

# CS307 Presentation Material (Final)

## 1. Implementation of searchCourse

### (1) Function in Database

```
1  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
2      language plpgsql
3  as
4  $$
5  declare
6      empty_location bool := (array_length(search_class_location, 1) = 0);
7  begin
8      return query
9          with conflict_sections as (
10             select *
11             from get_all_conflict_sections(search_student_id,
12                                             search_semester_id) as
13             (section_id int)
14             )
15             select available_sections.course_id,
16                    available_sections.course_name,
17                    available_sections.credit,
18                    available_sections.class_hour,
19                    available_sections.is_pf_grading,
20                    available_sections.section_id,
21                    available_sections.section_name,
22                    available_sections.total_capacity,
23                    available_sections.left_capacity,
24                    csc2.id,
25                    i2.id,
26                    i2.full_name as ins_full_name,
27                    csc2.day_of_week,
28                    csc2.week_list,
29                    csc2.class_begin,
30                    csc2.class_end,
31                    csc2.location
32             from (select distinct c.id    as course_id,
33                                c.name   as course_name,
34                                c.credit,
35                                c.class_hour,
36                                c.is_pf_grading,
37                                cs.id    as section_id,
38                                cs.semester_id,
39                                cs.name  as section_name,
40                                cs.full_name,
41                                cs.total_capacity,
```

```

42         from course c
43             join course_section cs
44                 on c.id = cs.course_id
45             join course_section_class csc
46                 on cs.id = csc.section_id
47             join semester s
48                 on s.id = cs.semester_id
49             join instructor i
50                 on csc.instructor_id = i.id
51 where s.id = search_semester_id
52 and case search_course_id is null
53     when true then true
54     when false then
55         position(search_course_id in c.id) > 0
56     end
57 and case search_name is null
58     when true then true
59     when false then
60         position(search_name in cs.full_name) > 0
61     end
62 and case search_instructor is null
63     when true then true
64     when false then (
65         position(search_instructor in i.other_name) = 1
66         or position(search_instructor in i.last_name) = 1
67         or position(search_instructor in i.full_name) = 1)
68     end
69 and case search_day_of_week is null
70     when true then true
71     when false then
72         csc.day_of_week = search_day_of_week
73     end
74 and case search_class_time is null
75     when true then true
76     when false then search_class_time between
77         csc.class_begin and csc.class_end
78     end
79 and case (search_class_location is null or empty_location)
80     when true then true
81     when false then
82         match_location(csc.location,
search_class_location)
83     end
84 and case search_course_type
85     -- ALL
86     when 1 then true
87     -- MAJOR_COMPULSORY
88     when 2 then c.id in (
89         select course_id
90         from major_course_relations mcr
91             join student s2
92                 on mcr.major_id = s2.major_id
93         where s2.id = search_student_id
94             and mcr.is_compulsory
95     )
96     -- MAJOR_ELECTIVE
97     when 3 then c.id in (
98         select course_id

```

```

99         from major_course_relations mcr
100         join student s2
101         on mcr.major_id = s2.major_id
102         where s2.id = search_student_id
103         and not mcr.is_compulsory
104     )
105     -- CROSS_MAJOR
106     when 4 then c.id in (
107         select distinct mcr.course_id
108         from major_course_relations mcr
109         where mcr.course_id not in (
110             select course_id
111             from major_course_relations mcr2
112             join student s3
113             on mcr2.major_id =
s3.major_id
114             where s3.id = search_student_id
115         )
116     )
117     -- PUBLIC
118     when 5 then c.id not in (
119         select distinct course_id
120         from major_course_relations
121     )
122     end
123     and case ignore_full
124         when false then true
125         when true then cs.left_capacity > 0
126     end
127     and case ignore_conflict
128         when false then true
129         when true
130             then cs.id not in (
131                 select section_id
132                 from conflict_sections
133             )
134         end
135     and case ignore_passed
136         when false then true
137         when true then c.id not in (
138             select distinct cs1.course_id
139             from student_section_relations ssr
140             join course_section cs1
141             on cs1.id = ssr.section_id
142             where (ssr.grade = -1 or ssr.grade >= 60)
143             and ssr.student_id = search_student_id
144         )
145     end
146     and case ignore_missing_prerequisites
147         when false then true
148         when true then
149             judge_prerequisite(search_student_id, c.id)
150     end
151     order by course_id, cs.full_name
152     limit page_size offset page_index * page_size)
available_sections
153     join course_section_class csc2
154     on available_sections.section_id = csc2.section_id

