

CS411_Team072_Stage3 Submission

Our database implements eight main tables: Professors, Departments, Users, GeneralCourse, SectionAttributes, GPAHistory, Teaches, and UserFeedback. Five of these tables contain more than 1000 rows: Professors, GeneralCourse, SectionAttributes, GPAHistory, and Teaches. The DDL commands for making and linking these tables are included in the repository in the src/db_setup.ddl file. All of the tables, excluding Users and UserFeedback, use real data. One thing we would like to note is that Departments was not included in our previous documents or submissions, but we added this table because our data didn't have a standard format for the department. We standardized the format by using DepartmentCode for everything and have this table to get the department name from the code easily.

Data sources

Professors: Come from a [page with most of the faculty](#)

Departments: Comes from GeneralCourse

Users: Generated by us

GeneralCourse: Scraped all the courses starting from [here](#) for Fall 2023. [Example page](#) we scrape GeneralCourse info from

SectionAttributes: Scraped from the "sections" portion of a GeneralCourse .xml page. [Example page](#), we scrape the SectionAttributes info from

To clarify further on *GeneralCourse* and *SectionAttributes*, we basically follow the chain of .xml files that, starting from Semester, give you all departments, courses, sections, etc. See the create_course_csv.py file in the data_parsing folder in the github to see how we did this.

GPAHistory: Comes from some of the raw data in [Wade's dataset](#)

Teaches: Generated with the Professors and SectionAttributes .csv files

UserFeedback: Generated by us

See the screenshots below for statistics on our tables and some example entries (hot off the GCP console).

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to cs411-team072.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
gcloud sql connect db-sp24-team072 --user=root --quietapote20@cloudshell:~ (cs411-team072)$ gcloud sql connect db-sp24-team072 --user=root --quiet
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 393472
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show tables
-> ;
ERROR 1046 (3D000): No database selected
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| uiuc_course_hub |
| uiuc_faculty |
+-----+
```

Figure 1: GCP Connection and databases. Our database is uiuc_course_hub

DDL Statements for our Tables (Figure 2-5)

```
# DDL for team072 database for CS 411 SP24

CREATE TABLE IF NOT EXISTS Professors(
    NetId                VARCHAR(255) PRIMARY KEY,
    LastName              VARCHAR(255),
    FirstName            VARCHAR(255),
    LastNameFirstIni     VARCHAR(255),
    DepartmentCode       VARCHAR(10)
);

CREATE TABLE IF NOT EXISTS Departments(
    DepartmentCode       VARCHAR(10) PRIMARY KEY,
    Department           VARCHAR(255)
);

CREATE TABLE IF NOT EXISTS Users(
    Email                VARCHAR(255) PRIMARY KEY,
    Password             VARCHAR(255),
    LastName             VARCHAR(255),
    FirstName            VARCHAR(255)
);
```

Figure 2 : DDL Commands

```

CREATE TABLE IF NOT EXISTS GeneralCourse(
    CourseNum          INT,
    CourseName          VARCHAR(255),
    CreditHours          VARCHAR(255),
    Description          VARCHAR(16384),
    GenEdAttrib          VARCHAR(255),
    DepartmentCode       VARCHAR(10),
    Department           VARCHAR(255),
    PRIMARY KEY (CourseNum, DepartmentCode)
);

CREATE TABLE IF NOT EXISTS SectionAttributes(
    CourseNum          INT,
    CRN                INT,
    Section             VARCHAR(10),
    SectionType         VARCHAR(255),
    Professors          VARCHAR(1024),
    DepartmentCode       VARCHAR(10),
    Department           VARCHAR(255),
    Year                INT,
    Semester            VARCHAR(16),
    FOREIGN KEY (CourseNum, DepartmentCode) REFERENCES GeneralCourse(CourseNum, DepartmentCode) ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (CRN, Semester, Year)
);

```

Figure 3 : DDL Commands

```

CREATE TABLE IF NOT EXISTS GPAHistory(
    DepartmentCode       VARCHAR(10),
    CourseNum            INT,
    CRN                  INT,
    CourseName            VARCHAR(255),
    Sched_Type            VARCHAR(10),
    A_plus                INT,
    A_stan                INT,
    A_minus               INT,
    B_plus                INT,
    B_stan                INT,
    B_minus               INT,
    C_plus                INT,
    C_stan                INT,
    C_minus               INT,
    D_plus                INT,
    D_stan                INT,
    D_minus               INT,
    F_stan                INT,
    Avg_Grade             DECIMAL(10,2),
    Instructor            VARCHAR(255),
    Semester              VARCHAR(16),
    Year                  INT,
    FOREIGN KEY (CourseNum, DepartmentCode) REFERENCES GeneralCourse(CourseNum, DepartmentCode) ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (CRN, Semester, Year)
);

CREATE TABLE IF NOT EXISTS Teaches(
    CRN                  INT,
    Semester              VARCHAR(16),
    Year                  INT,
    NetId                 VARCHAR(255),
    FOREIGN KEY (CRN, Semester, Year) REFERENCES SectionAttributes(CRN, Semester, Year) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (NetId) REFERENCES Professors(NetId) ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (CRN, Semester, Year, NetId)
);

```

Figure 4: DDL Commands

```

CREATE TABLE IF NOT EXISTS UserFeedback(
    FeedbackId          VARCHAR(255) PRIMARY KEY,
    Email               VARCHAR(255),
    Year                INT,
    CRN                 INT,
    Semester            VARCHAR(16),
    Testimony           VARCHAR(16384),
    Rating              DECIMAL(10,2),
    Difficulty           DECIMAL(10,2),
    TimeCommit          DECIMAL(10,2),
    Time_stamp          TIMESTAMP,
    FOREIGN KEY (Email) REFERENCES Users(Email) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (CRN, Semester, Year) REFERENCES SectionAttributes(CRN, Semester, Year) ON DELETE CASCADE ON UPDATE CASCADE
);

```

Figure 5 : DDL Commands

