CS307 Presentation Material (Final)

1. Implementation of `searchCourse`

(1) Function in Database

```
create function search_course(search_student_id integer, search_semester_id integer, search_course_id character varying, search_name character varying, search_instructor character varying, search_day_of_week integer, search_class_time integer, search_class_location character varying[], search_course_type integer, ignore_full boolean, ignore_conflict boolean, ignore_passed boolean, ignore_missing_prerequisites boolean, page_size integer, page_index integer) returns SETOF record
    language plpgsql
as
$$
declare
    empty_location bool := (array_length(search_class_location, 1) = 0);
begin
    return query
        with conflict_sections as (
            select *
            from get_all_conflict_sections(search_student_id,
                                           search_semester_id) as (section_id int)
        )
        select available_sections.course_id,
               available_sections.course_name,
               available_sections.credit,
               available_sections.class_hour,
               available_sections.is_pf_grading,
               available_sections.section_id,
               available_sections.section_name,
               available_sections.total_capacity,
               available_sections.left_capacity,
               csc2.id,
               i2.id,
               i2.full_name as ins_full_name,
               csc2.day_of_week,
               csc2.week_list,
               csc2.class_begin,
               csc2.class_end,
               csc2.location
        from (select distinct c.id    as course_id,
                              c.name  as course_name,
                              c.credit,
                              c.class_hour,
                              c.is_pf_grading,
                              cs.id   as section_id,
                              cs.semester_id,
                              cs.name as section_name,
                              cs.full_name,
                              cs.total_capacity,
                              cs.left_capacity
```

```
                from course c
                         join course_section cs
                              on c.id = cs.course_id
                         join course_section_class csc
                              on cs.id = csc.section_id
                         join semester s
                              on s.id = cs.semester_id
                         join instructor i
                              on csc.instructor_id = i.id
                where s.id = search_semester_id
                  and case search_course_id is null
                          when true then true
                          when false then
                              position(search_course_id in c.id) > 0
                      end
                  and case search_name is null
                          when true then true
                          when false then
                              position(search_name in cs.full_name) > 0
                      end
                  and case search_instructor is null
                          when true then true
                          when false then (
                           position(search_instructor in i.other_name) = 1
                           or position(search_instructor in i.last_name) = 1
                           or position(search_instructor in i.full_name) = 1)
                      end
                  and case search_day_of_week is null
                          when true then true
                          when false then
                              csc.day_of_week = search_day_of_week
                      end
                  and case search_class_time is null
                          when true then true
                          when false then search_class_time between
                              csc.class_begin and csc.class_end
                      end
                  and case (search_class_location is null or empty_location)
                          when true then true
                          when false then
                              match_location(csc.location,
search_class_location)
                      end
                  and case search_course_type
                    -- ALL
                          when 1 then true
                    -- MAJOR_COMPULSORY
                          when 2 then c.id in (
                              select course_id
                              from major_course_relations mcr
                                       join student s2
                                            on mcr.major_id = s2.major_id
                              where s2.id = search_student_id
                                and mcr.is_compulsory
                          )
                    -- MAJOR_ELECTIVE
                          when 3 then c.id in (
                              select course_id
```

```
                            from major_course_relations mcr
                                     join student s2
                                          on mcr.major_id = s2.major_id
                            where s2.id = search_student_id
                              and not mcr.is_compulsory
                        )
                    -- CROSS_MAJOR
                        when 4 then c.id in (
                            select distinct mcr.course_id
                            from major_course_relations mcr
                            where mcr.course_id not in (
                                select course_id
                                from major_course_relations mcr2
                                         join student s3
                                              on mcr2.major_id =
s3.major_id
                                where s3.id = search_student_id
                            )
                        )
                    -- PUBLIC
                        when 5 then c.id not in (
                            select distinct course_id
                            from major_course_relations
                        )
                    end
                and case ignore_full
                        when false then true
                        when true then cs.left_capacity > 0
                    end
                and case ignore_conflict
                        when false then true
                        when true
                            then cs.id not in (
                            select section_id
                            from conflict_sections
                        )
                    end
                and case ignore_passed
                        when false then true
                        when true then c.id not in (
                            select distinct cs1.course_id
                            from student_section_relations ssr
                                     join course_section cs1
                                          on cs1.id = ssr.section_id
                            where (ssr.grade = -1 or ssr.grade >= 60)
                              and ssr.student_id = search_student_id
                        )
                    end
                and case ignore_missing_prerequisites
                        when false then true
                        when true then
                            judge_prerequisite(search_student_id, c.id)
                    end
            order by course_id, cs.full_name
            limit page_size offset page_index * page_size)
available_sections
                 join course_section_class csc2
                      on available_sections.section_id = csc2.section_id
```

```
                join instructor i2
                     on i2.id = csc2.instructor_id
        order by course_id, available_sections.full_name;
end;
$$;
```

