CS 557 Assignment3

This assignment is split into two problems. The first problem concerns the practice of SQL queries to interrogate and to update a database. The second problem involves questions that require reading about File Structures, Indexing, and Hashing.

Group Members:

Yi Ren *(002269013)* Wentao Lu *(002276355)* Junjia Lin *(002268506)* Luyun Nie *(002268087)*

Dependency

- pgAdmin
- PostgreSql

Problem 1

- **1. Find the names of all students who have taken at least one Comp. Sci. course; make sure there are no duplicate names in the result.**

```
/*Question 1: */
select distinct student.name from student
where student.id in (
    select distinct takes.id from takes
    where takes.course_id in(
        select course.course_id from course
        where course.dept_name like 'Comp. Sci.'));
```

- **2. Find the IDs and names of all students who have not taken any course offering before Spring 2009.**

```
/* Question 2: */
select distinct student.id, student.name from student
where student.id not in(
    select takes.id from takes where takes.year < 2009 );
```

- **3. For each department, find the maximum salary of instructors in that department. You may assume that every department has at least one instructor.**

```
/* Question 3: */
select instructor.dept_name, max(instructor.salary) from instructor
group by instructor.dept_name;
```

- **4. Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query.**

```
/* Question 4: */
select instructor.dept_name, max(instructor.salary) from instructor
group by instructor.dept_name
order by max(instructor.salary) asc limit 1;
```

- **5. Create a new course "CS-001", titled "Weekly Seminar",with 0 credits.**

```
/* Question 5: 0 credit couldn't be inserted since the constraints*/
/* Unless we DISABLE or DROP the CONSTRAINT*/
insert into course values ('CS-001','Weekly Seminar',null,1);
```

- **6. Create a section of this course in Autumn 2009, with sec id of 1.**

```
/* Question 6: */
insert into public.section values('CS-001', '1', 'Fall', '2009',null,null,null);
```

- **7. Enroll every student in the Comp. Sci. department in the above section.**

```
/* Question 7 : */
Insert into takes(id,course_id,sec_id,semester,year)
Select id,'CS-001','1','Fall',2009 From student
where dept_name like 'Comp. Sci.';
```

- **8. Delete enrollments in the above section where the student's name is Chavez.**

```
/* Question 8: */
Delete from takes
Where course_id ='CS-001' and sec_id = '1' and semester='Fall' and year =2009
and id in(
    select id from student
    where name like '%Chavez%');
```

- **9. Delete the course CS-001. What will happen if you run this delete statement without first deleting offerings (sections) of this course?**
  If we delete the course CS-001 without deleting section of this course, the change would be cascaded to the data related in table 'section' 'prereq' and 'take', since we have `on delete cascade` referential action for the foreign key in the tables mentioned.

```sql
/* Question 9: */
delete from course
where course_id  = 'CS-001';
```

- **10. Delete all takes tuples corresponding to any section of any course with the word “database” as a part of the title; ignore case when matching the word with the title.**

```sql
/* Question 10: */
delete from section
where course_id in (
    select course_id from course
    where lower(title) like '%database%'
);
```

- **11. Suppose that we have a relation marks(ID, score) and we wish to assign grades to students based on the score as follows: grade F if score < 40, grade C if 40 ≤ score < 60, grade B if 60 ≤ score < 80, and grade A if 80 ≤ score. Write SQL queries to do the following:**

```
a. Display the grade for each student, based on the marks relation.
```

```sql
/* Question 11-a: */
SELECT id, score,
    CASE
        WHEN score < 40 THEN 'F'
        WHEN 40 <= score and score < 60 THEN 'C'
        WHEN 60 <= score and score < 80 THEN 'B'
        WHEN 80 <= score THEN 'A'
    END AS grade FROM marks;
```

```
b. Find the number of students with each grade.
```

For this question, we implemented our query with 2 method: WITH and UPDATE.

```sql
/* Method - 1: WITH*/
WITH temp_marks AS (
    SELECT id, score,
    CASE
        WHEN score < 40 THEN 'F'
        WHEN 40 <= score and score < 60 THEN 'C'
        WHEN 60 <= score and score < 80 THEN 'B'
        WHEN 80 <= score THEN 'A'
    END AS grade FROM marks
```

```
)
SELECT grade, COUNT(grade) FROM temp_marks GROUP BY grade ORDER BY grade;
```

We need to add a new COLUMN or change the type of COLUMN `score` from numeric to varchar before UPDATE.

```
/* Method - 2: UPDATE*/
ALTER TABLE marks
ADD COLUMN grade VARCHAR;
UPDATE marks SET grade =
    CASE
        WHEN score < 40 THEN 'F'
        WHEN 40 <= score and score < 60 THEN 'C'
        WHEN 60 <= score and score < 80 THEN 'B'
        WHEN 80 <= score THEN 'A'
        END;
SELECT grade, COUNT(grade) FROM marks GROUP BY grade ORDER BY grade;
```

- **12. The SQL like operator is case sensitive, but the lower() function on strings can be used to perform case insensitive matching. To show how, write a query that finds departments whose names contain the string "sci" as a substring, regardless of the case.**

```
/* QUestion 12: */

select * from department
where lower(dept_name) like '%sci%';
```

Problem 2

- **1. What is the difference between primary and secondary storage?**

1. The `primary storage` is always referred to the storage that could be *volatile*, for example, RAM and CPU main memory, they tend to *lose data* when computer loses power. On the contrary, the `secondary storage`, such as magnetic disks, flash memory and SSD(solid-state drives) can store data more *reliably*.
2. The `primary storage` has lower *capacity* and higher *price* than `secondary storage`.

