CSCB20 Worksheet – SQL and MySQL

1 Logging in to MySQL on cmslab

Open a PuTTY window (windows) or terminal (Mac) and type:

```
ssh utorid@cmslab.utsc.utoronto.ca
```

Enter your password.

Start MySQL by typing

```
mysql -u Utorid -pUtorid
```

You have your own database with name the same as your utorid. You can see what databases you have by typing:

```
show databases;
```

All commands in `MySQL` end in a `;`.

To use a particular database write the command:

```
use databaseName;
```

Where `databaseName` is your utorid.

You can now create tables and start making queries. You will set up some tables on your mathlab account in the tutorial.

To check which tables belong in your database, type:

```
show tables;
```

To see what columns belong to table `table_name`, type:

```
show columns from table_name;
```

2 Performing Simply Queries

We will use the University database tables for an example.

Selecting Columns:

- Select the name of each instructor from the instructor table.
  `SELECT name FROM instructor;`

- Select the name and department for each instructor.
  `SELECT name, dept_name FROM instructor;`

- Select the department for each instructor.
  `SELECT dept_name FROM instructor;`

  Was this the answer we wanted? Let's fix it... `SELECT DISTINCT dept_name FROM instructor;`

- Maybe we would like to see the instructor names and departments *ordered* by department name.

  `SELECT name, dept_name FROM instructor ORDER BY dept_name;`

- Select all columns from the course relation.
  `SELECT * FROM course;`

Selecting Columns With Conditions

- Select the names of the instructors in the '`Comp. Sci`' department.

  `SELECT name FROM instructor WHERE dept_name = 'Comp. Sci';`

- Select the names and salaries of the instructors with salary at least $65,000 ordered by increasing salary.

  `SELECT name, salary FROM instructor WHERE salary >= 65000 ORDER BY salary;`

  How about in decreasing salary order?

  ```
  SELECT name, salary FROM instructor WHERE salary >= 65000 ORDER BY salary
  DESC;
  SELECT name, salary FROM instructor WHERE salary >= 65000 ORDER BY -salary;
  ```

  Key words are `ORDER BY ... [ASC or DESC]`.
- Select the course_id of courses offered in the Spring of 2010.

  `SELECT course_id FROM section WHERE semester = 'Spring' and year = '2010';`

Selecting From Multiple Tables – Natural Join:

- Select all columns from the natural join of instructor and teaches.

  `SELECT * FROM instructor NATURAL JOIN teaches;`

- Select the name, course_id and course title of all courses taught in 2010 ordered by course_id.

  ```
  SELECT name, course_id, title FROM instructor NATURAL JOIN teaches NATURAL
  JOIN course WHERE year = '2010' ORDER BY course_id;
  ```

  Can we use a `NATURAL JOIN` to find all those departments that share the same building? NO...

Selecting from Multiple Tables - Inner Join:

- Select all departments and their buildings that share a building.
  ```
  SELECT A.dept_name, A.building FROM department A INNER JOIN department B
  ON A.building = B. building
  AND A.dept_name != B.dept_name
  ORDER BY building;
  ```

- Select the instructor_id of instructors who teach more than one course at the same time.
  ```
  SELECT A.id FROM teaches A INNER JOIN teaches B
  ON A.semester = B.semester
  AND A.year = B.year
  AND A.course_ id != B.course_id
  AND A.id = B.id;
  ```

  Something isn't quite correct...missing `id 83821`...:

  ```
  SELECT DISTINCT A.id FROM teaches A INNER JOIN teaches B
  ON A.semester = B.semester
  AND A.year = B.year
  AND (A.course_ id != B.course_id OR A.sec_id != B.sec_id)
  AND A.id = B.id;
  ```

- Select the name and course id of all instructors that teach more than one course at a time.

  ```
  SELECT DISTINCT name, A.course_id FROM teaches A INNER JOIN teaches B INNER
  JOIN instructor
  ON A.semester = B.semester AND
  A.year = B.year AND
  (A.course_id != B.course_id OR A.sec_id != B.sec_id) AND
  A.id = B.id AND
  A.id = instructor.id;
  ```

  Is your head starting to spin?

Q. What is the difference between `SELECT * FROM A JOIN B ON Conditions` and `SELECT * FROM A JOIN B WHERE Conditions`?

A. ON is used to restrict the join of the tables reducing the size of the table whose rows can then be pruned using WHERE. If we only use WHERE conditions then the JOIN is just a cartesian product that makes a huge table that then has to be pruned using where conditions - very inefficient.

2.1 Using Aggregate Functions

SQL has aggregate functions `AVG, SUM, COUNT, MIN, MAX` which can be applied to columns in the select statement.