Code 1. Function of course searching

(2) Method in Java

```
conn = sds.getSQLConnection();
pStmt = conn.prepareStatement("select * from search_course(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?) as (course_id varchar, course_name varchar, credit int, class_hour int, is_pf_grading bool, section_id int, section_name varchar, total_capacity int, left_capacity int,  class_id int, ins_id int, ins_name varchar, day_of_week int, week_list int, class_begin int, class_end int, location varchar)");
```

Code 2. Call function in Java

2. Implementation of `enrollCourse`

(1) Function in Database

```
create function enroll_course(stu_id integer, sec_id integer) returns integer
    language plpgsql
as
$$
declare
    temp_course varchar;
    temp_bool   bool;
    left_cap    int;
begin
    -- NO STUDENT (UNKNOWN ERROR)
    if not exists(select null from student where id = stu_id) then
        return 0;
    end if;

    if not exists(select null from course_section where id = sec_id) then
        -- COURSE_NOT_FOUND
        return 1;
    end if;

    if exists(select grade
              from student_section_relations
              where student_id = stu_id
                and section_id = sec_id) then
        -- ALREADY_ENROLLED
        return 2;
    end if;

    select course_id, left_capacity
    into temp_course, left_cap
```

```
    from course_section
    where id = sec_id for update;

    if exists(select null
              from student_section_relations ssr
                       join course_section cs on cs.id = ssr.section_id
              where cs.course_id = temp_course
                and (ssr.grade = -1 or ssr.grade >= 60)
                and ssr.student_id = stu_id) then
        -- ALREADY_PASSED
        return 3;
    end if;

    select judge_prerequisite(stu_id, temp_course) into temp_bool;

    if not temp_bool then
        -- PREREQUISITE_NOT_FULFILLED
        return 4;
    end if;

    select detect_conflict(stu_id, sec_id) into temp_bool;
    if temp_bool then
        -- COURSE_CONFLICT_FOUND
        return 5;
    end if;


    if left_cap <= 0 then
        -- COURSE_IS_FULL
        return 6;
    end if;

    update course_section set left_capacity = (left_capacity - 1) where id =
sec_id;
    insert into student_section_relations (student_id, section_id) values
(stu_id, sec_id);
    -- SUCCESS
    return 7;

end;
$$;
```

Code 3. Function of enrolling course

We used keyword `for update` to make sure that the left capacity of the section we want to enroll will keep the same until we update it.

(2) Design of Prerequisite

We used post-order expression to represent prerequisite. The prerequisite will be first transformed into a String using the method in Code 3.

```
pCase = new Cases<>() {
  StringBuilder temp;

  @Override
  public String match(AndPrerequisite self) {
```

```
    String[] children = self.terms.stream()
        .map(term -> term.when(this))
        .toArray(String[]::new);
    temp = new StringBuilder(children[0]);
    for (int i = 1; i < children.length; i++) {
      temp.append("|").append(children[i]).append("|AND");
    }
    return temp.toString();
  }

  @Override
  public String match(OrPrerequisite self) {
    String[] children = self.terms.stream()
        .map(term -> term.when(this))
        .toArray(String[]::new);
    temp = new StringBuilder(children[0]);
    for (int i = 1; i < children.length; i++) {
      temp.append("|").append(children[i]).append("|OR");
    }
    return temp.toString();
  }

  @Override
  public String match(CoursePrerequisite self) {
    return self.courseID;
  }
};
```

