INF1343, Winter 2011

Answers to Assignment 1: SQL Queries

Part A: Menagerie

Exercise 1

```
select owner
from pet
where species="dog"
order by weight desc
limit 1;
```

```
+-------+
| owner |
+-------+
| Diane |
+-------+
1 row in set (0.00 sec)
```

Notes:

- For some reason, some students forgot to limit this to dogs. Please read the problem description carefully!
- Here and in other problems involving string constants, the query would not work if you use “smart” quotes around “dog”. I.e., it should be:
  ```
  where species="dog"
  ```
  and not
  ```
  where species=“dog”
  ```
  This is not just a matter of style since your query simply wouldn't work. As far as MySQL is concerned, ",“, and ” and are all different characters. The first one of those is a legal way to delimit strings. The other two are not.
- Some students added up the weight of all dogs for each person. This is not what was asked.
- Some students used a subquery, along the lines of:
  ```
  select owner from pet
  where weight=(select max(weight) from pet where species="dog");
  ```
  This query would give the right result in some cases, but not always. It actually asks a different question: “Who owns a pet whose weight is the same as the weight of the heaviest dog?” This is *not* the same as asking “Who owns the heaviest dog”. Consider what happens when Harold gets himself a pet monkey whose weight just happens to also weigh 37.2 kg.

Exercise 2

```
select sum(weight)
from pet
where species="dog" and owner="Harold";
```

```
+-----------------+
| sum(weight)     |
+-----------------+
| 2.2999999523163 |
+-----------------+
1 row in set (0.00 sec)
```

Notes:

- Again, some students forgot to limit this to dogs.
- Some students added ROUND around the weight. That's a nice touch but wasn't expected. (No points were subtracted or adding for doing so.)

Part B: Child Care

Exercise 3

```
select phone
from centres
where fee_subsidy_available="Y" and infant_spaces > 20
order by infant_spaces;
```

```
+--------------------+
| phone              |
+--------------------+
| (416) 438-6880x233 |
| (416) 751-2358     |
| (416) 266-9755     |
| (416) 438-5989     |
+--------------------+
4 rows in set (0.00 sec)
```

Notes:

- I meant to ask you to order the results by the number of infant spaces *in descending order*, and some of you interpreted it this way. I accepted the answer with or without DESC.

Exercise 4

```
select count(*)
from centres
where name like "%Center%";
```

```
+----------+
| count(*) |
+----------+
|        3 |
+----------+
1 row in set (0.00 sec)
```

Notes:

- You can interpret the question as asking for a case-sensitive or a case insensitive match. However, searching for “%center% is incorrect: depending on how the database is configured, searching for “%center” might end up not catching “Center,” which is what you actually were asked about.
- Searching for “%enter” is clearly incorrect, since this would catch words like “renter.”
- Some of you tried to make sure to only get the cases where “Center” appears as a word rather than a part of a word. This is hard to do and wasn't expected, so no points were added or subtracted. (If you tried to only match for words and didn't quite do that right, I didn't subtract points either.)
- Counting the number of matches is a key part of this problem. Submission that did not use COUNT did not get full credit.
- COUNT(names) also works as an alternative to COUNT(*). (Actually, any field would work, but that would be bad style.)

Exercise 5

```
select ward, sum(toddler_spaces)
from centres
group by ward
order by sum(toddler_spaces) desc
limit 10;
```

```
+------+--------------------+
| ward | sum(toddler_spaces) |
+------+--------------------+
|   20 |                402 |
|   28 |                290 |
|   13 |                269 |
|   16 |                205 |
|   27 |                203 |
|   38 |                200 |
|    8 |                200 |
|    5 |                200 |
|   10 |                199 |
|   22 |                197 |
+------+--------------------+
10 rows in set (0.00 sec)
```

Notes:

- Using GROUP BY is essential in this case. Those who didn't lost most of the points.

Exercise 6

```
select ward
from centres
where auspice="Commercial Agency"
group by ward
having sum(toddler_spaces) > sum(total_spaces)*.25
order by sum(toddler_spaces) / sum(total_spaces) desc;
```

```
+------+
| ward |
+------+
|   28 |
|   13 |
|   20 |
|   27 |
+------+
4 rows in set (0.00 sec)
```

Notes:

- This cannot be correctly done without using HAVING. Also, note the order of the clauses.

Part C: Flying Blind

Exercise 7

```
select country_code
from cities
where population > 1000000
group by country_code
order by count(*) desc
limit 1;
```

Notes:

- This query would only work if the limit on the population is specified as 1000000. The following options are all wrong:

```
1,000,000 -- syntax error
1 000 000 -- syntax error
"1000000" -- string rather than number comparison
1000000000 -- that's a billion, not a million
```

- This problem requires using GROUP BY and COUNT. Those who didn't use those lost most of the points.

Exercise 8

```
select avg(elevation)
from cities
where country_code="CA" and name like "T%";
```

Notes:

- The proper 2-letter code for Canada is “CA”.
- The pattern needs to be “T%”, since the name must *start* with a “T”. “%T%” can catch cities such as “Victoria.”

Part D: Monkeys

Exercise 9

```
insert into monkeys values ("Chewbacca", 84.1, 1, "S");
insert into monkeys values ("Leia", 22.1, 1, "S");
insert into monkeys values ("Vader", 24.1, 1, "H");
insert into monkeys values ("Obiwan", 18.1, 1, "S");
insert into monkeys values ("Yoda", 12.3, 1, "H");
```

```
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)
```

Notes:

- Any reasonable values would be ok. I even accepted giant monkeys 500cm in size!
- Listing the columns after “monkeys” is a nice touch but wasn't expected.
- The five records could all be entered with a single query. No points were added or subtracted for this.
- The size and the number of bananas should be provided without quotes. If you put quotes around them, the query still works, but you are now relying on implicit casting. Specifying size as “81.1cm” might also work depending on how casting works, but you are really taking chances here, so this approach lost points. (Note that the database wouldn't care whether you say “81.1cm” or “81.1km”. It will just ignore the letters in the end — if you are lucky.)