Selecting Columns with Aggregate Function:

- Select the average salary of all instructors.
  ```
  SELECT AVG(salary) FROM instructor;
  ```

  Notice that the column header shows the aggregate function. Perhaps we would rather name this title '`Average Salary`'. We can do this with the `AS` clause.
  ```
  SELECT AVG(salary) AS ‘Average Salary’ FROM instructor;
  ```

- Select the average salary of all instructors in the computer science department.
  ```
  SELECT AVG(salary) AS ‘Average Salary’ FROM instructor WHERE dept_name = ‘Comp.
  Sci.’;
  ```

- Select the department and the departments average salary.
  ```
  SELECT dept_name, AVG(salary) AS ‘Average Salary’ FROM instructor GROUP BY
  dept_name;
  ```

  `GROUP BY` groups the rows according to the attribute specified and then any aggregate functions are applied to each group.

- Find the total number of instructors who teach a course in the Spring 2010 semester.
  ```
  SELECT COUNT(id) FROM teaches WHERE semester=’Spring’ AND year=2010;
  ```

  This gives us the wrong answer...want `DISTINCT` instructors.

  ```
  SELECT COUNT(DISTINCT id) FROM teaches WHERE semester=’Spring’ AND year=2010;
  ```

  Use `DISTINCT` whenever you do not want the duplicates to be included. With the `AVG` function we want to include all duplicates, with `COUNT` sometimes we don't.

- Select the department name and average salary for instructors for each department. Include only those departments whose salaries are greater than $42000.

  ```
  SELECT dept_name, AVG(salary) AS avg_salary FROM instructor GROUP BY dept_name
  HAVING AVG (salary) > 42000;
  ```
  The HAVING cause is a condition that applies to *groups* rather than *tuples.* The HAVING clause is applied *after* the groups have been formed.

Q. In what order are the SQL commands applied to a relation to create the smaller relation?

1. FROM
2. The predicate in the WHERE clause (if present) is applied to tuples satisfying the FROM clause.
3. Tuples satisfying the WHERE predicate are then placed into groups by the GROUP BY clause (if present).
4. The HAVING clause (if present) is applied to each group. Groups not satisfying the HAVING clause are removed.
5. The SELECT clause is applied to the remaining groups, applying aggregate functions to get the resulting tuple for each group.

- For each course section offered in 2009, find the average total credits (tot_cred) of all students enrolled in the section, if the section had at least 2 students.

```
SELECT course_id, semester, year, sec_id, AVG(tot_cred) FROM takes NATURAL
JOIN student WHERE year = 2009 GROUP BY course_id, semester, year, sec_id HAVING
COUNT(ID) >= 2;
```

- Find all the courses taught in the both the Fall 2009 and Spring 2010 semesters.

```
SELECT course_id FROM section WHERE semester='Spring' AND year='2010' AND
course_id IN (SELECT course_id FROM section WHERE semester='Fall' AND year='2009');
```

- Find all the courses taught in the Fall 2009 but not in Spring 2010 semesters.

```
SELECT course_id FROM section WHERE semester='Spring' AND year='2010' AND
course_id NOT IN (SELECT course_id FROM section WHERE semester='Fall' AND year='2009')
```

`IN` or `NOT IN` can be use to check if an attribute belongs to a tuple. For example:

- Find the names of the instructors whose names are neither 'Mozart' nor 'Einstein'.

```
SELECT name FROM instructor WHERE name NOT IN ('Mozart', 'Einstein');
```

We can also check whether a tuple of attributes belongs in a relation using `IN` or `NOT IN`.

- Find the total number of distinct students who have taken course sections taught by the instructor with `ID 10101`.

```
SELECT COUNT (DISTINCT ID) FROM takes WHERE
(course_id, sec_id, semester, year) IN (SELECT course_id, sec_id, semester,
year FROM teaches WHERE teaches.ID = 10101);
```

- Find all the students whose names contain "an" in them.

```
SELECT name FROM student WHERE
name LIKE '%an%';
```

- Find all the students whose names are 4 characters long.

```
SELECT name FROM student WHERE
name LIKE '_ _ _ _';
```

Use the `LIKE` or `NOT LIKE` condition to search for strings that match. The % matches anything and the _ matches exactly one character.

- We know that we can SELECT ... FROM any relation. This means that the FROM clause can be the result of a SELECT statement itself.

Find the average instructors? salaries of those departments where the average salary is greater than $42,000 (*without using a* `HAVING` clause.

Can think of this as first, lets get a relation containing the departmental average salaries and then lets select from it.

```
SELECT dept_name, avg_salary
FROM (SELECT dept_name, AVG(salary) AS avg_salary FROM instructor GROUP BY
dept_name) AS T
WHERE T.avg_salary > 42000;
```

3 University Relations

Relations and their schemas:

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Teaches

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Instructor

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Section

course_id	*prereq_id*
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Prereq

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Department

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Course