```

mysql> SELECT COUNT(*) FROM GPAHistory;
+-----+
| COUNT(*) |
+-----+
|    17294 |
+-----+
1 row in set (0.10 sec)

mysql> SELECT COUNT(*) FROM GeneralCourse;
+-----+
| COUNT(*) |
+-----+
|    4497 |
+-----+
1 row in set (0.46 sec)

mysql> SELECT COUNT(*) FROM Professors;
+-----+
| COUNT(*) |
+-----+
|    1643 |
+-----+
1 row in set (0.16 sec)

mysql> SELECT COUNT(*) FROM SectionAttributes;
+-----+
| COUNT(*) |
+-----+
|   12040 |
+-----+
1 row in set (0.11 sec)

mysql> SELECT COUNT(*) FROM Teaches
-> ;
+-----+
| COUNT(*) |
+-----+
|    2979 |
+-----+

```

Figure 6: Tables with more than 1000 rows

Queries

1. List of professors who teach 400 and 500 level courses with avg_grade greater than 3.5, group by department name

```
Select p.FirstName, p.LastName, p.Netid, g.CourseNum, d.Department, g.CourseName, g.Avg_grade
from Professors p join Teaches t on t.NetId=p.NetId
Join GPAHistory g on g.CRN=t.CRN
Join Departments d on d.DepartmentCode=g.DepartmentCode
Where g.CourseNum like "4%" or g.CourseNum like "5%"
Group by d.Department, p.FirstName, p.LastName, p.Netid, g.CourseNum, g.CourseName, g.Avg_grade
Having avg(g.avg_grade) >3.5
Order by g.avg_grade desc, d.department asc
limit 15;
```

Output:

mysql> Select p.FirstName, p.LastName, p.Netid, g.CourseNum, d.Department, g.CourseName, g.Avg_grade from Professors p join Teaches t on t.NetId=p.NetId Join GPAHistory g on g.CRN=t.CRN a d on d.DepartmentCode=g.DepartmentCode Where g.CourseNum like "4%" or g.CourseNum like "5%" Group by d.Department, p.FirstName, p.LastName, p.Netid, g.CourseNum, g.CourseName, g.Avg_grade) >3.5 Order by g.avg_grade desc, d.department asc limit 15;

FirstName	LastName	Netid	CourseNum	Department	CourseName	Avg_grade
Mimi Thi	Nguyen	mimin@illinois.edu	504	Accountancy	Auditing	3.98
Lena	Song	lenasong@illinois.edu	498	Bioengineering	Immunoeengineering	3.98
Grace	Giorgio	giorgio@illinois.edu	432	Communication	Gender Communication	3.98
Yuan-Xiang	Pan	yxpan@illinois.edu	422	Food Science and Human Nutrition	Intro Personalized Nutrition	3.98
Bo	Zhang	bozhang3@illinois.edu	475	Psychology	Personnel Psych	3.98
Hans Friedrich	Koehn	hkoehn@illinois.edu	506	Psychology	Statistical Methods I	3.98
Johnell L	Bentz	jbentz@illinois.edu	514	Special Education	Equity Issues in Spec Educ	3.98
Arminda	Formigao Gameiro	arminda@illinois.edu	518	Accountancy	Financial Statement Fraud	3.97
Jamie Lee Berry	Putnam	jamielb3@illinois.edu	518	Accountancy	Financial Statement Fraud	3.97
John A	Juvik	juvik@illinois.edu	455	Biochemistry	Technqs Biochem & Biotech	3.97
Moonsub	Shim	mshim@illinois.edu	550	Business Administration	Business Practicum	3.97
Tina H	Huang	thhuang@illinois.edu	500	Education Policy, Organization and Leadership	EdD Essentials Seminar	3.97
Laura Jean	Mattie	ljhahn@illinois.edu	515	Information Sciences	Information Modeling	3.97
Cheryl	Light Shriner	slight@illinois.edu	470	Special Education	Learning Environments I	3.97
Chengxiang	Zhai	chaz@illinois.edu	410	Computer Science	Text Information Systems	3.96

15 rows in set (0.03 sec)

Figure 7 : Query 1

2. Ranking of departments based on their academic performance, represented by the average GPA of courses offered, and the breadth of their academic offerings, indicated by the number of distinct courses available. Departments with higher average GPAs and more course offerings are considered to be of higher academic standing.

```
With DepartmentGPA AS (
    Select DepartmentCode, AVG(Avg_Grade) AS AverageGPA
    From GPAHistory Group By DepartmentCode
), DepartmentCourses AS (
    Select DepartmentCode, Count(DISTINCT CourseNum) AS CourseCount
    From GeneralCourse Group By DepartmentCode
), DepartmentRanking AS (
    Select
        d.DepartmentCode, d.Department,
        IFNULL(g.AverageGPA, 0) AS AvgGPA,
        IFNULL(c.CourseCount, 0) AS CourseCount,
        (IFNULL(g.AverageGPA, 0) * 0.7 + IFNULL(c.CourseCount, 0) * 0.3) AS CombinedScore
    From Departments d
    LEFT JOIN DepartmentGPA g ON d.DepartmentCode = g.DepartmentCode
    LEFT JOIN DepartmentCourses c ON d.DepartmentCode = c.DepartmentCode
)
Select DepartmentCode, Department, AvgGPA, CourseCount, CombinedScore,
RANK() OVER (ORDER BY CombinedScore DESC) AS Ranking
From DepartmentRanking
Order By CombinedScore DESC, DepartmentCode
LIMIT 15;
```

Metrics:

Average GPA: Calculated from the GPAHistory table, representing the academic quality.

Course Offerings: The count of distinct courses in the GeneralCourse table, representing the breadth of the department's offerings.

Combined Score: A weighted score that considers both the average GPA and the number of courses to rank departments.

Output:

```
-> LIMIT 15;
```

DepartmentCode	Department	AvgGPA	CourseCount	CombinedScore	Ranking
MUSC	Music Lessons and Ensembles	3.902500	142	45.3317500	1
MUS	Music	3.597831	108	34.9184817	2
IS	Information Sciences	3.761738	103	33.5332166	3
ECE	Electrical and Computer Engineering	3.240278	101	32.5681946	4
CS	Computer Science	3.512753	89	29.1589271	5
THEA	Theatre	3.543867	84	27.6807069	6
CLE	Clinical Sciences and Engineering	0.000000	89	26.7000000	7
BADM	Business Administration	3.689046	77	25.6823322	8
MATH	Mathematics	3.114456	76	24.9801192	9
PSYC	Psychology	3.251376	67	22.3759632	10
FIN	Finance	3.492004	65	21.9444028	11
CEE	Civil and Environmental Engineering	3.380821	61	20.6665747	12
ENGL	English	3.547385	60	20.4831695	13
CHEM	Chemistry	3.063567	58	19.5444969	14
FSHN	Food Science and Human Nutrition	3.644809	56	19.3513663	15

15 rows in set (0.17 sec)

Figure 8 : Query 2

3. List of 1 Credit course offered by the CS department having avg_grade > 3.9

```
Select distinct c.CourseNum, c.CourseName, c.CreditHours, g.Semester, g.Year, g. Avg_grade
from GeneralCourse c join GPAHistory g on g.CourseNum=c.CourseNum
Where c.CreditHours='1 hours.' and g.Year in (2022,2023) AND g.Semester LIKE 'Sp%'
Group by c.CourseNum, c.CourseName, c.CreditHours, g.Semester, g.Year, g. Avg_grade
Having g.Avg_grade >=3.9
Order by g.CourseNum desc Limit 15;
```

Output:

```
-> c.CourseNum DESC
-> LIMIT 15;
```

CourseNum	CourseName	CreditHours	Semester	Year	Avg_grade
598	Seminar	1 hours.	Spring	2022	3.90
598	Seminar	1 hours.	Spring	2022	3.91
598	Seminar	1 hours.	Spring	2022	3.98
598	Seminar	1 hours.	Spring	2023	3.91
598	Seminar	1 hours.	Spring	2023	3.94
592	Preparing Graduate Fellowships	1 hours.	Spring	2023	3.93
591	Current Research in Geoscience	1 hours.	Spring	2023	3.95
591	Engineering Advanced Seminar	1 hours.	Spring	2023	3.95
591	Introductory Professional Development for Chemists	1 hours.	Spring	2023	3.95
591	Seminar	1 hours.	Spring	2023	3.95
590	Doctoral Research Seminar and Colloquium	1 hours.	Spring	2022	3.92
590	Doctoral Research Seminar and Colloquium	1 hours.	Spring	2022	3.95
590	Doctoral Research Seminar and Colloquium	1 hours.	Spring	2022	3.97
590	Doctoral Research Seminar and Colloquium	1 hours.	Spring	2023	3.92
590	Seminar	1 hours.	Spring	2022	3.92

15 rows in set (0.06 sec)

Figure 9 : Query 3

4. List the Number of Courses offered by the CS and ECE departments with an avg_grade greater than or equal to 3.5 using Union.

```
SELECT g.DepartmentCode, c.CourseNum, c.CourseName, c.CreditHours, g.Semester, g.Avg_grade
FROM GeneralCourse c
JOIN GPAHistory g ON g.CourseNum = c.CourseNum
WHERE g.DepartmentCode = 'CS' AND g.CourseNum LIKE '5%' AND g.Avg_grade >= 3.5
GROUP BY g.DepartmentCode, c.CourseNum, c.CourseName, c.CreditHours, g.Semester, g.Avg_grade

UNION

SELECT g.DepartmentCode, c.CourseNum, c.CourseName, c.CreditHours, g.Semester, g.Avg_grade
FROM GeneralCourse c
JOIN GPAHistory g ON g.CourseNum = c.CourseNum
WHERE g.DepartmentCode = 'ECE' AND g.CourseNum LIKE '5%' AND g.Avg_grade >= 3.5
GROUP BY g.DepartmentCode, c.CourseNum, c.CourseName, c.CreditHours, g.Semester, g.Avg_grade

ORDER BY Semester
LIMIT 15;
```

Output:

DepartmentCode	CourseNum	CourseName	CreditHours	Semester	Avg_grade
CS	527	Advanced Vitamins and Minerals: Regulations of Metabolism	3 hours.	Fall	3.83
CS	527	Advanced Price Analysis	4 hours.	Fall	3.74
CS	527	Graduate Level Trombone	2 TO 5 hours.	Fall	3.83
CS	527	Geospatial Artificial Intelligence & Machine Learning	4 hours.	Fall	3.83
CS	527	Seminar in 18th C Literature	4 hours.	Fall	3.74
CS	527	Topics in Software Engineering	4 hours.	Fall	3.74
CS	527	Seminar in 18th C Literature	4 hours.	Fall	3.83
CS	527	Plasma Technology of Gaseous Electronics	4 hours.	Fall	3.83
CS	527	MSFE Professional Development	0 hours.	Fall	3.83
CS	527	System-On-Chip Design	4 hours.	Fall	3.83
CS	527	Advanced Regression Analysis	4 hours.	Fall	3.83
ECE	549	Asymptotic Methods	4 hours.	Fall	3.52
CS	527	System-On-Chip Design	4 hours.	Fall	3.74
CS	527	Parasitology/Epidemiology Sem	1 hours.	Fall	3.83
CS	527	Topics in Software Engineering	4 hours.	Fall	3.83

15 rows in set (0.03 sec)

Figure 10 : Query 4

Indexing Analysis

Query 1

Indexing Analysis Report:

The initial query aims to identify professors who have taught courses with an average GPA greater than 3.5. These courses belong to departments and have course numbers starting with either '4' or '5'

(helping in assessing the performance of professors in teaching higher-level courses and their impact on students' academic performance)

On Running EXPLAIN ANALYZE on the first query following output was obtained:

```
| -> Limit: 15 row(s) (actual time=270.333..270.337 rows=15 loops=1)
|   -> Sort: g.Avg_Grade DESC, d.Department (actual time=270.332..270.335 rows=15 loops=1)
|     -> Filter: (avg(g.Avg_Grade) > 3.5) (actual time=265.305..265.322 rows=546 loops=1)
|       -> Table scan on <temporary> (actual time=264.491..264.732 rows=1006 loops=1)
|         -> Aggregate using temporary table (actual time=264.488..264.488 rows=1006 loops=1)
|           -> Nested loop inner join (cost=3805.89 rows=1700) (actual time=92.848..261.037 rows=1029 loops=1)
|             -> Nested loop inner join (cost=3210.83 rows=1700) (actual time=75.924..211.591 rows=1029 loops=1)
|               -> Nested loop inner join (cost=2615.77 rows=1700) (actual time=75.724..209.579 rows=1029 loops=1)
|                 -> Covering index scan on t using NetId (cost=316.11 rows=2979) (actual time=75.660..191.078 rows=2979 loops=1)
|                   -> Filter: (((g.CourseNum like '4%') or (g.CourseNum like '5%')) and (g.DepartmentCode is not null)) (cost=0.50 rows=1) (actual time=0.006..0.006 rows=0 loops=2979)
|                     -> Index lookup on g using PRIMARY (CRN=t.CRN) (cost=0.50 rows=3) (actual time=0.005..0.005 rows=1 loops=2979)
|                       -> Single-row index lookup on d using PRIMARY (DepartmentCode=g.DepartmentCode) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1029)
|                         -> Single-row index lookup on p using PRIMARY (NetId=t.NetId) (cost=0.25 rows=1) (actual time=0.048..0.048 rows=1 loops=1029)
```

Nested loop inner join cost: 3805.89

Indexing Design 1

Indexing GPAHistory(Avg_grade) for this query:

```
CREATE INDEX idx_gpa_history_avg_grade ON GPAHistory(Avg_grade);
```

After running for GPAHistory(Avg_grade)

```
| -> Limit: 15 row(s) (actual time=21.301..21.305 rows=15 loops=1)
|   -> Sort: g.Avg_Grade DESC, d.Department (actual time=21.300..21.303 rows=15 loops=1)
|     -> Filter: (avg(g.Avg_Grade) > 3.5) (actual time=19.415..19.883 rows=546 loops=1)
|       -> Table scan on <temporary> (actual time=19.409..19.621 rows=1006 loops=1)
|         -> Aggregate using temporary table (actual time=19.406..19.406 rows=1006 loops=1)
|           -> Nested loop inner join (cost=3805.89 rows=1700) (actual time=0.117..16.527 rows=1029 loops=1)
|             -> Nested loop inner join (cost=3210.83 rows=1700) (actual time=0.108..15.337 rows=1029 loops=1)
|               -> Nested loop inner join (cost=2615.77 rows=1700) (actual time=0.096..14.386 rows=1029 loops=1)
|                 -> Covering index scan on t using NetId (cost=316.11 rows=2979) (actual time=0.066..1.054 rows=2979 loops=1)
|                   -> Filter: (((g.CourseNum like '4%') or (g.CourseNum like '5%')) and (g.DepartmentCode is not null)) (cost=0.50 rows=1) (actual time=0.004..0.004 rows=0 loops=2979)
|                     -> Index lookup on g using PRIMARY (CRN=t.CRN) (cost=0.50 rows=3) (actual time=0.003..0.004 rows=1 loops=2979)
|                       -> Single-row index lookup on d using PRIMARY (DepartmentCode=g.DepartmentCode) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1029)
|                         -> Single-row index lookup on p using PRIMARY (NetId=t.NetId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1029)
```

Observations / Analysis:

- Nested loop inner join cost: 3805.89
- The cost did not change for this indexing, likely because most of the cost comes from the joins on attributes that are already indexed as primary keys.

Indexing Design 2

Indexing GPAHistory(CourseNum) for this query

```
CREATE INDEX idx_gpa_history_course_num ON GPAHistory(CourseNum)
```

After running for GPAHistory(CourseNum)

```
| -> Limit: 15 row(s) (actual time=23.337..23.340 rows=15 loops=1)
|   -> Sort: g.Avg_Grade DESC, d.Department (actual time=23.335..23.338 rows=15 loops=1)
|     -> Filter: (avg(g.Avg_Grade) > 3.5) (actual time=20.853..21.469 rows=546 loops=1)
|       -> Table scan on <temporary> (actual time=20.846..21.116 rows=1006 loops=1)
|         -> Aggregate using temporary table (actual time=20.843..20.843 rows=1006 loops=1)
|           -> Nested loop inner join (cost=3805.89 rows=1700) (actual time=0.224..17.550 rows=1029 loops=1)
|             -> Nested loop inner join (cost=3210.83 rows=1700) (actual time=0.212..16.295 rows=1029 loops=1)
|               -> Nested loop inner join (cost=2615.77 rows=1700) (actual time=0.191..15.324 rows=1029 loops=1)
|                 -> Covering index scan on t using NetId (cost=316.11 rows=2979) (actual time=0.154..1.280 rows=2979 loops=1)
|                   -> Filter: (((g.CourseNum like '4%') or (g.CourseNum like '5%')) and (g.DepartmentCode is not null)) (cost=0.50 rows=1) (actual time=0.004..0.005 rows=0 loops=2979)
|                     -> Index lookup on g using PRIMARY (CRN=t.CRN) (cost=0.50 rows=3) (actual time=0.003..0.004 rows=1 loops=2979)
|                       -> Single-row index lookup on d using PRIMARY (DepartmentCode=g.DepartmentCode) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1029)
|                         -> Single-row index lookup on p using PRIMARY (NetId=t.NetId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1029)
```

Observations / Analysis:

- Nested loop inner join cost: 3805.89
- The cost did not change for this indexing, likely because most of the cost comes from the joins on attributes that are already indexed as primary keys.

Indexing Design 3

Indexing GPAHistory(DepartmentCode) for this query:

```
CREATE INDEX idx_gpa_history_department_code ON GPAHistory(DepartmentCode);
```


After indexing GPAHistory(DepartmentCode)

```
| -> Limit: 15 row(s) (actual time=20.596..20.600 rows=15 loops=1)
|   -> Sort: g.Avg_Grade DESC, d.Department (actual time=20.595..20.598 rows=15 loops=1)
|     -> Filter: (avg(g.Avg_Grade) > 3.5) (actual time=18.750..19.199 rows=546 loops=1)
|       -> Table scan on <temporary> (actual time=18.743..18.950 rows=1006 loops=1)
|         -> Aggregate using temporary table (actual time=18.740..18.740 rows=1006 loops=1)
|           -> Nested loop inner join (cost=3805.89 rows=1700) (actual time=0.183..15.943 rows=1029 loops=1)
|             -> Nested loop inner join (cost=3210.83 rows=1700) (actual time=0.171..14.804 rows=1029 loops=1)
|               -> Nested loop inner join (cost=2615.77 rows=1700) (actual time=0.157..13.936 rows=1029 loops=1)
|                 -> Covering index scan on t using NetId (cost=316.11 rows=2979) (actual time=0.122..1.235 rows=2979 loops=1)
|                   -> Filter: ((g.CourseNum like '4%') or (g.CourseNum like '5%')) and (g.DepartmentCode is not null) (cost=0.50 rows=1) (actual time=0.004..0.004 rows=0 loops=29)
|                     -> Index lookup on g using PRIMARY (CRN=t.CRN) (cost=0.50 rows=3) (actual time=0.003..0.003 rows=1 loops=2979)
|                       -> Single-row index lookup on d using PRIMARY (DepartmentCode=g.DepartmentCode) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1029)
|                         -> Single-row index lookup on p using PRIMARY (NetId=t.NetId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1029)
```

Observations / Analysis:

- Nested loop inner join cost: 3805.89
- The cost did not change for this indexing, likely because most of the cost comes from the joins on attributes that are already indexed as primary keys.

Query 1 Final Analysis:

- None of our other indexing schemes helped decrease the cost
- We chose items that were in the GROUP BY portion of this query since the joins were on primary keys, but it did not seem to affect the performance in this case, likely because the GROUP BY didn't seem to have an affect on the cost with the primary keys also being included and used for the index lookups (seen in the screenshots).
- For simplicity, we will consider the original indexing the best.

Query 2:

Indexing Analysis Report:

Initially the query aims to rank departments based on their academic performance (average GPA of courses) and breadth of academic offerings (number of distinct courses), reflecting departmental academic standing.

On Running EXPLAIN ANALYZE on the second query following output was obtained:

```
| -> Limit: 15 row(s) (actual time=597.576..597.579 rows=15 loops=1)
|   -> Sort: CombinedScore DESC, DepartmentCode (actual time=597.575..597.577 rows=15 loops=1)
|     -> Table scan on <temporary> (cost=2.50..2.50 rows=0) (actual time=597.425..597.470 rows=187 loops=1)
|       -> Temporary table (cost=0.00..0.00 rows=0) (actual time=597.423..597.423 rows=187 loops=1)
|         -> Window aggregate: rank() OVER (ORDER BY ((ifnull(g.AverageGPA,0) * 0.7) + (ifnull(c.CourseCount,0) * 0.3)) desc) (actual time=597.174..597.341 rows=187 loops=1)
|           -> Sort: CombinedScore DESC (actual time=597.150..597.181 rows=187 loops=1)
|             -> Stream results (cost=91406.77 rows=0) (actual time=595.929..597.012 rows=187 loops=1)
|               -> Nested loop left join (cost=91406.77 rows=0) (actual time=595.911..596.699 rows=187 loops=1)
|                 -> Nested loop left join (cost=4324.70 rows=0) (actual time=39.319..39.749 rows=187 loops=1)
|                   -> Table scan on d (cost=18.95 rows=187) (actual time=0.085..0.186 rows=187 loops=1)
|                     -> Index lookup on g using <auto_key> (DepartmentCode=d.DepartmentCode) (actual time=0.211..0.211 rows=1 loops=187)
|                       -> Materialize CTE DepartmentGPA (cost=0.00..0.00 rows=0) (actual time=39.225..39.225 rows=135 loops=1)
|                         -> Table scan on <temporary> (actual time=38.946..38.980 rows=135 loops=1)
|                           -> Aggregate using temporary table (actual time=38.944..38.944 rows=135 loops=1)
|                             -> Table scan on GPAHistory (cost=1807.30 rows=17223) (actual time=0.639..17.807 rows=17224 loops=1)
|                               -> Index lookup on c using <auto_key> (DepartmentCode=d.DepartmentCode) (actual time=2.978..2.978 rows=1 loops=187)
|                                 -> Materialize CTE DepartmentCourses (cost=1311.52..1311.52 rows=3782) (actual time=556.579..556.579 rows=187 loops=1)
|                                   -> Group aggregate: count(distinct GeneralCourse.CourseNum) (cost=933.32 rows=3782) (actual time=554.280..556.086 rows=187 loops=1)
|                                     -> Sort: GeneralCourse.DepartmentCode (cost=555.12 rows=3782) (actual time=554.249..554.668 rows=4497 loops=1)
|                                       -> Index scan on GeneralCourse using PRIMARY (actual time=18.245..551.840 rows=4497 loops=1)
```

Stream results cost = 91,406.77

Indexing Design 1

Indexing CourseNum in the GeneralCourse table::

CREATE INDEX idx_general_course_course_num ON GeneralCourse (CourseNum);

After indexing CourseNum

