|  |  |  |  |
| --- | --- | --- | --- |
| **Lab Week 10 Report** | | | |
| **Class** | Database | **Group ID** | 11 |
| **Student ID** | 12151302 | **Name** | LEE HAESEONG |
| **Student ID** | 12151310 | **Name** | JANG HYOJUN |
| **Student ID** | 12171479 | **Name** | KIM HYUNJIN |
| **Student ID** | 12181465 | **Name** | KIM JAEWUK |
| **Lab Topic** | SQL Exercises with group members | | |

**Task 1: Find the highest salary of any instructor**

|  |
| --- |
| * **SQL Code**           select id, name from instructor //select id, name at instructor entity       where salary = (select max(salary) from instructor); //whose salary is max   * **Analysis**      Einstein’s salary is maximum value at ‘95000’. So Einstein’s id and name will be shown.   * **Result** |

**Task 2: Find the IDs of all students who were taught by an instructor named Einstein. (Make sure there are no duplicates in the result)**

|  |
| --- |
| * **SQL Code**   SELECT distinct student.ID  FROM (student join takes using(ID)) //use distinct ~ from ~ to remove duplication  JOIN (instructor join teaches using(ID))  USING (course\_id, sec\_id, semester, year)  WHERE instructor.name = ‘Einstein’;   * **Analysis**   SQL code shown above results in students who were taught by the instructor named ‘Einstein’. Four tables , ‘student’ and ‘takes’, ‘instructor’ and ‘teaches’,  are merged respectively to show us the tuples which have the same ‘ID’ attribute. On the last part of the code, which is ‘WHERE’, finally get the student's ID asked in the question.  In data insertion code…    Einstein’s ID is ‘22222’.    Plus, Einstein is associated with only one course ‘PHY-101’ in Fall, 2009.    Peltier whose student id is ‘44553’ is a guy who only took ‘PHY-001’ course in Fall, 2009. So the result we have got is correct.   * **Result** |

**Task 3: Find the enrollment of each section that was offered in Autumn 2009**

|  |
| --- |
| * **SQL Code**            select course\_id, sec\_id, count(\*) as enrollment        from section natural join takes //use natural join to avoid duplication        where semester=‘fall’ and year=‘2009’        group by course\_id, sec\_id; //‘group by’ notifies how many by type   * **Analysis**      We can see the list of fall 2009 with simple code. (The question say autumn but in database, there was only fall semester, so we replace autumn by fall.) There is 6 CS-101 course\_id of fall 2009, so it’s enrollment will be 6. The other course\_id(CS-347 and PHY-101) are the same way.   * **Result** |

**Task 4: Find the maximum enrollment, across all sections, in Autumn 2009**

|  |
| --- |
| * **SQL code**     SELECT max(enrollment)  from (select count(ID) as enrollment  from section natural join takes //use natural join to avoid duplication  where semester = ‘FALL’ and year = 2009  group by course\_id, sec\_id) c;   * **Analysis**   SQL code shown above makes an artificial attribute, max(enrollment). This attribute shows us a maximum enrollment across all sections. ‘COUNT’ is an API which helps us to count the data registered in the table. Since we want to know a maximum number of students who have enrolled in the section among all section open in Autumn, 2009, this is a reason for using an artificial attribute. At first when we were searching for sections which open in Autumn, there was a problem. All the sections open in Autumn were inserted as ‘Fall’ instead of ‘Autumn’ so we used ‘Fall’ to look for the sections.  In SQL data insertion code...            ‘CS-101’ and ‘CS-347’ are the sections open in Autumn, 2009 as shown above.            There were 6 students enrolled in ‘CS-101’ and 2 students in ‘CS-347’.            Because we want to know a maximum enrollment in Autumn, 2009, the result we have got is correct.   * **Result** |

**Task 5: Find the sections that had the maximum enrollment in Autumn 2009**

|  |
| --- |
| * **SQL Code**             select course\_id, sec\_id          from         (select course\_id, sec\_id, count(\*) as enrollment         from takes         where semester=‘fall’ and year =‘2009’         group by course\_id, sec\_id) as new\_table //make new table to compare each other.         where enrollment >= all //and compare         (select count(\*) as en         from takes         where semester=‘fall’ and year=‘2009’         group by course\_id, sec\_id);   * **Analysis**   First, we try to use max code with enrollment    having count(\*)=(select max(enrollment)  with this code, but enrollment is what we make with previous code, so error was occurred. So at first, we choose the list of fall 2009 and create this list as new table. And next, we compare the number of enrollment in this new table with a sign of inequality(≧).  When we search course\_id and sec\_id of fall 2009, the result is below.    We can see 6 CS-101 course\_ids, 2 CS-347 course\_ids, and 1 PHY-101 course\_id. (Actually, We can also see this result at Task3.) So the result will be shown with CS-101 course\_id and sec\_id.   * Result |

**Task 6: Increase the salary of each instructor in Comp. Sci. department by 10%**

|  |
| --- |
| * **SQL code**   update instructor //use update code to edit the data in instructor entity  set salary = salary\*1.10  where dept\_name  = ‘Comp. Sci. ‘;  select \* from instructor;   * **Analysis**   SQL code shown above literally increases the salary of all instructors who are associated with the Computer Science department. ‘salary’ attribute is saved as the value multiplied by 1.10 recursively.    Initial salary of the instructor is shown above. The result we’ve got shows the instructor table where updated salary of the Computer Science department instructors only.   * **Result** |

**Task 7: Delete all courses that have never been offered (that is, do not occur in the section relation)**

|  |
| --- |
| * **SQL Code**   delete from course //delete course\_id from course entity  where course\_id not in (select course\_id from section); //which is not in the section entity.   * **Analysis**     Left one is 12 course\_ids which are in the section entity. Right one is 13 course\_ids which are in the course entity. The number of list is different because of BIO-399 course\_id. So with the below SQL code, we can delete 1 row.   * **Result** |
|  |

**Task 8: Insert every student whose tot\_cred attribute is greater than 100 as an instructor in the same department. With salary of $10,000**

|  |
| --- |
| * **SQL code**   insert into instructor //use insert into to insert new data to entity  select ID, name, dept\_name, 10000  from student  where tot\_cred > 100; //write tot\_cred conditional sentence  select \* from instructor;   * **Analysis**   SQL code shown above is adding new instructors among students whose total credit is greater than 100 with 10k salary. Before we executed the above SQL code, there had been a job before it was taken. There was problem with a constraint shown below.    All felt dealing with SQL is cumbersome and hardly found a way of finding information about constraint, we just simply deleted that constraint just like this.    After this, there weren’t any problems and we successfully added ‘Chavez’ and ‘Zhang’ as new instructors who fulfil given conditions.   * **Result** |