MySQL subqueries using a veterinary hospital database

Yonathan Amare | yonathanamare1@gmail.com

```
-- Used a veterinary hospital database to formulate the

following queries using MySQL use veterinary_hospital;


-- Query 1: Finds which medications on the medications table
have a price greater than the  average price of all medications
on the table. When calculating the average, medications that
don't have a code are excluded. Displays the MedicationCode,
MedicationName, and displays a  formatted Price column so that
two decimal places display for all prices. Sorts the results by
Price from largest to smallest.

SELECT MedicationCode, MedicationName, FORMAT(Price, 2) AS Price

FROM medications

WHERE Price > (SELECT AVG(Price) FROM medications

WHERE MedicationCode != '0000') ORDER BY Price DESC;


-- Query 2: Shows which doctors have visits on the visits table
where the VisitType is "Surgery"  and the TotalDue of the visit
is  greater  than  the  average  TotalDue  of  all  visits  where
VisitType  = "Surgery".

SELECT doctors.DoctorID, doctors.LastName, doctors.FirstName,

visits.VisitNumber, visits.TotalDue FROM doctors

INNER JOIN visits ON doctors.DoctorID = visits.DoctorID

WHERE visits.VisitType = 'Surgery'

AND visits.TotalDue > (SELECT AVG(TotalDue) FROM visits

WHERE VisitType = 'Surgery') ORDER BY doctors.DoctorID;


-- Query 3a): Displays all visits on the visits table that have
```

```
at least one visit detail on the  visitdetails table.
VisitNumber, PetID, VisitDate, and TotalDue columns are displayed
and results  are sorted by recent to oldest VisitDate. Uses a
subquery that is introduced in a WHERE clause  and uses the IN
operator.

SELECT VisitNumber, PetID, VisitDate, TotalDue

FROM visits

WHERE VisitNumber IN (SELECT DISTINCT VisitNumber FROM
visitdetails)

ORDER BY VisitDate DESC;


--Query 3b) Same as the previous query but uses the NOT IN
operator.

SELECT VisitNumber, PetID, VisitDate, TotalDue

FROM visits

WHERE VisitNumber NOT IN (SELECT DISTINCT VisitNumber FROM
visitdetails)

ORDER BY VisitDate DESC;


-- Query 4a) Formulates a query that returns exact same
results as Query 3a but uses an inner  join instead.
SELECT v.VisitNumber, v.PetID, v.VisitDate, v.TotalDue

FROM visits v

INNER JOIN (SELECT DISTINCT VisitNumber FROM visitdetails) vd

ON v.VisitNumber = vd.VisitNumber ORDER BY v.VisitDate DESC;


--Query 4b) Same as 4a but uses a left outer join instead.

SELECT v.VisitNumber, v.PetID, v.VisitDate, v.TotalDue

FROM visits v

LEFT OUTER JOIN visitdetails vd ON v.VisitNumber = vd.VisitNumber

WHERE vd.VisitNumber IS NULL

ORDER BY v.VisitDate DESC;
```

```
-- Query 5a) Displays all visits on visits table where TotalDue
is less than TotalDue of any  visit found on visits table where
VisitType = “Surgery”. Displays VisitNumber, VisitDate, and
modified TotalDue column. Sorts results in TotalDue ascending
order. Uses a subquery with the ANY  keyword.

SELECT VisitNumber, VisitDate, TotalDue

FROM visits

WHERE TotalDue < ANY (SELECT TotalDue FROM visits

WHERE VisitType = 'Surgery') ORDER BY TotalDue;


-- Query 5b) Returns same result as 5a but uses MIN

aggregate function in the subquery SELECT VisitNumber,

VisitDate, TotalDue

FROM visits

WHERE TotalDue < (SELECT MIN(TotalDue) FROM visits

WHERE VisitType = 'Surgery') ORDER BY TotalDue;


-- Query 6) Displays all pets if their length is greater
than the average length for its  animaltype using a
correlated subquery.

SELECT AnimalType, PetID, PetName, DateOfBirth, Length

FROM pets p1

WHERE Length > (SELECT AVG(Length) FROM pets p2 WHERE

p2.AnimalType = p1.AnimalType) ORDER BY AnimalType,

Length;


-- This GROUP BY clause will check the above results.

SELECT AnimalType, avg(Length)
```

```
FROM pets p1

GROUP BY AnimalType
ORDER By AnimalType;


-- Query 7) Uses a correlated subquery in the SELECT clause to
display customers and count number  of pets owned. Subquery
counts number of pets on pets table as NumberofPets. Results are
sorted  by count from largest to smallest.

SELECT CustomerNumber, CustomerType, CustomerName,

(SELECT COUNT(*) FROM pets WHERE CustomerNumber =

customers.CustomerNumber) AS NumberofPets FROM customers

ORDER BY NumberofPets DESC;


-- Query 8) Creates a view for the previous query and names the
view customer_pet_counts.  Displays only customers who have a
pet and sorts results from smallest to highest CustomerNumber.

CREATE VIEW customer_pet_counts AS

SELECT CustomerNumber, CustomerType, CustomerName,

(SELECT COUNT(*) FROM pets WHERE CustomerNumber =

customers.CustomerNumber) AS NumberofPets FROM customers;


SELECT *

FROM customer_pet_counts

WHERE NumberofPets > 0

ORDER BY CustomerNumber;


-- Query 9) Returns pets that have had a visit in the year 2020
(VisitDate) and who have had a  follow up visit (FollowUpDate).
Uses correlated subquery through WHERE clause and EXISTS
operator. Sorts results in PetID ascending order.

SELECT PetID, PetName, AnimalType, Gender
```

```
FROM pets p
WHERE EXISTS (
 SELECT *
 FROM visits v
 WHERE v.PetID = p.PetID
 AND v.VisitDate BETWEEN '2020-01-01' AND '2020-12-31'
 AND v.FollowUpDate IS NOT NULL
)
ORDER BY PetID;
```

