o1
    ->  where p1.partnum = o1.partnum
    ->    and p1.partnum = o.partnum);
+---------+-------------------------+------------------+
| partnum | sum(o.quantity*p.price) | count(p.partnum) |
+---------+-------------------------+------------------+
|      10 |                60000.00 |                3 |
|      23 |                49063.00 |                2 |
|      76 |               196100.00 |                5 |
+---------+-------------------------+------------------+
3 rows in set (0.01 sec)
```

Analýza ▼

Poddotaz nepočítá jen jeden průměr pomocí funkce `AVG(O1.QUANTITY*P1.PRICE)`. Kvůli korelaci mezi dotazem a poddotazem (`AND P1.PARTNUM = O.PARTNUM`) je tento průměr počítán pro každou skupinu součástek a poté porovnán:

```
HAVING SUM(O.QUANTITY*P.PRICE) >
```

TIP

> Při použití korelovaných poddotazů s klauzulemi `GROUP BY` a `HAVING` se sloupce v klauzuli `HAVING` musejí nacházet buď v klauzuli `SELECT`, nebo v klauzuli `GROUP BY`. V opačném případě obdržíte u řádků s neplatným sloupcem chybovou zprávu, protože poddotaz se vyhodnocuje pro každou skupinu, a ne pro každý řádek. Nemůžete přece provést platné porovnání s něčím, co se nepoužívá v dané skupině.

Klíčová slova EXISTS, ANY a ALL

Použití klíčových slov `EXISTS`, `ANY` a `ALL` není pro náhodného pozorovatele intuitivně zřejmé. Operátor `EXISTS` přijímá jako argument poddotaz a vrací buď hodnotu `TRUE`, pokud tento poddotaz něco vrátí, nebo `FALSE`, pokud je jeho výsledná sada prázdná:

Vstup/výstup ▼

```
mysql> select name, orderedon
    -> from orders
```

```
    -> where exists
    -> (select *
    ->  from orders
    ->  where name = 'Mega Kola');
+--------------+------------+
| NAME         | ORDEREDON  |
+--------------+------------+
| Mega Kola    | 2006-03-15 |
| Mega Kola    | 2006-03-19 |
| Mega Kola    | 2006-09-02 |
| Mega Kola    | 2006-06-30 |
| CykloSpec    | 2006-06-30 |
| CykloSpec    | 2006-05-30 |
| CykloSpec    | 2006-01-17 |
| LX Obchůdek  | 2006-01-17 |
| LX Obchůdek  | 2006-06-01 |
| Cyklo ABC    | 2006-06-01 |
| Cyklo ABC    | 2006-07-01 |
| Cyklo ABC    | 2006-07-01 |
| Cyklo Franta | 2006-07-11 |
+--------------+------------+
13 rows in set (0.00 sec)
```

Poddotaz uvnitř `EXISTS` se v tomto nekorelovaném příkladu vyhodnotí pouze jednou. Výsledek poddotazu obsahuje nejméně jeden řádek, a proto se `EXISTS` vyhodnotí na `TRUE` a vypíšou se všechny řádky v dotazu. Pokud poddotaz změníme níže uvedeným způsobem, pak neobdržíme žádné výsledky.

```
SELECT NAME, ORDEREDON
FROM ORDERS
WHERE EXISTS
(SELECT *
 FROM ORDERS
 WHERE NAME ='Povětšinou neškodný')
```

Operátor `EXISTS` se zde vyhodnotí na `FALSE`. Poddotaz negeneruje žádný výsledek, protože text „Povětšinou neškodný“ neodpovídá žádnému ze jmen.

POZNÁMKA

Všimněte si, že v poddotazu uvnitř operátoru `EXISTS` používáme `SELECT *`. Operátor `EXISTS` se totiž nestará o počet vrácených sloupců.

Tímto způsobem lze pomocí operátoru `EXISTS` ověřit existenci určitých řádků a řídit výstup dotazu na základě jejich přítomnosti či nepřítomnosti.

Použijeme-li operátor `EXISTS` v korelovaném poddotazu, vyhodnotí se pro každý případ definovaný vytvořenou korelací:

Vstup/výstup ▼