Exercise 10

```
update monkeys set bananas=bananas+1 where mood="S";
```

```
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

Notes:

- It is not correct to just set the number of bananas to 2, since we can't be sure that each monkey still has 1 banana.

Part E: Diveshop

Exercise 11

```
select Name
from DIVECUST
join DIVEORDS
  on DIVECUST.Customer_No = DIVEORDS.Customer_No
where VacationCost > 20000;
```

```
+--------------+
| Name         |
+--------------+
| Lowell Lutz  |
| Ken Soule    |
| Marcus Selby |
| Sven Schiro  |
+--------------+
4 rows in set (0.00 sec)
```

Notes:

- For this and other problems in this part you were explicitly asked to use JOIN … ON … rather than JOIN … USING (…). This is because using JOIN … ON … requires more thinking and thus contributes more to learning. It also behaves more predictably. (See the notes on exercise 15 below.) Solutions using USING lost points.

Exercise 12

```
select Destination_Name
from DEST
join DIVEORDS
  on DIVEORDS.Destination_No = DEST.Destination_No
group by DEST.Destination_No
order by count(DEST.Destination_No) desc
limit 2;
```

```
+------------------+
| Destination_Name |
+------------------+
| Santa Barbara    |
| New Jersey       |
+------------------+
2 rows in set (0.00 sec)
```

Notes:

- New Jersey, Fiji and Cozumel are equally popular, so either is acceptable for the second slot. (Which one you got depended on the order of joins.) Some students decided to play it safe and limited the number of results to 4. This is not what was asked for, but I accepted this as valid.
- The assignment specifically asked to count multiple trips by the same person separately.
- Grouping by Destination_Name is *incorrect*. Several destinations may have the same name!

Exercise 13

```
select DIVEORDS.PaymentMethod
from DIVEORDS
join DIVECUST on DIVECUST.Customer_No=DIVEORDS.Customer_No
where DIVECUST.State_Prov!="CA"
group by DIVEORDS.PaymentMethod
order by count(*) desc
limit 1;
```

```
+---------------+
| PaymentMethod |
+---------------+
| Cash          |
+---------------+
1 row in set (0.00 sec)
```

Exercise 14

```
select DEST.Winter_Temp_C
from DEST
join DIVEORDS
  on DIVEORDS.Destination_No=DEST.Destination_No
join DIVECUST
  on DIVECUST.Customer_No=DIVEORDS.Customer_No
where DIVECUST.State_Prov="MI";
```

```
+---------------+
| Winter_Temp_C |
+---------------+
|        24.444 |
+---------------+
1 row in set (0.00 sec)
```

Exercise 15

The question

What destination has Blue Angelfish, is suitable for a beginner and would cost less than $3000?

The explanation

The query answers the question by joining four tables, then selecting rows with WHERE and finally projecting a single column. Three of those tables are joined because they have the fields based on which we do the selection: BIOLIFE has the Common_Name field, SITES has the Skill_Level field, and DEST has the Travel_Cost. BIOSITE is only used because we need it to link BIOLIFE to SITES. (If we didn't care about the skill level, we would still need to join use SITES to link DEST to BIOSITE.) BIOLIFE is joined to BIOSITES using the shared Species_No field. The resulting table has a Site_No field (from BIOSITE), which is used to join it to SITES. The resulting table then has a Destination_No field (from SITES), which we use when joining DEST.

How are the other queries different

Queries 2 pretty much does the same thing as query 1, just using shorter syntax, by taking advantage of the fact that the fields on which we join have the same name in both tables (for each of the three joins). Query 3 asks the database to the fields that have the same name. The output would be the same for the current database, but the results may be different if more columns are added. So, it's better to not use query 3. Query 4 is simply incorrect: it has no conditions on joins, so we would get back a cartesian product.

Which query is best

Either query 1 or query 2 should be used. Query 2 is shorter. Query 1 is more explicit. Choosing between them is a matter of style to some extent. Query 3 should be avoided. Query 4 is simply wrong.

Notes:

- If you said that, for example, query 1 works by joining SITES to DEST on destination number, this is not *quite* correct. SITES is joined to the result of all previous joins. However, no points were subtracted for this.
- Some students described some of the queries as being different either because they are “equi-joins” or because there are “SLQ-92” queries. This is incorrect. Queries 1, 2 and 3 are all SQL-92 equi-joins.
- Some of you wrote that query 2 would no longer work if the names of the fields change. This is true, but the same is true for query 1!
- Some of you described Query 4 as the “traditional JOIN” syntax. This is not correct. Query 4 is simply wrong. The “traditional” SQL JOIN relies on the WHERE clause to avoid getting a cartesian product. Query 4 does not do this.
- Strictly speaking, queries 1 and 2 are not 100% interchangeable. Depending on the database implementation, they can give us different set of columns. For example, the first JOIN in query 1 will give us a table that would have both the BIOLIFE.Species_No and the BIOSITE.Species_No. A join that uses USING may potentially replace those two columns with one.
- Consequently, queries using JOIN … ON … are somewhat more predictable. However, the choice between query 1 and query 2 is a matter of debate.