Code 4. Transforming prerequisite object to a String

After this, the prerequisite `(MAE308 OR RD412 OR RD463) AND MSE201` will be changed into a String like `MAE308|RD412|OR|RD463|OR|MSE201|AND`.

```
create procedure add_course(cour_id character varying, cour_name character varying, cour_credit integer, cour_ch integer, cour_pf boolean, cour_pre character varying)
    language plpgsql
as
$$
declare
    temp_array varchar[];
    i          int;
begin
    insert into course (id, name, credit, class_hour, is_pf_grading)
    values (cour_id, cour_name, cour_credit, cour_ch, cour_pf);

    if cour_pre is not null then
        select regexp_split_to_array(cour_pre, E'\\|') into temp_array;
        for i in 1 .. array_length(temp_array, 1)
            loop
                if temp_array[i] = 'AND' then
                    insert into course_prerequisite_relations(course_id, prerequisite_id, and_logic)
                    VALUES (cour_id, null, true);
                elseif temp_array[i] = 'OR' then
                    insert into course_prerequisite_relations(course_id, prerequisite_id, and_logic)
```

```
                    VALUES (cour_id, null, false);
                else
                    insert into course_prerequisite_relations(course_id,
prerequisite_id, and_logic)
                    VALUES (cour_id, temp_array[i], null);
                end if;
            end loop;
    end if;
end;
$$;
```

Code 5. Procedure add_course

Then we will send the String into database and the database will add the prerequisite into the table `course_prerequisite_relations` in procedure `add_course`. The procedure `add_course` is shown in Code 4. We uses a individual table to store the prerequisites instead of directly store the post-order expression in the table `course`. This is mainly because that if a course is deleted, then we need to delete it from the prerequisite expression. If we store the expression in a varchar type, then we can not add foreign key in the String and as a result, we cannot use cascade delete in SQL to delete the course in the expression, and it will cost a lot to search all the expressions, find the course id and delete it. So finally we decided to store the prerequisite post-order expression in a new table and use multiple records to represent one expression.

	id	course_id	prerequisite_id	and_logic
26	26	ME304	MAE308	<null>
27	27	ME304	RD412	<null>
28	28	ME304	<null>	false
29	29	ME304	RD463	<null>
30	30	ME304	<null>	false
31	31	ME304	MSE201	<null>
32	32	ME304	<null>	true

Figure 1. Prerequisite records

Then we can get prerequisite records like Figure 1. if `prerequisite_id` is null, then this record represents a Boolean operator in the expression, and if `and_logic` is true then it's `AND` else it's `OR`; if `and_logic` is null, the record represents a prerequisite. One of the two columns must be null and the other must not be `null`. We maintain this by a **check constraint** shown in Code 5.

```
create table course_prerequisite_relations
(
    id              serial  not null
        constraint course_prerequisite_relations_pkey
            primary key,
    course_id       varchar not null
        constraint course_prerequisite_relations_course_id_fkey
            references course
            on delete cascade,
    prerequisite_id varchar
        constraint course_prerequisite_relations_prerequisite_id_fkey
            references course
            on delete cascade,
    and_logic       boolean,
    constraint prerequisite_or_logic -- The check constraint
        check (((prerequisite_id IS NOT NULL) AND (and_logic IS NULL)) OR
```

```
                ((prerequisite_id IS NULL) AND (and_logic IS NOT NULL)))
);
```