```

```

155         join instructor i2
156             on i2.id = csc2.instructor_id
157         order by course_id, available_sections.full_name;
158     end;
159 $$;

```

Code 1. Function of course searching

## (2) Method in Java

```

1 conn = sds.getConnection();
2 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

```

1 create function enroll_course(stu_id integer, sec_id integer) returns
integer
2     language plpgsql
3 as
4 $$
5 declare
6     temp_course varchar;
7     temp_bool    bool;
8     left_cap     int;
9 begin
10    -- NO STUDENT (UNKNOWN ERROR)
11    if not exists(select null from student where id = stu_id) then
12        return 0;
13    end if;
14
15    if not exists(select null from course_section where id = sec_id) then
16        -- COURSE_NOT_FOUND
17        return 1;
18    end if;
19
20    if exists(select grade
21              from student_section_relations
22              where student_id = stu_id
23                  and section_id = sec_id) then
24        -- ALREADY_ENROLLED
25        return 2;
26    end if;
27
28    select course_id, left_capacity
29    into temp_course, left_cap

```

```

30     from course_section
31     where id = sec_id for update;
32
33     if exists(select null
34               from student_section_relations ssr
35               join course_section cs on cs.id = ssr.section_id
36               where cs.course_id = temp_course
37                     and (ssr.grade = -1 or ssr.grade >= 60)
38                     and ssr.student_id = stu_id) then
39         -- ALREADY_PASSED
40         return 3;
41     end if;
42
43     select judge_prerequisite(stu_id, temp_course) into temp_bool;
44
45     if not temp_bool then
46         -- PREREQUISITE_NOT_FULFILLED
47         return 4;
48     end if;
49
50     select detect_conflict(stu_id, sec_id) into temp_bool;
51     if temp_bool then
52         -- COURSE_CONFLICT_FOUND
53         return 5;
54     end if;
55
56
57     if left_cap <= 0 then
58         -- COURSE_IS_FULL
59         return 6;
60     end if;
61
62     update course_section set left_capacity = (left_capacity - 1) where id =
sec_id;
63     insert into student_section_relations (student_id, section_id) values
(stu_id, sec_id);
64     -- SUCCESS
65     return 7;
66
67 end;
68 $$;

```

### 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.

```

1  pCase = new Cases<>() {
2      StringBuilder temp;
3
4      @Override
5      public String match(AndPrerequisite self) {

```

```

6      String[] children = self.terms.stream()
7          .map(term -> term.when(this))
8          .toArray(String[]::new);
9      temp = new StringBuilder(children[0]);
10     for (int i = 1; i < children.length; i++) {
11         temp.append("|").append(children[i]).append("|AND");
12     }
13     return temp.toString();
14 }
15
16 @Override
17 public String match(OrPrerequisite self) {
18     String[] children = self.terms.stream()
19         .map(term -> term.when(this))
20         .toArray(String[]::new);
21     temp = new StringBuilder(children[0]);
22     for (int i = 1; i < children.length; i++) {
23         temp.append("|").append(children[i]).append("|OR");
24     }
25     return temp.toString();
26 }
27
28 @Override
29 public String match(CoursePrerequisite self) {
30     return self.courseID;
31 }
32 };

```

#### 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.

```

1  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)
2      language plpgsql
3  as
4  $$
5  declare
6      temp_array varchar[];
7      i          int;
8  begin
9      insert into course (id, name, credit, class_hour, is_pf_grading)
10     values (cour_id, cour_name, cour_credit, cour_ch, cour_pf);
11
12     if cour_pre is not null then
13         select regexp_split_to_array(cour_pre, E'\\|') into temp_array;
14         for i in 1 .. array_length(temp_array, 1)
15             loop
16                 if temp_array[i] = 'AND' then
17                     insert into course_prerequisite_relations(course_id,
18 prerequisite_id, and_logic)
19                     values (cour_id, null, true);
20                 elseif temp_array[i] = 'OR' then
21                     insert into course_prerequisite_relations(course_id,
22 prerequisite_id, and_logic)

```

```

21         VALUES (cour_id, null, false);
22     else
23         insert into course_prerequisite_relations(course_id,
prerequisite_id, and_logic)
24         VALUES (cour_id, temp_array[i], null);
25     end if;
26 end loop;
27 end if;
28 end;
29 $$;

```

#### 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 use 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.