- **2. Why are disks, not tapes, used to store online database files?**
  We can easily fetch or update data on the `disk`, since the read/write head could skip between tracks *uncontinuously*, however, when it comes to `tapes`, it'll be mounted and scanned until required block is under read/write head, which might be *time-consuming*.

- **3. Discuss the process of disk initialization.**
  For the `disk initialization`, the operating system divides the track into *fixed equal-sized blocks*, which cannot be changed dynamically. A disk with hard-coded sectors often has the sectors *subdivided*

or *combined* into blocks during initialization. Also there could be *interblock gaps* between blocks, which contains some special control information written during disk initialization.

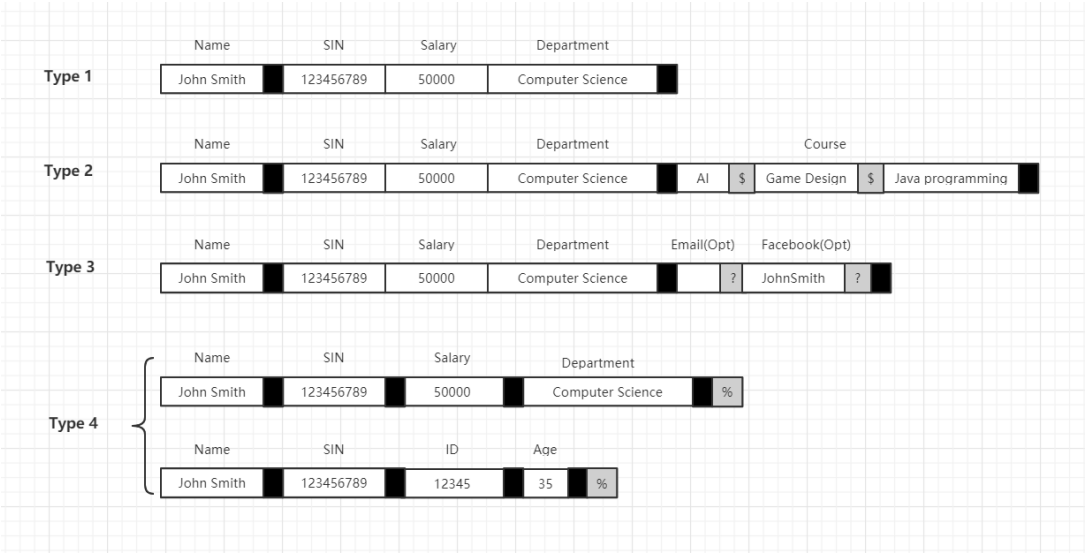
- **4. Why is accessing a disk block expensive? Discuss the time components involved in accessing a disk block.**
  Given the address of data, totally we need:

1. `Seek time`: the read/write head should be placed on the *correct track* by our disk controller.
2. `Rotational delay`: the designated block might need some time to *rotate* under the header.
3. `Block transfer time`: the additional time that we need to *transfer* our data.

- **5. What are the reasons for having variable-length records? What types of separator characters are needed for each?**
  Reasons:

1. One or more fields could be of *varying size*, even the file records has the same type.
2. One or more fields could have *multiple valuse*, even the file records has the same type.
3. One or more fields could be *optional*, even the file records has the same type.
4. The file could have different kind or record types.

Separator characters:
To determine the bytes within a particular record that represent each field, we can use special separator characters to terminate variable-length fields.
Type 1: Only need a seperating character to seperate different field.
Type 2: Need the seperating character at the end of field and also character to seperate multiple values.
Type 3: Need the seperating character at the end of field and also character to seperate optional values.
Type 4: Need the seperating character at the end of field and also terminate character at the end of records.



- **6. Discuss the techniques for allocating file blocks on disk.**

1. `Contiguous allocation`: the blocks on disk are allocated one by one(contiguously), in this way, we could read the file easily, which might be even faster with double buffering, however, this technique can hardly expand since there's no more room adjacent to the allocated block.

2. `Linked allocation`: the blocks are not allocated contiguously, we have a pointer which points to the next block in every block. On the one hand, it could be much easier to expand the file than Contiguous allocation, on the other hand, the pointers make it slower to read the whole file.
3. `Indexed allocation`: we have one or more index blocks containing pointers to the actual file blocks.

- **7. What is the difference between a file organization and an access method?**
  The `file organization` refers to how the records and blocks are stored on the disk or other storage medium, while the `access method` refers to how the records can be accessed, which might be different according to the way of the file organization.

- **8. What is the difference between static and dynamic files?**
  The `static file` can hardly be updated, while we could perform more update operations on the `dynamic files`.