Code 6. DDL of creating table course_prerequisite_relations

When we judge if a student satisfies a course's prerequisite, we only need to search a course's id in this table and order the records by id. If a course is not in the table, that means this course doesn't have any prerequisite, return true; else we use a loop to go through all the records and use a stack to store the results of Boolean expression. The code of judging prerequisite is shown in Code 6.

```
create function judge_prerequisite(stu_id integer, cour_id character varying) returns boolean
    language plpgsql
as
$$
declare
    pre   record;
    stack bool[];
    top   int := 1;
begin
    if not exists(select null from course where id = cour_id) or
       not exists(select null from student where id = stu_id) then
        return null;
    end if;
    drop table if exists selected;
    create temp table if not exists selected as
    select distinct x.id, x.and_logic, coalesce(y.rst, false) as rst
    from (select * from course_prerequisite_relations cpr where cpr.course_id = cour_id) x
             left join
         (select course_id, true as rst
          from student_section_relations ssr
                   join course_section cs on cs.id = ssr.section_id
          where ssr.student_id = stu_id
            and (ssr.grade >= 60 or ssr.grade = -1)) y
         on coalesce(x.prerequisite_id, '-1') = y.course_id;

    if not exists(select null from selected) then
        return true;
    end if;

    stack[1] = true; -- In case that there are no prerequisite left, only Boolean operations
    for pre in (select id, and_logic, rst from selected order by id)
        loop
            if pre.and_logic is null then
                stack[top] := pre.rst;
            elseif pre.and_logic then
                if top <= 2 then        -- In case that there are prerequisite deleted
                    top := top - 1;
                else
                    top := top - 2;
                    stack[top] := (stack[top] and stack[top + 1]);
                end if;
            else
```

```
                if top <= 2 then        -- In case that there are
prerequisite deleted
                    top := top - 1;
                else
                    top := top - 2;
                    stack[top] := (stack[top] or stack[top + 1]);
                end if;
            end if;
            top := top + 1;
        end loop;
    drop table if exists selected;
    return stack[1];
end;
```

Code 6. Judging prerequisite

This design is reliable: it is correct if one or more prerequisites are deleted. Even when there is only records representing Boolean operators remaining, this function can still return the correct answer. All we need to make sure is the expression is inserted into the prerequisite table in a correct order (it's ok even the id is not consecutive!).

3. Speed up

(1) Binary Week List

Week list is mainly used to detect conflict courses and search course according to week.

We uses a integer to store a course's week list: transform the integer into a binary number, and each bit represent if there is course in this week.

For example, `30036 = 111010101010100`, which means this class takes place at the 3rd, 5th, 7th, 9th, 11th, 13th, 14th, 15th weeks.

```
// Transform from Set<Short> to integer
int weekListNum = 0;
for (Short aShort : weekList) {
  weekListNum += (1 << (aShort - 1));
}

// Transform from integer to Set<Short>
short weekCnt = 0;
while (weekListNum > 0) {
  weekCnt++;
  if (weekListNum % 2 == 1) {
    tempClass.weekList.add(weekCnt);
  }
  weekListNum >>= 1;
}
```

Code 7. Transform between integer and Set

The advantage of this design is, when we need to judge whether two class have conflict weeks, we only need to make an `AND` of their week list integers: If the result is 0, then they have no overlapping weeks; if the result is not 0, then change the result into a binary form and the digit that is 1 is the weeks that they have in common. Compared with the design of storing week list in

an array, this method only need one operation to detect conflict while the array may need to compare one by one.

Also, when we need to judge if a class takes place at week n, we only need to `AND` the class's week list with `(1 << (n-1))`. If the result is 0 then false, else the answer is true.

(2) Lots of Function and Procedure Used

Lower the cost of JDBC transmission. We only need one call in Java.

(3) Get the Static Table in Advance

If a function uses a static select result repeatedly, we will do the query first and create a temporary table to store it. Then we will not need to select repeatedly.

Also, uses keyword `in` instead of `if exists`: only select once.

(4) Store the Data that Need to be Used in Future in Table

Some processed data that we might use a lot of times in the future such as a user's full name or a section's full name will be saved in table in advance when the record is inserted, and we used a trigger to keep it correct. In this case, we won't need to process it every time we want to use it.

```
create function generate_instructor_full_name() returns trigger
    language plpgsql
as
$$
begin
    if new.first_name ~ '[a-zA-Z ]' and new.last_name ~ '[a-zA-Z ]'
    then
        new.full_name := new.first_name || ' ' || new.last_name;
        new.other_name := new.first_name || new.last_name;
    else
        new.full_name := new.first_name || new.last_name;
        new.other_name := new.first_name || ' ' || new.last_name;
    end if;
    return new;
end;
$$;

create trigger update_instructor_full_name
    before insert or update
    on instructor
    for each row
execute procedure generate_instructor_full_name();
```