```

1  create table course_prerequisite_relations
2  (
3      id          serial not null
4      constraint course_prerequisite_relations_pkey
5          primary key,
6      course_id   varchar not null
7      constraint course_prerequisite_relations_course_id_fkey
8          references course
9          on delete cascade,
10     prerequisite_id varchar
11     constraint course_prerequisite_relations_prerequisite_id_fkey
12         references course
13         on delete cascade,
14     and_logic    boolean,
15     constraint prerequisite_or_logic -- The check constraint
16     check (((prerequisite_id IS NOT NULL) AND (and_logic IS NULL)) OR

```

```

17      ((prerequisite_id IS NULL) AND (and_logic IS NOT NULL)))
18  );

```

#### 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.

```

1  create function judge_prerequisite(stu_id integer, cour_id character
2  varying) returns boolean
3  language plpgsql
4  as
5  $$
6  declare
7      pre    record;
8      stack  bool[];
9      top    int := 1;
10 begin
11     if not exists(select null from course where id = cour_id) or
12     not exists(select null from student where id = stu_id) then
13         return null;
14     end if;
15     drop table if exists selected;
16     create temp table if not exists selected as
17     select distinct x.id, x.and_logic, coalesce(y.rst, false) as rst
18     from (select * from course_prerequisite_relations cpr where
19     cpr.course_id = cour_id) x
20     left join
21     (select course_id, true as rst
22     from student_section_relations ssr
23     join course_section cs on cs.id = ssr.section_id
24     where ssr.student_id = stu_id
25     and (ssr.grade >= 60 or ssr.grade = -1)) y
26     on coalesce(x.prerequisite_id, '-1') = y.course_id;
27
28     if not exists(select null from selected) then
29         return true;
30     end if;
31
32     stack[1] = true; -- In case that there are no prerequisite left, only
33     Boolean operations
34     for pre in (select id, and_logic, rst from selected order by id)
35     loop
36         if pre.and_logic is null then
37             stack[top] := pre.rst;
38         elseif pre.and_logic then
39             if top <= 2 then          -- In case that there are
40             prerequisite deleted
41                 top := top - 1;
42             else
43                 top := top - 2;
44                 stack[top] := (stack[top] and stack[top + 1]);
45             end if;
46         else

```



```

43         if top <= 2 then           -- In case that there are
prerequisite deleted
44             top := top - 1;
45         else
46             top := top - 2;
47             stack[top] := (stack[top] or stack[top + 1]);
48         end if;
49     end if;
50     top := top + 1;
51 end loop;
52 drop table if exists selected;
53 return stack[1];
54 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 use an integer to store a course's week list: transform the integer into a binary number, and each bit represents if there is a course in this week.

For example, `30036 = 1110101010100`, which means this class takes place at the 3rd, 5th, 7th, 9th, 11th, 13th, 14th, 15th weeks.

```

1  // Transform from Set<Short> to integer
2  int weekListNum = 0;
3  for (Short aShort : weekList) {
4      weekListNum += (1 << (aShort - 1));
5  }
6
7  // Transform from integer to Set<Short>
8  short weekCnt = 0;
9  while (weekListNum > 0) {
10     weekCnt++;
11     if (weekListNum % 2 == 1) {
12         tempClass.weekList.add(weekCnt);
13     }
14     weekListNum >>= 1;
15 }

```

**Code 7. Transform between integer and Set**

The advantage of this design is, when we need to judge whether two classes 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.

```
1  create function generate_instructor_full_name() returns trigger
2      language plpgsql
3  as
4  $$
5  begin
6      if new.first_name ~ '[a-zA-Z ]' and new.last_name ~ '[a-zA-Z ]'
7      then
8          new.full_name := new.first_name || ' ' || new.last_name;
9          new.other_name := new.first_name || new.last_name;
10     else
11         new.full_name := new.first_name || new.last_name;
12         new.other_name := new.first_name || ' ' || new.last_name;
13     end if;
14     return new;
15 end;
16 $$;
17
18 create trigger update_instructor_full_name
19     before insert or update
20     on instructor
21     for each row
22     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 searchClasTime,
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;