```
mysql> select name, orderedon
    -> from orders o
```

```
    -> where exists
    -> (select *
    ->  from customer c
    ->  where town = 'Brno'
    ->    and c.name = o.name)
+-------------+------------+
| NAME        | ORDEREDON  |
+-------------+------------+
| CykloSpec   | 2006-06-30 |
| CykloSpec   | 2006-05-30 |
| CykloSpec   | 2006-01-17 |
| LX Obchůdek | 2006-01-17 |
| LX Obchůdek | 2006-06-01 |
+-------------+------------+
5 rows in set (0.00 sec)
```

Tato drobná modifikace prvního, nekorelovaného dotazu vrátí všechny obchody s jízdními koly z Brna, které provedly objednávky. Následující poddotaz se spustí pro každý řádek v dotazu korelovaném podle jmen v tabulkách `CUSTOMER` a `ORDER`:

```
(SELECT *
 FROM CUSTOMER C
 WHERE TOWN = 'Brno'
 AND C.NAME = O.NAME)
```

Operátor `EXISTS` vrátí hodnotu `TRUE` pro ty řádky, které mají odpovídající jména v tabulce `CUSTOMER` s umístěním v Brně. V opačném případě vrátí hodnotu `FALSE`.

Při použití operátoru `EXISTS` není ani nutné, aby poddotaz vůbec vracel konkrétní data. Pokud jsou podmínky v poddotazu splněny, pak lze jednoduše vrátit libovolně zvolenou hodnotu. V následujícím příkladu vracíme místo všech sloupců (`*`) číslo 1, čímž zvýšíme výkon poddotazu:

```
SELECT NAME, ORDEREDON
FROM ORDERS O
WHERE EXISTS
(SELECT 1
 FROM CUSTOMER C
 WHERE TOWN = 'Brno'
 AND C.NAME = O.NAME)
```

S operátorem `EXISTS` úzce souvisejí také operátory `ANY`, `ALL` a `SOME`. Operátory `ANY` a `SOME` jsou, co se funkčnosti týče, naprosto identické. Optimista by řekl, že uživatel tak má na výběr, který z nich bude používat. Pesimista by tuto situaci viděl jako další komplikaci. Operátor `EXISTS` kontroluje, zda poddotaz vrátí jakákoli data. Operátory `ANY`, `ALL` a `SOME` se používají k porovnání hodnoty sloupce z dotazu s daty vrácenými poddotazem. Operátory `ANY` a `SOME` ověřují, zda se hodnota daného sloupce nachází v datech vrácených poddotazem. Operátor `ALL` se používá ke kontrole, zda hodnota daného sloupce přesně odpovídá hodnotě či hodnotám vráceným poddotazem. Podívejte se tento dotaz:

Vstup/výstup ▼

```
mysql> select name, orderedon
    -> from orders
    -> where name = any
    -> (select name
    ->  from orders
    ->  where name = 'Mega Kola');
+-----------+------------+
| NAME      | ORDEREDON  |
+-----------+------------+
| Mega Kola | 2006-03-15 |
| Mega Kola | 2006-03-19 |
| Mega Kola | 2006-09-02 |
| Mega Kola | 2006-06-30 |
+-----------+------------+
4 rows in set (0.00 sec)
```

Operátor `ANY` porovnává výstup následujícího poddotazu s každým řádkem dotazu a vrací hodnotu `TRUE` pro každý řádek dotazu, který obsahuje nějaký výsledek z poddotazu.

```
(SELECT NAME
 FROM ORDERS
 WHERE NAME = 'Mega Kola')
```

Po nahrazení `ANY` klíčovým slovem `SOME` obdržíme naprosto stejný výsledek:

Vstup/výstup ▼

```
mysql> select name, orderedon
    -> from orders
    -> where name = some
    -> (select name
    ->  from orders
    ->  where name = 'Mega Kola');
+-----------+------------+
| NAME      | ORDEREDON  |
+-----------+------------+
| Mega Kola | 2006-03-15 |
| Mega Kola | 2006-03-19 |
| Mega Kola | 2006-09-02 |
| Mega Kola | 2006-06-30 |
+-----------+------------+
4 rows in set (0.00 sec)
```

Pravděpodobně jste si již všimli podobnosti s operátorem `IN`. Stejný dotaz využívající operátor `IN` vypadá takto:

Vstup/výstup ▼

```
mysql> select name, orderedon
    -> from orders
    -> where name in
```

```
    -> (select name
    ->  from orders
    ->  where name = 'Mega Kola');
+-----------+------------+
| NAME      | ORDEREDON  |
+-----------+------------+
| Mega Kola | 2006-03-15 |
| Mega Kola | 2006-03-19 |
| Mega Kola | 2006-09-02 |
| Mega Kola | 2006-06-30 |
+-----------+------------+
4 rows in set (0.00 sec)
```

Jak můžete vidět, operátor `IN` vrací tentýž výsledek jako operátory `ANY` a `SOME`. Copak se svět úplně zbláznil? Ještě ne. Dokáže snad operátor `IN` tohle?

Vstup/výstup ▼