Code 8. Generating full names of instructors

Code 8 shows the trigger of generating a instructor's full name and other name (used in search), and the result is shown in Figure 2.

id	first_name	last_name	full_name	other_name
30000105	陈	继勇	陈继勇	陈 继勇
30000365	赵	飞	赵飞	赵 飞
30000314	Andrew	Hutchins	Andrew Hutchins	AndrewHutchins
30000037	严	明	严明	严 明
30000077	王	俊坚	王俊坚	王 俊坚
30000125	吕	沫	吕沫	吕 沫
30000362	卢	阳	卢阳	卢 阳
30000192	程	然	程然	程 然
30000068	黄	业绪	黄业绪	黄 业绪

Figure 2. Full name and other name stored in table

(5) Add Index

Index is a basic way to improve searching speed, but it also cause the decrease of inserting speed. After adding 2 index to the same table our benefit will drop rapidly. So control the number of index is also important.

```
1 create index idx_course_section_class_section_id on
  course_section_class(section_id);
2 create index idx_course_section_course_id_semester_id on
  course_section(course_id, semester_id);
3 create index idx_course_section_class_instructor_id on course_section_class
  (instructor_id);
```

Code 9. Added Index

Here we choose 3 most common used search condition and make them indexed.

But to our surprise, our index didn't do much work, we thought it is because of the total searching time is too small, and there exists other much stronger factors that drag our searching speed down.