```

```

12     int weekListNum, sType;
13     Connection conn = null;
14     PreparedStatement pstmt = null;
15     ResultSet rst = null;
16     boolean stop;
17     try {
18         conn = sds.getSQLConnection();
19         pstmt = conn.prepareStatement(
20             "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)");
21         pstmt.setInt(1, studentId);
22         pstmt.setInt(2, semesterId);
23         if (searchCid == null) {
24             pstmt.setNull(3, Types.VARCHAR);
25         } else {
26             pstmt.setString(3, searchCid);
27         }
28         if (searchName == null) {
29             pstmt.setNull(4, Types.VARCHAR);
30         } else {
31             pstmt.setString(4, searchName);
32         }
33         if (searchInstructor == null) {
34             pstmt.setNull(5, Types.VARCHAR);
35         } else {
36             pstmt.setString(5, searchInstructor);
37         }
38         if (searchDayOfWeek == null) {
39             pstmt.setNull(6, Types.INTEGER);
40         } else {
41             pstmt.setInt(6, searchDayOfWeek.getValue());
42         }
43         if (searchClassTime == null) {
44             pstmt.setNull(7, Types.INTEGER);
45         } else {
46             pstmt.setInt(7, searchClassTime);
47         }
48         if (searchClassLocations == null) {
49             pstmt.setNull(8, Types.ARRAY);
50         } else {
51             pstmt.setArray(8,
52                 conn.createArrayOf("varchar", searchClassLocations.toArray(new
String[0])));
53         }
54         switch (searchCourseType) {
55             case ALL:
56                 sType = 1;
57                 break;
58             case MAJOR_COMPULSORY:
59                 sType = 2;
60                 break;
61             case MAJOR_ELECTIVE:
62                 sType = 3;

```

```

63         break;
64     case CROSS_MAJOR:
65         sType = 4;
66         break;
67     case PUBLIC:
68         sType = 5;
69         break;
70     default:
71         sType = 0;
72         break;
73 }
74 pstmt.setInt(9, sType);
75 pstmt.setBoolean(10, ignoreFull);
76 pstmt.setBoolean(11, ignoreConflict);
77 pstmt.setBoolean(12, ignorePassed);
78 pstmt.setBoolean(13, ignoreMissingPrerequisites);
79 pstmt.setInt(14, pageSize);
80 pstmt.setInt(15, pageIndex);
81 rst = pstmt.executeQuery();
82 if (rst.next()) {
83     stop = false;
84     while (!stop) {
85         tempEntry = new CourseSearchEntry();
86         tempEntry.course = new Course();
87         tempEntry.sectionClasses = new HashSet<>();
88         tempEntry.section = new CourseSection();
89         tempEntry.conflictCourseNames = new ArrayList<>();
90         tempEntry.course.id = rst.getString(1);
91         tempEntry.course.name = rst.getString(2);
92         tempEntry.course.credit = rst.getInt(3);
93         tempEntry.course.classHour = rst.getInt(4);
94         tempEntry.course.grading =
95             rst.getBoolean(5) ? CourseGrading.PASS_OR_FAIL :
CourseGrading.HUNDRED_MARK_SCORE;
96         tempEntry.section.id = rst.getInt(6);
97         tempEntry.section.name = rst.getString(7);
98         tempEntry.section.totalCapacity = rst.getInt(8);
99         tempEntry.section.leftCapacity = rst.getInt(9);
100
101         while (rst.getInt(6) == tempEntry.section.id) {
102             if(rst.getInt(19) == 0) {
103                 // get classes
104                 tempClass = new CourseSectionClass();
105                 tempClass.instructor = new Instructor();
106                 tempClass.weekList = new HashSet<>();
107                 tempClass.id = rst.getInt(10);
108                 tempClass.instructor.id = rst.getInt(11);
109                 tempClass.instructor.fullName = rst.getString(12);
110                 tempClass.dayOfWeek = DayOfWeek.of(rst.getInt(13));
111                 weekListNum = rst.getInt(14);
112                 weekCnt = 0;
113                 while (weekListNum > 0) {
114                     weekCnt++;
115                     if (weekListNum % 2 == 1) {
116                         tempClass.weekList.add(weekCnt);
117                     }
118                     weekListNum >>= 1;
119                 }