```
mysql> select name, orderedon
    -> from orders
    -> where name > any
    -> (select name
    ->  from orders
    ->  where name = 'Cyklo Franta');
+-------------+------------+
| NAME        | ORDEREDON  |
+-------------+------------+
| Mega Kola   | 2006-03-15 |
| Mega Kola   | 2006-03-19 |
| Mega Kola   | 2006-09-02 |
| Mega Kola   | 2006-06-30 |
| CykloSpec   | 2006-06-30 |
| CykloSpec   | 2006-05-30 |
| CykloSpec   | 2006-01-17 |
| LX Obchůdek | 2006-01-17 |
| LX Obchůdek | 2006-06-01 |
+-------------+------------+
9 rows in set (0.00 sec)
```

Odpověď je samozřejmě: nedokáže. Operátor `IN` funguje jako více rovnítek. Operátory `IN` a `SOME` lze použít s dalšími relačními operátory, jako je větší než nebo menší než. Dobře si jej proto zapamatujte.

Operátor `ALL` vrací hodnotu `TRUE` pouze tehdy, pokud všechny výsledky poddotazu splňují jistou podmínku. Používá se kupodivu jako dvojitý zápor:

Vstup/výstup ▼

```
mysql> select name, orderedon
    -> from orders
    -> where name <> all
    -> (select name
    ->  from orders
```

```
    ->  where name = 'Cyklo Franta');
+-------------+------------+
| NAME        | ORDEREDON  |
+-------------+------------+
| Mega Kola   | 2006-03-15 |
| Mega Kola   | 2006-03-19 |
| Mega Kola   | 2006-09-02 |
| Mega Kola   | 2006-06-30 |
| CykloSpec   | 2006-06-30 |
| CykloSpec   | 2006-05-30 |
| CykloSpec   | 2006-01-17 |
| LX Obchůdek | 2006-01-17 |
| LX Obchůdek | 2006-06-01 |
| Cyklo ABC   | 2006-06-01 |
| Cyklo ABC   | 2006-07-01 |
| Cyklo ABC   | 2006-07-01 |
+-------------+------------+
12 rows in set (0.00 sec)
```

Tento příklad vrací všechny obchody kromě „Cyklo Franta“. Výraz `<> ALL` se vyhodnotí na `TRUE` jen tehdy, pokud výsledná sada neobsahuje to, co je uvedeno na levé straně operátoru `<>`.

Shrnutí

V této lekci jste si vyzkoušeli desítky cvičení obsahujících poddotazy. Díky tomu jste se naučili, jak používat jednu z nejdůležitějších součástí jazyka SQL. Poddotaz představuje metodu pro umístění dodatečných podmínek na data vrácená dotazem. Poddotaz poskytuje úžasnou flexibilitu při definování podmínek, především pak podmínek, u nichž neznáte přesnou hodnotu. Představte si, že potřebujete získat seznam všech produktů s nadprůměrnou cenou, přičemž nemusíte okamžitě vědět, jaká je celková průměrná cena. V takovém případě sáhnete po poddotazu, který průměrnou cenu vypočítá.

V této lekci jsme též otevřeli jednu z nejobtížnějších částí jazyka SQL: korelované poddotazy. Korelované poddotazy vytvářejí vztah mezi dotazem a poddotazem, který se vyhodnocuje pro každou instanci tohoto vztahu. Kromě toho jste se dozvěděli o operátorech `EXISTS`, `ANY`, `SOME` a `ALL`, které se používají v poddotazech. Operátor `EXISTS` ověřuje, zda poddotaz vrací data na základě podmínek v tomto poddotazu. Operátory `ANY` a `SOME` jsou podobné jako operátor `IN` a kontrolují, zda se v datech vrácených poddotazem nachází hodnota daného sloupce. Operátor `ALL` se používá ke zjištění, zda jsou data sloupce stejná jako ta, která vrací poddotaz. Nenechte se odradit délkou výsledných dotazů. Snadno jim porozumíte, když si je rozdělíte na jednotlivé poddotazy.

Otázky a odpovědi

Otázka: V této lekci jsem si všiml, že v některých případech existuje pro získání téhož výsledku více způsobů. Není tato flexibilita matoucí?

Odpověď: To opravdu není. Díky tomu, že máte k dispozici více způsobů, jak dosáhnout téhož výsledku, můžete vytvářet opravdu parádní příkazy. Flexibilita je předností jazyka SQL.

Otázka: Jaké situace vyžadují, abych musel jít pro získání informace mimo dotaz?

Odpověď: Poddotazy vám umožňují lépe upřesnit podmínky na data, která váš dotaz vrátí. Pomocí poddotazu můžete umístit podmínku na dotaz, aniž byste znali přesné hodnoty, které chcete použít v porovnání.

Otázka: Jaká je skutečná výhoda při používání korelovaných poddotazů oproti běžným poddotazům?

Odpověď: Korelované poddotazy vám oproti standardním poddotazům nabízejí větší flexibilitu, protože můžete tabulky v poddotazu spojovat s tabulkami v hlavním dotazu. Podstatná je opět větší flexibilita k vytváření promyšlenějších dotazů.

Úkoly pro vás

Tato část nabízí kvízové otázky, které vám pomohou s upevněním získaných znalostí, a dále cvičení, jež vám poskytnou praktické zkušenosti s používáním osvojené látky. Pokuste se před nahlédnutím na odpovědi v příloze A odpovědět na otázky v kvízu a ve cvičení.

Kvíz

1. V části „Vnořování poddotazů“ vracel ukázkový poddotaz několik hodnot:

```
CykloSpec
CykloSpec
LX Obchůdek
LX Obchůdek
Cyklo Franta
```

   Některé z nich jsou tu dvakrát. Proč ve výsledné sadě tyto duplicity nejsou?

2. Jsou následující tvrzení pravdivá, či nepravdivá?
   a. Agregační funkce `SUM`, `COUNT`, `MIN`, `MAX` a `AVG` vracejí více hodnot.
   b. Maximální počet poddotazů, které lze vnořit, je dva.
   c. Korelované poddotazy jsou zcela nezávislé.
3. Bude následující poddotaz fungovat s níže uvedenými tabulkami `ORDERS` a `PART`?

```
SQL> SELECT *
FROM PART;

PARTNUM  DESCRIPTION     PRICE
-------  ------------  -------
     54  Pedály         542.50
     42  Sedla          245.00
     46  Pneu           152.50
     23  Horské kolo    3504.50
     76 Silniční kolo   5300.00
     10 Dvojkolo       12000.00
6 rows selected.
```

```
SQL> SELECT *
FROM ORDERS;

ORDEREDON    NAME          PARTNUM  QUANTITY  REMARKS
-----------  ------------  -------  --------  ---------
15-MAY-2006  Mega Kola          23         6  Zaplaceno
19-MAY-2006  Mega Kola          76         3  Zaplaceno
2-SEP-2006   Mega Kola          10         1  Zaplaceno
30-JUN-2006  Mega Kola          42         8  Zaplaceno
30-JUN-2006  CykloSpec          54        10  Zaplaceno
30-MAY-2006  CykloSpec          23         8  Zaplaceno
17-JAN-2006  CykloSpec          76        11  Zaplaceno
17-JAN-2006  LX Obchůdek        76         5  Zaplaceno
1-JUN-2006   LX Obchůdek        10         3  Zaplaceno
1-JUN-2006   Cyklo ABC          10         1  Zaplaceno
1-JUL-2006   Cyklo ABC          76         4  Zaplaceno
1-JUL-2006   Cyklo ABC          46        14  Zaplaceno
11-JUL-2006  Cyklo Franta       76        14  Zaplaceno
13 rows selected.
```

a.

```
SELECT * FROM ORDERS
WHERE PARTNUM =
SELECT PARTNUM FROM PART
WHERE DESCRIPTION = 'Mega Kola';
```

b.

```
SELECT PARTNUM
FROM ORDERS
WHERE PARTNUM =
(SELECT * FROM PART
 WHERE DESCRIPTION = 'LX Obchůdek');
```

c.

```
SELECT NAME, PARTNUM
FROM ORDERS
WHERE EXISTS
(SELECT * FROM ORDERS
 WHERE NAME = 'Mega Kola');
```

Cvičení

1. Představte si, že databázový systém MySQL nepodporuje poddotazy. Napište dva samostatné dotazy, které vrátí pole `NAME` a `ORDERON` z tabulky `ORDERS` pro ta jména, která jsou umístěná za jménem „Cyklo Franta“.

   První krok spočívá ve stanovení dotazu, jenž vytvoří výslednou sadu, která se použije při porovnání.

2. Napište dotaz, který zobrazí název nejdražší součástky.