(6) Lower the Connection Cost

As we all know, seeking to connect to the database will takes up a lot of time. So we minimize the count of the communications with database . For example, we change the code of the implementation `SearchCourse` to decrease one communication which is used seek for information about the conflicted classes.

```
1  @Override
2    public List<CourseSearchEntry> searchCourse(int studentId, int
   semesterId,
3        @Nullable String searchCid, @Nullable String searchName, @Nullable
   String searchInstructor,
4        @Nullable DayOfWeek searchDayOfWeek, @Nullable Short searchClassTime,
5        @Nullable List<String> searchClassLocations, CourseType
   searchCourseType, boolean ignoreFull,
6        boolean ignoreConflict, boolean ignorePassed, boolean
   ignoreMissingPrerequisites,
7        int pageSize, int pageIndex) {
8      List<CourseSearchEntry> courseSearchResult = new ArrayList<>();
9      CourseSearchEntry tempEntry;
10     CourseSectionClass tempClass;
11     short weekCnt;
```

```
    int weekListNum, sType;
    Connection conn = null;
    PreparedStatement pStmt = null;
    ResultSet rst = null;
    boolean stop;
    try {
      conn = sds.getSQLConnection();
      pStmt = conn.prepareStatement(
          "select * from search_course(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?) as (course_id varchar, course_name varchar, credit int, class_hour int, is_pf_grading bool, section_id int, section_name varchar, total_capacity int, left_capacity int,  class_id int, ins_id int, ins_name varchar, day_of_week int, week_list int, class_begin int, class_end int, location varchar, e_full_name varchar, type_int int, a_full_name varchar)");
      pStmt.setInt(1, studentId);
      pStmt.setInt(2, semesterId);
      if (searchCid == null) {
        pStmt.setNull(3, Types.VARCHAR);
      } else {
        pStmt.setString(3, searchCid);
      }
      if (searchName == null) {
        pStmt.setNull(4, Types.VARCHAR);
      } else {
        pStmt.setString(4, searchName);
      }
      if (searchInstructor == null) {
        pStmt.setNull(5, Types.VARCHAR);
      } else {
        pStmt.setString(5, searchInstructor);
      }
      if (searchDayOfWeek == null) {
        pStmt.setNull(6, Types.INTEGER);
      } else {
        pStmt.setInt(6, searchDayOfWeek.getValue());
      }
      if (searchClassTime == null) {
        pStmt.setNull(7, Types.INTEGER);
      } else {
        pStmt.setInt(7, searchClassTime);
      }
      if (searchClassLocations == null) {
        pStmt.setNull(8, Types.ARRAY);
      } else {
        pStmt.setArray(8,
            conn.createArrayOf("varchar", searchClassLocations.toArray(new String[0])));
      }
      switch (searchCourseType) {
        case ALL:
          sType = 1;
          break;
        case MAJOR_COMPULSORY:
          sType = 2;
          break;
        case MAJOR_ELECTIVE:
          sType = 3;
```

```
          break;
        case CROSS_MAJOR:
          sType = 4;
          break;
        case PUBLIC:
          sType = 5;
          break;
        default:
          sType = 0;
          break;
      }
      pStmt.setInt(9, sType);
      pStmt.setBoolean(10, ignoreFull);
      pStmt.setBoolean(11, ignoreConflict);
      pStmt.setBoolean(12, ignorePassed);
      pStmt.setBoolean(13, ignoreMissingPrerequisites);
      pStmt.setInt(14, pageSize);
      pStmt.setInt(15, pageIndex);
      rst = pStmt.executeQuery();
      if (rst.next()) {
        stop = false;
        while (!stop) {
          tempEntry = new CourseSearchEntry();
          tempEntry.course = new Course();
          tempEntry.sectionClasses = new HashSet<>();
          tempEntry.section = new CourseSection();
          tempEntry.conflictCourseNames = new ArrayList<>();
          tempEntry.course.id = rst.getString(1);
          tempEntry.course.name = rst.getString(2);
          tempEntry.course.credit = rst.getInt(3);
          tempEntry.course.classHour = rst.getInt(4);
          tempEntry.course.grading =
              rst.getBoolean(5) ? CourseGrading.PASS_OR_FAIL :
CourseGrading.HUNDRED_MARK_SCORE;
          tempEntry.section.id = rst.getInt(6);
          tempEntry.section.name = rst.getString(7);
          tempEntry.section.totalCapacity = rst.getInt(8);
          tempEntry.section.leftCapacity = rst.getInt(9);

          while (rst.getInt(6) == tempEntry.section.id) {
            if(rst.getInt(19) == 0) {
              // get classes
              tempClass = new CourseSectionClass();
              tempClass.instructor = new Instructor();
              tempClass.weekList = new HashSet<>();
              tempClass.id = rst.getInt(10);
              tempClass.instructor.id = rst.getInt(11);
              tempClass.instructor.fullName = rst.getString(12);
              tempClass.dayOfWeek = DayOfWeek.of(rst.getInt(13));
              weekListNum = rst.getInt(14);
              weekCnt = 0;
              while (weekListNum > 0) {
                weekCnt++;
                if (weekListNum % 2 == 1) {
                  tempClass.weekList.add(weekCnt);
                }
                weekListNum >>= 1;
              }
```

```
              tempClass.classBegin = rst.getShort(15);
              tempClass.classEnd = rst.getShort(16);
              tempClass.location = rst.getString(17);
              tempEntry.sectionClasses.add(tempClass);
            } else if (rst.getInt(19) == 1) {
              tempEntry.conflictCourseNames.add(rst.getString(18));
            }

            if (!rst.next()) {
              stop = true;
              break;
            }
          }
          courseSearchResult.add(tempEntry);
        }
      }
    } catch (SQLException e) {
      e.printStackTrace();
    } finally {
      try {
        if (rst != null) {
          rst.close();
        }
        if (pStmt != null) {
          pStmt.close();
        }
        if (conn != null) {
          conn.close();
        }
      } catch (SQLException e) {
        e.printStackTrace();
      }
    }
    return courseSearchResult;
  }
```