```

```

120         tempClass.classBegin = rst.getShort(15);
121         tempClass.classEnd = rst.getShort(16);
122         tempClass.location = rst.getString(17);
123         tempEntry.sectionClasses.add(tempClass);
124     } else if (rst.getInt(19) == 1) {
125         tempEntry.conflictCourseNames.add(rst.getString(18));
126     }
127
128     if (!rst.next()) {
129         stop = true;
130         break;
131     }
132 }
133 courseSearchResult.add(tempEntry);
134 }
135 }
136 } catch (SQLException e) {
137     e.printStackTrace();
138 } finally {
139     try {
140         if (rst != null) {
141             rst.close();
142         }
143         if (pstmt != null) {
144             pstmt.close();
145         }
146         if (conn != null) {
147             conn.close();
148         }
149     } catch (SQLException e) {
150         e.printStackTrace();
151     }
152 }
153 return courseSearchResult;
154 }

```

**Code 9. SearchCourse after reduce one communication with database**

```

1  create or replace function search_course(search_student_id integer,
2  search_semester_id integer, search_course_id character varying, search_name
3  character varying, search_instructor character varying, search_day_of_week
4  integer, search_class_time integer, search_class_location character
5  varying[], search_course_type integer, ignore_full boolean, ignore_conflict
6  boolean, ignore_passed boolean, ignore_missing_prerequisites boolean,
7  page_size integer, page_index integer) returns SETOF record
8  language plpgsql
9  as
10 $$
11 declare
12     empty_location bool := (array_length(search_class_location, 1) = 0);
13 begin
14     drop table if exists conflict_sections;
15     create temp table if not exists conflict_sections on commit drop as
16         (select *
17          from get_all_conflict_sections(search_student_id,
18                                         search_semester_id) as (section_id
19 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
19         (e_full_name varchar, e_sec_id integer));
20
21     return query
22         with available_sections as (
23             select distinct c.id      as course_id,
24                             c.name   as course_name,
25                             c.credit,
26                             c.class_hour,
27                             c.is_pf_grading,
28                             cs.id    as section_id,
29                             cs.semester_id,
30                             cs.name  as section_name,
31                             cs.full_name,
32                             cs.total_capacity,
33                             cs.left_capacity
34             from course c
35                 join course_section cs
36                 on c.id = cs.course_id
37                 join course_section_class csc on cs.id =
38                 csc.section_id
39                 join semester s on s.id = cs.semester_id
40                 join instructor i on csc.instructor_id = i.id
41             where s.id = search_semester_id
42                   and case search_course_id is null
43                       when true then true
44                       when false then
45                           position(search_course_id in c.id) > 0
46                       end
47                   and case search_name is null
48                       when true then true
49                       when false then
50                           position(search_name in cs.full_name) > 0
51                       end
52                   and case search_instructor is null
53                       when true then true
54                       when false then (position(search_instructor in
55                                           i.other_name) = 1
56                                           or position(search_instructor in i.last_name) = 1
57                                           or position(search_instructor in i.full_name) =
58                                           1)
59                       end
60                   and case search_day_of_week is null
61                       when true then true
62                       when false then csc.day_of_week = search_day_of_week
63                       end
64                   and case search_class_time is null
65                       when true then true
66                       when false then search_class_time between
67                           csc.class_begin
68                               and csc.class_end
69                       end
70                   and case (search_class_location is null or empty_location)