-- Query 10) Uses subquery in FROM clause to displays count of number of rows returned by Query 7. An easier way to do this query is adding using the aggregate function count in the SELECT clause.

```
SELECT COUNT(*)
FROM (
 SELECT CustomerNumber, CustomerType, CustomerName,
 (SELECT COUNT(*) FROM pets WHERE CustomerNumber =
customers.CustomerNumber) AS NumberofPets FROM customers
) AS counts;
```

-- Query 11) Calculates average number of visits per pet for each customer. Displays CustomerNumber and CustomerName from customers table, PetID from pets table, and NumberOfVisits which is the count of the number of visits of each pet found in PetID column from visits table.

```
SELECT c.CustomerNumber, c.CustomerName, p.PetID,
COUNT(*) AS NumberOfVisits FROM customers c
JOIN pets p ON p.CustomerNumber = c.CustomerNumber
JOIN visits v ON v.PetID = p.PetID
```

```
GROUP BY c.CustomerNumber, c.CustomerName, p.PetID;


-- Outer SELECT statement that uses first SELECT statement in
its FROM clause. Returns  CustomerNumber, CustomerName, and
average number of visits for each customer using “Average
Number of Visits Per Pet”. Final results sorted in
CustomerNumber ascending order.

SELECT CustomerNumber, CustomerName, ROUND(AVG(NumberOfVisits),
3) AS `Average Number of Visits  Per Pet`

FROM (

 SELECT c.CustomerNumber, c.CustomerName, p.PetID,

COUNT(*) AS NumberOfVisits  FROM customers c

 JOIN pets p ON p.CustomerNumber = c.CustomerNumber

 JOIN visits v ON v.PetID = p.PetID

 GROUP BY c.CustomerNumber, c.CustomerName, p.PetID

) AS visits_per_pet

GROUP BY CustomerNumber, CustomerName

ORDER BY CustomerNumber;


-- Query 12) Uses NOT EXISTS keyword to display all animal
types on the Animals table that are  not found on pets table.

SELECT animals

FROM animaltypes a

WHERE NOT EXISTS

     (SELECT *

 FROM pets

 WHERE a.animals = AnimalType);
```