Code 9. SearchCourse after reduce one communication with database

```
create or replace function search_course(search_student_id integer,
search_semester_id integer, search_course_id character varying, search_name
character varying, search_instructor character varying, search_day_of_week
integer, search_class_time integer, search_class_location character
varying[], search_course_type integer, ignore_full boolean, ignore_conflict
boolean, ignore_passed boolean, ignore_missing_prerequisites boolean,
page_size integer, page_index integer) returns SETOF record
    language plpgsql
as
$$
declare
    empty_location bool := (array_length(search_class_location, 1) = 0);
begin
    drop table if exists conflict_sections;
    create temp table if not exists conflict_sections on commit drop as
        (select *
         from get_all_conflict_sections(search_student_id,
                                        search_semester_id) as (section_id
int));
```

```
13
14      drop table if exists all_conflicts_table;
15      create temp table if not exists all_conflicts_table on commit drop as
16          (select *
17           from get_enrolled_conflict_sections(search_student_id,
18                                               search_semester_id) as
(e_full_name varchar, e_sec_id integer));
19
20      return query
21          with available_sections as (
22              select distinct c.id    as course_id,
23                              c.name  as course_name,
24                              c.credit,
25                              c.class_hour,
26                              c.is_pf_grading,
27                              cs.id   as section_id,
28                              cs.semester_id,
29                              cs.name as section_name,
30                              cs.full_name,
31                              cs.total_capacity,
32                              cs.left_capacity
33              from course c
34                       join course_section cs
35                            on c.id = cs.course_id
36                       join course_section_class csc on cs.id =
csc.section_id
37                       join semester s on s.id = cs.semester_id
38                       join instructor i on csc.instructor_id = i.id
39              where s.id = search_semester_id
40                and case search_course_id is null
41                         when true then true
42                         when false then
43                             position(search_course_id in c.id) > 0
44                    end
45                and case search_name is null
46                         when true then true
47                         when false then
48                             position(search_name in cs.full_name) > 0
49                    end
50                and case search_instructor is null
51                         when true then true
52                         when false then (position(search_instructor in
i.other_name) = 1
53                             or position(search_instructor in i.last_name) = 1
54                             or position(search_instructor in i.full_name) =
1)
55                    end
56                and case search_day_of_week is null
57                         when true then true
58                         when false then csc.day_of_week = search_day_of_week
59                    end
60                and case search_class_time is null
61                         when true then true
62                         when false then search_class_time between
csc.class_begin
63                             and csc.class_end
64                    end
65                and case (search_class_location is null or empty_location)
```

```
                    when true then true
                    when false then match_location(csc.location,
search_class_location)
                end
              and case search_course_type
                -- ALL
                    when 1 then true
                -- MAJOR_COMPULSORY
                    when 2 then c.id in (select course_id
                                         from major_course_relations mcr
                                                  join student s2 on
mcr.major_id = s2.major_id
                                         where s2.id = search_student_id
                                           and mcr.is_compulsory)
                -- MAJOR_ELECTIVE
                    when 3 then c.id in (select course_id
                                         from major_course_relations mcr
                                                  join student s2 on
mcr.major_id = s2.major_id
                                         where s2.id = search_student_id
                                           and not mcr.is_compulsory)
                -- CROSS_MAJOR
                    when 4 then c.id in (select distinct mcr.course_id
                                         from major_course_relations mcr
                                         where mcr.course_id not in
(select course_id
                                                                 from
major_course_relations mcr2

     join student s3 on mcr2.major_id = s3.major_id

where s3.id = search_student_id))
                -- PUBLIC
                    when 5 then c.id not in
                                (select distinct course_id from
major_course_relations)
                end
              and case ignore_full
                    when false then true
                    when true then cs.left_capacity > 0
                end
              and case ignore_conflict
                    when false then true
                    when true
                        then cs.id not in (select section_id from
conflict_sections)
                end
              and case ignore_passed
                    when false then true
                    when true then c.id not in (select distinct
cs1.course_id
                                                from
student_section_relations ssr
                                                         join
course_section cs1 on cs1.id = ssr.section_id
                                                where (ssr.grade = -1 or
ssr.grade >= 60)
```

```
                                                and ssr.student_id =
search_student_id)
                end
              and case ignore_missing_prerequisites
                       when false then true
                       when true then judge_prerequisite(search_student_id,
c.id)
                end
            order by course_id, cs.full_name
            limit page_size offset page_index * page_size)
        select available_sections.course_id,
               available_sections.course_name,
               available_sections.credit,
               available_sections.class_hour,
               available_sections.is_pf_grading,
               available_sections.section_id,
               available_sections.section_name,
               available_sections.total_capacity,
               available_sections.left_capacity,
               csc2.id,
               i2.id,
               i2.full_name                 as ins_full_name,
               csc2.day_of_week,
               csc2.week_list,
               csc2.class_begin,
               csc2.class_end,
               csc2.location,
               null                         as conflict_full_name,
               0                            as order_case,
               available_sections.full_name as a_full_name
        from available_sections
                 join course_section_class csc2 on
available_sections.section_id = csc2.section_id
                 join instructor i2 on i2.id = csc2.instructor_id
        union all
        select available_sections.course_id,
               available_sections.course_name,
               available_sections.credit,
               available_sections.class_hour,
               available_sections.is_pf_grading,
               available_sections.section_id,
               available_sections.section_name,
               available_sections.total_capacity,
               available_sections.left_capacity,
               all_conflicts_table.e_sec_id,
               null,
               null,
               null,
               null,
               null,
               null,
               null,
               all_conflicts_table.e_full_name as conflict_full_name,
               1                               as order_case,
               available_sections.full_name    as a_full_name
        from available_sections
                 join all_conflicts_table
```

```
                    on available_sections.section_id =
all_conflicts_table.e_sec_id
        order by course_id, a_full_name,
                 order_case, conflict_full_name;
end;
$$;
```