```

```

66         when true then true
67         when false then match_location(csc.location,
search_class_location)
68     end
69     and case search_course_type
70         -- ALL
71         when 1 then true
72         -- MAJOR_COMPULSORY
73         when 2 then c.id in (select course_id
74                               from major_course_relations mcr
75                               join student s2 on
mcr.major_id = s2.major_id
76                               where s2.id = search_student_id
77                               and mcr.is_compulsory)
78         -- MAJOR_ELECTIVE
79         when 3 then c.id in (select course_id
80                               from major_course_relations mcr
81                               join student s2 on
mcr.major_id = s2.major_id
82                               where s2.id = search_student_id
83                               and not mcr.is_compulsory)
84         -- CROSS_MAJOR
85         when 4 then c.id in (select distinct mcr.course_id
86                               from major_course_relations mcr
87                               where mcr.course_id not in
(select course_id
88                                     from
major_course_relations mcr2
89                                     join student s3 on mcr2.major_id = s3.major_id
90                                     where s3.id = search_student_id))
91         -- PUBLIC
92         when 5 then c.id not in
93             (select distinct course_id from
major_course_relations)
94     end
95     and case ignore_full
96         when false then true
97         when true then cs.left_capacity > 0
98     end
99     and case ignore_conflict
100         when false then true
101         when true
102             then cs.id not in (select section_id from
conflict_sections)
103     end
104     and case ignore_passed
105         when false then true
106         when true then c.id not in (select distinct
cs1.course_id
107                                     from
student_section_relations ssr
108                                     join
course_section cs1 on cs1.id = ssr.section_id
109                                     where (ssr.grade = -1 or
ssr.grade >= 60)

```



```

110                                     and ssr.student_id =
search_student_id)
111         end
112         and case ignore_missing_prerequisites
113             when false then true
114             when true then judge_prerequisite(search_student_id,
c.id)
115         end
116         order by course_id, cs.full_name
117         limit page_size offset page_index * page_size)
118     select available_sections.course_id,
119           available_sections.course_name,
120           available_sections.credit,
121           available_sections.class_hour,
122           available_sections.is_pf_grading,
123           available_sections.section_id,
124           available_sections.section_name,
125           available_sections.total_capacity,
126           available_sections.left_capacity,
127           csc2.id,
128           i2.id,
129           i2.full_name           as ins_full_name,
130           csc2.day_of_week,
131           csc2.week_list,
132           csc2.class_begin,
133           csc2.class_end,
134           csc2.location,
135           null                   as conflict_full_name,
136           0                      as order_case,
137           available_sections.full_name as a_full_name
138     from available_sections
139           join course_section_class csc2 on
available_sections.section_id = csc2.section_id
140           join instructor i2 on i2.id = csc2.instructor_id
141     union all
142     select available_sections.course_id,
143           available_sections.course_name,
144           available_sections.credit,
145           available_sections.class_hour,
146           available_sections.is_pf_grading,
147           available_sections.section_id,
148           available_sections.section_name,
149           available_sections.total_capacity,
150           available_sections.left_capacity,
151           all_conflicts_table.e_sec_id,
152           null,
153           null,
154           null,
155           null,
156           null,
157           null,
158           null,
159           all_conflicts_table.e_full_name as conflict_full_name,
160           1                              as order_case,
161           available_sections.full_name   as a_full_name
162     from available_sections
163           join all_conflicts_table

```

```

164         on available_sections.section_id =
all_conflicts_table.e_sec_id
165         order by course_id, a_full_name,
166                 order_case, conflict_full_name;
167 end;
168 $$;

```

#### Code 10. The corresponding changes in SQL

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

```

Import departments
Import majors
Import users
Import semesters
Import courses
Import sections
Import classes
Import major courses
Import time usage: 1.83s
Test search course 1: 1000
Test search course 1 time: 37.73s
Test enroll course 1: 1000
Test enroll course 1 time: 9.00s
Test drop enrolled course 1: 797
Test drop enrolled course 1 time: 0.12s
Import student courses
Import student courses time: 3.75s
Test drop course: 88423
Test drop course time: 1.12s
Test course table 2: 1000
Test course table 2 time: 0.12s
Test search course 2: 1000
Test search course 2 time: 38.99s
Test enroll course 2: 1000
Test enroll course 2 time: 8.40s

```

Figure 3. Before changes (after adding index)

```
Import departments
Import majors
Import users
Import semesters
Import courses
Import sections
Import classes
Import major courses
Import time usage: 1.17s
Test search course 1: 1000
Test search course 1 time: 33.22s
Test enroll course 1: 1000
Test enroll course 1 time: 9.71s
Test drop enrolled course 1: 797
Test drop enrolled course 1 time: 0.05s
Import student courses
Import student courses time: 3.86s
Test drop course: 88423
Test drop course time: 1.12s
Test course table 2: 1000
Test course table 2 time: 0.12s
Test search course 2: 1000
Test search course 2 time: 30.23s
Test enroll course 2: 1000
Test enroll course 2 time: 9.02s

Process finished with exit code 0
```

**Figure 4.**After changes (still with index)