Code 10. The corresponding changes in SQL

After doing this change we got great improvement, 8 seconds less than before.

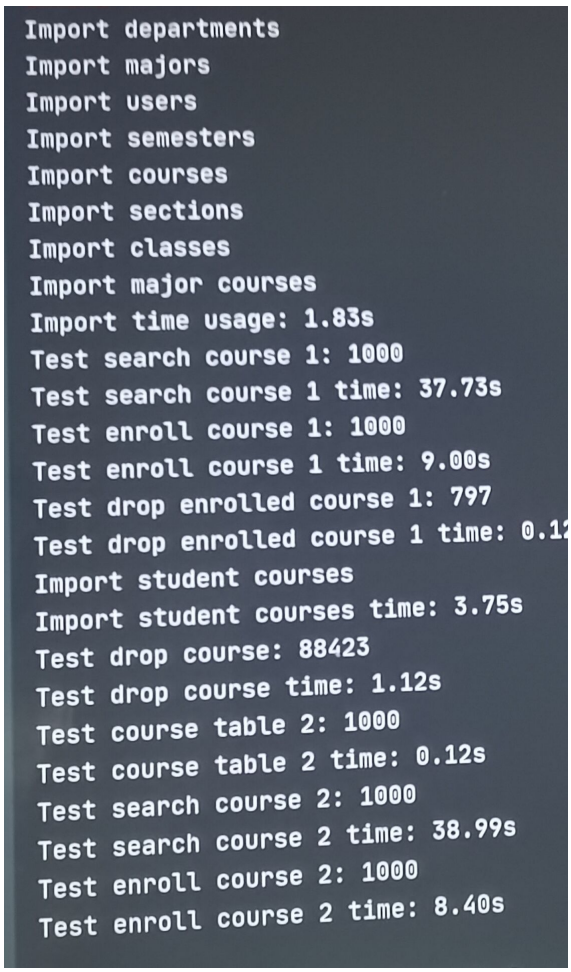

Figure 3. Before changes (after adding index)

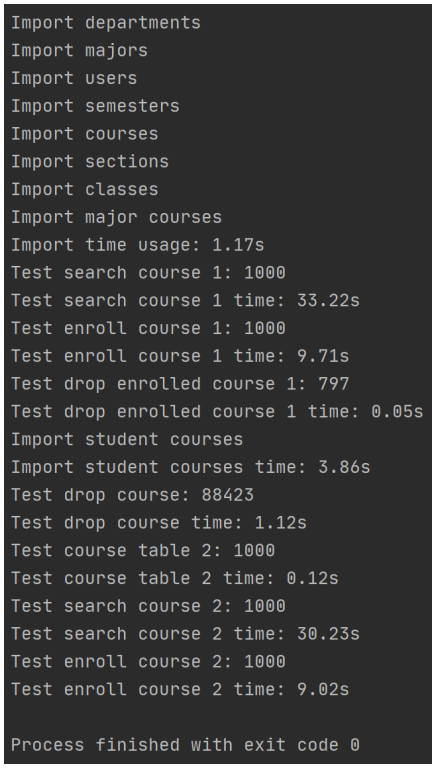

Figure 4.After changes (still with index)