```

| -> Limit: 15 row(s) (actual time=24.881..24.883 rows=15 loops=1)
    -> Sort: CombinedScore DESC, DepartmentCode (actual time=24.880..24.881 rows=15 loops=1)
        -> Table scan on <temporary> (cost=2.50..2.50 rows=0) (actual time=24.773..24.804 rows=187 loops=1)
            -> Temporary table (cost=0.00..0.00 rows=0) (actual time=24.772..24.772 rows=187 loops=1)
                -> Window aggregate: rank() OVER (ORDER BY ((ifnull(g.AverageGPA,0) * 0.7) + (ifnull(c.CourseCount,0) * 0.3)) desc) (actual time=24.611..24.717 rows=187 loops=1)
                    -> Sort: CombinedScore DESC (actual time=24.602..24.621 rows=187 loops=1)
                        -> Stream results (cost=91406.77 rows=0) (actual time=23.805..24.513 rows=187 loops=1)
                            -> Nested loop left join (cost=91406.77 rows=0) (actual time=23.784..24.287 rows=187 loops=1)
                                -> Nested loop left join (cost=4324.70 rows=0) (actual time=19.930..20.216 rows=187 loops=1)
                                    -> Table scan on d (cost=18.95 rows=187) (actual time=0.055..0.131 rows=187 loops=1)
                                        -> Index lookup on g using <auto_key0> (DepartmentCode=d.DepartmentCode) (actual time=0.107..0.107 rows=1 loops=187)
                                            -> Materialize CTE DepartmentGPA (cost=0.00..0.00 rows=0) (actual time=19.864..19.864 rows=135 loops=1)
                                                -> Table scan on <temporary> (actual time=19.701..19.725 rows=135 loops=1)
                                                    -> Aggregate using temporary table (actual time=19.699..19.699 rows=135 loops=1)
                                                        -> Table scan on GPAHistory (cost=1807.30 rows=17223) (actual time=0.081..0.972 rows=17294 loops=1)
                                                            -> Index lookup on c using <auto_key0> (DepartmentCode=d.DepartmentCode) (actual time=0.021..0.022 rows=1 loops=187)
                                                                -> Materialize CTE DepartmentCourses (cost=1206.85..1206.85 rows=3782) (actual time=3.844..3.844 rows=187 loops=1)
                                                                    -> Group aggregate: count(distinct GeneralCourse.CourseNum) (cost=828.65 rows=3782) (actual time=2.613..3.665 rows=187 loops=1)
                                                                        -> Sort: GeneralCourse.DepartmentCode (cost=450.45 rows=3782) (actual time=2.595..2.829 rows=4497 loops=1)
                                                                            -> Index scan on GeneralCourse using idx_general_course_course_num (actual time=0.025..1.417 rows=4497 loops=1)

```

Observations / Analysis

- Stream results cost = 91406.77
- This indexing scheme was the same as the original, and that is a bit surprising since the index we made is actually used (where the primary key is used in the original). This may be because the number of unique course numbers at the time of indexing was the same as the number of entries (corresponding to the same number of primary key values), but more in depth analysis is needed.

Indexing Design 2

Indexing DepartmentCode in the GPAHistory:

```
CREATE INDEX idx_gpa_history_department_code ON GPAHistory(DepartmentCode);
```

After indexing CourseNum

```

| -> Limit: 15 row(s) (actual time=273.917..273.919 rows=15 loops=1)
    -> Sort: CombinedScore DESC, DepartmentCode (actual time=273.916..273.918 rows=15 loops=1)
        -> Table scan on <temporary> (cost=2.50..2.50 rows=0) (actual time=273.822..273.852 rows=187 loops=1)
            -> Temporary table (cost=0.00..0.00 rows=0) (actual time=273.821..273.821 rows=187 loops=1)
                -> Window aggregate: rank() OVER (ORDER BY ((ifnull(g.AverageGPA,0) * 0.7) + (ifnull(c.CourseCount,0) * 0.3)) desc) (actual time=273.679..273.784 rows=187 loops=1)
                    -> Sort: CombinedScore DESC (actual time=273.670..273.690 rows=187 loops=1)
                        -> Stream results (cost=1218482595.07 rows=12180691182) (actual time=272.858..273.579 rows=187 loops=1)
                            -> Nested loop left join (cost=1218482595.07 rows=12180691182) (actual time=272.842..273.359 rows=187 loops=1)
                                -> Nested loop left join (cost=326394.80 rows=3220701) (actual time=37.354..37.636 rows=187 loops=1)
                                    -> Table scan on d (cost=18.95 rows=187) (actual time=0.059..0.123 rows=187 loops=1)
                                        -> Index lookup on g using <auto_key0> (DepartmentCode=d.DepartmentCode) (actual time=0.200..0.200 rows=1 loops=187)
                                            -> Materialize CTE DepartmentGPA (cost=5251.90..5251.90 rows=17223) (actual time=37.282..37.282 rows=135 loops=1)
                                                -> Group aggregate: avg(GPAHistory.Avg_Grade) (cost=3529.60 rows=17223) (actual time=2.280..37.022 rows=135 loops=1)
                                                    -> Index scan on GPAHistory using idx_gpa_history_department_code (cost=1807.30 rows=17223) (actual time=1.702..33.967 rows=17294 loops=1)
                                                        -> Index lookup on c using <auto_key0> (DepartmentCode=d.DepartmentCode) (actual time=1.260..1.260 rows=1 loops=187)
                                                            -> Materialize CTE DepartmentCourses (cost=1206.85..1206.85 rows=3782) (actual time=235.478..235.478 rows=187 loops=1)
                                                                -> Group aggregate: count(distinct GeneralCourse.CourseNum) (cost=828.65 rows=3782) (actual time=234.149..235.302 rows=187 loops=1)
                                                                    -> Sort: GeneralCourse.DepartmentCode (cost=450.45 rows=3782) (actual time=234.130..234.387 rows=4497 loops=1)
                                                                        -> Index scan on GeneralCourse using PRIMARY (actual time=0.022..232.466 rows=4497 loops=1)

```

Observations / Analysis

- Stream results cost = 1,218,482,595.07
- We are surprised by these results because the output is very similar. The materialize CTE step is where the issues lie in this one because the costs on the others for this step are 0, but they are very high for this query, so indexing GPAHistory with the DepartmentCode increased the overhead needed for that step significantly somehow.

Indexing Design 3

Indexing DepartmentCode in the General Course:

```
CREATE INDEX idx_general_course_department_code ON
GeneralCourse(DepartmentCode);
```

After Indexing DepartmentCode in the General Course:

```

| -> Limit: 15 row(s) (actual time=25.506..25.509 rows=15 loops=1)
    -> Sort: CombinedScore DESC, DepartmentCode (actual time=25.505..25.507 rows=15 loops=1)
        -> Table scan on <temporary> (cost=2.50..2.50 rows=0) (actual time=25.406..25.439 rows=187 loops=1)
            -> Temporary table (cost=0.00..0.00 rows=0) (actual time=25.405..25.405 rows=187 loops=1)
                -> Window aggregate: rank() OVER (ORDER BY ((ifnull(g.AverageGPA,0) * 0.7) + (ifnull(c.CourseCount,0) * 0.3)) desc ) (actual time=25.237..25.343 rows=187 loops=1)
                    -> Sort: CombinedScore DESC (actual time=25.229..25.249 rows=187 loops=1)
                        -> Stream results (cost=91429.79 rows=0) (actual time=24.255..25.130 rows=187 loops=1)
                            -> Nested loop left join (cost=91429.79 rows=0) (actual time=24.241..24.878 rows=187 loops=1)
                                -> Nested loop left join (cost=4324.70 rows=0) (actual time=21.299..21.678 rows=187 loops=1)
                                    -> Table scan on d (cost=18.95 rows=187) (actual time=0.054..0.206 rows=187 loops=1)
                                        -> Index lookup on g using <auto_key0> (DepartmentCode=d.DepartmentCode) (actual time=0.114..0.115 rows=1 loops=187)
                                            -> Materialize CTE DepartmentGPA (cost=0.00..0.00 rows=0) (actual time=21.236..21.236 rows=135 loops=1)
                                                -> Table scan on <temporary> (actual time=21.033..21.059 rows=135 loops=1)
                                                    -> Aggregate using temporary table (actual time=21.031..21.031 rows=135 loops=1)
                                                        -> Table scan on GPAHistory (cost=1807.30 rows=17223) (actual time=0.063..9.336 rows=17294 loops=1)
                                                            -> Index lookup on c using <auto_key0> (DepartmentCode=d.DepartmentCode) (actual time=0.017..0.017 rows=1 loops=187)
                                                                -> Materialize CTE DepartmentCourses (cost=1134.90..1134.90 rows=3783) (actual time=2.929..2.929 rows=187 loops=1)
                                                                    -> Group aggregate: count(distinct GeneralCourse.CourseNum) (cost=756.60 rows=3783) (actual time=0.075..2.748 rows=187 loops=1)
                                                                        -> Covering index skip scan for deduplication on GeneralCourse using idx_general_course_department_code (cost=378.30 rows=3783) (actual time=0.049..2.277 rows=4497 loops=1)
|

```

Observations / Analysis

- Stream results cost = 91429.79
- This indexing scheme resulted in slightly worse results than the original, and this is a bit strange because certain parts of the query are better than the original, but it seems to be made up in some of the stranger index lookups.

Query 2 Final Analysis:

- None of our other indexing schemes helped decrease the cost, but some increased the cost
- This query likely didn't go down because most of the work was done with the primary keys, but we are still a little perplexed by the significantly increased cost for the 2nd indexing because that is something used in the joins in this query, so that is something that needs further analysis.
- For simplicity, we will consider the original indexing the best.

Query 3:

Indexing Analysis Report:

The query aims to List the number of 1 Credit courses offered by the CS department having avg_grade > 3.9.

```

| -> Limit: 15 row(s) (actual time=14.828..14.828 rows=0 loops=1)
    -> Sort: g.CourseNum DESC (actual time=14.826..14.826 rows=0 loops=1)
        -> Filter: (g.Avg_Grade >= 3.90) (actual time=14.813..14.813 rows=0 loops=1)
            -> Table scan on <temporary> (cost=2187.63..2193.32 rows=258) (actual time=14.812..14.812 rows=0 loops=1)
                -> Temporary table with deduplication (cost=2187.61..2187.61 rows=258) (actual time=14.806..14.806 rows=0 loops=1)
                    -> Nested loop inner join (cost=2161.81 rows=258) (actual time=14.782..14.782 rows=0 loops=1)
                        -> Filter: ((g.'Year' in (2022,2023)) and (g.Semester like ' Sp % ')) and (g.CourseNum is not null)) (cost=1807.30 rows=383) (actual time=14.780..14.780 rows=0 loops=1)
                            -> Table scan on g (cost=1807.30 rows=17223) (actual time=0.145..12.812 rows=17294 loops=1)
                                -> Filter: (c.CreditHours = '1 hours.') (cost=0.25 rows=1) (never executed)
                                    -> Index lookup on c using PRIMARY (CourseNum=g.CourseNum) (cost=0.25 rows=7) (never executed)
|

```

Table scan cost: 2193.32

Indexing Design 1

Indexing GPAHistory(Avg_grade) for this query:

```
CREATE INDEX idx_GPAHistory_Avg_Grade ON GPAHistory(Avg_Grade);
```

After running for GPAHistory(Avg_grade)

```

-> Sort: g.CourseNum DESC (actual time=10.115..10.115 rows=0 loops=1)
-> Filter: (g.Avg_Grade >= 3.90) (actual time=10.107..10.107 rows=0 loops=1)
-> Table scan on <temporary> (cost=2187.63..2193.32 rows=258) (actual time=10.106..10.106 rows=0 loops=1)
-> Temporary table with deduplication (cost=2187.61..2187.61 rows=258) (actual time=10.102..10.102 rows=0 loops=1)
-> Nested loop inner join (cost=2161.81 rows=258) (actual time=10.084..10.084 rows=0 loops=1)
-> Filter: ((g.'Year' in (2022,2023)) and (g.Semester like ' Sp % ')) and (g.CourseNum is not null)) (cost=1807.30 rows=383) (actual time=10.082..10.082 rows=0 loops=1)
-> Table scan on g (cost=1807.30 rows=17223) (actual time=0.138..8.438 rows=17294 loops=1)
-> Filter: (c.CreditHours = '1 hours.') (cost=0.25 rows=1) (never executed)
-> Index lookup on c using PRIMARY (CourseNum=g.CourseNum) (cost=0.25 rows=7) (never executed)

```

Observations / Analysis

- Table scan cost: 2193.32
- The cost did not change for this indexing, likely because most of the cost comes from the joins on attributes that are already indexed as primary keys. The new index we chose did not have an effect on which indices were used for lookup in this case (the primary keys were still used).

Indexing Design 2

Indexing GeneralCourse(CourseName) for this query:

```
CREATE INDEX idx_GeneralCourse_CourseName1 ON
GeneralCourse (CourseName) ;
```

After running for GeneralCourse(CourseName)

```

-> Sort: g.CourseNum DESC (actual time=9.923..9.923 rows=0 loops=1)
-> Filter: (g.Avg_Grade >= 3.90) (actual time=9.915..9.915 rows=0 loops=1)
-> Table scan on <temporary> (cost=2187.63..2193.32 rows=258) (actual time=9.914..9.914 rows=0 loops=1)
-> Temporary table with deduplication (cost=2187.61..2187.61 rows=258) (actual time=9.910..9.910 rows=0 loops=1)
-> Nested loop inner join (cost=2161.81 rows=258) (actual time=9.894..9.894 rows=0 loops=1)
-> Filter: ((g.'Year' in (2022,2023)) and (g.Semester like 'Sp %') and (g.CourseNum is not null)) (cost=1807.30 rows=383) (actual time=9.893..9.893 rows=0 loops=1)
-> Table scan on g (cost=1807.30 rows=1723) (actual time=0.062..0.267 rows=1724 loops=1)
-> Filter: (c.CreditHours = '1 hours.') (cost=0.25 rows=1) (never executed)
-> Index lookup on c using PRIMARY (CourseNum=g.CourseNum) (cost=0.25 rows=7) (never executed)

```

Observations / Analysis

- Table scan cost: 2193.32
- The cost did not change for this indexing, likely because most of the cost comes from the joins on attributes that are already indexed as primary keys. The new index we chose did not have an effect on which indices were used for lookup in this case (the primary keys were still used).

Indexing Design 3

Indexing GeneralCourse(CreditHours) for this query:

```
CREATE INDEX idx_GeneralCourse_CreditHours1 ON
GeneralCourse (CreditHours) ;
```

After running for GeneralCourse(CreditHours)

```

-> Limit: 15 row(s) (actual time=10.055..10.055 rows=0 loops=1)
-> Sort: g.CourseNum DESC (actual time=10.054..10.054 rows=0 loops=1)
-> Filter: (g.Avg_Grade >= 3.90) (actual time=10.047..10.047 rows=0 loops=1)
-> Table scan on <temporary> (cost=2164.59..2176.99 rows=794) (actual time=10.046..10.046 rows=0 loops=1)
-> Temporary table with deduplication (cost=2164.57..2164.57 rows=794) (actual time=10.042..10.042 rows=0 loops=1)
-> Nested loop inner join (cost=2085.18 rows=794) (actual time=10.026..10.026 rows=0 loops=1)
-> Filter: ((g.'Year' in (2022,2023)) and (g.Semester like 'Sp %') and (g.CourseNum is not null)) (cost=1807.30 rows=383) (actual time=10.025..10.025 rows=0 loops=1)
-> Table scan on g (cost=1807.30 rows=1723) (actual time=0.061..0.369 rows=1724 loops=1)
-> Index lookup on c using idx_GeneralCourse_CreditHours1 (CreditHours='1 hours.', CourseNum=g.CourseNum) (cost=0.52 rows=2) (never executed)

```

Observations / Analysis

- Table scan cost: 2176.99
- This index results in a slightly lower cost than the original.
- One can see that unlike the other results, the index we created was actually used in place of the primary key for the index lookup, which is where we get our slightly lower cost.

Query 3 Final Analysis:

- Only the third indexing scheme reduced the cost
- Similar to Query 1, the first two indexing schemes used attributes that were in the GROUP BY portion of the query, which had no effect on the output. However, since CreditHours was also included in the WHERE clause, we think that is why the third indexing scheme was slightly improved from the original.

- The third indexing is the best for this query.

Query 4:

Indexing analysis report:

The Query aims to List the Number of Courses offered by the CS and ECE departments with an avg_grade greater than or equal to 3.5 using Union operator.

```
| -> Limit: 15 row(s) (cost=4025.55..4025.55 rows=15) (actual time=29.842..29.848 rows=15 loops=1)
  -> Sort: Semester, limit input to 15 row(s) per chunk (cost=4025.55..4025.55 rows=15) (actual time=29.841..29.844 rows=15 loops=1)
    -> Table scan on <union temporary> (cost=3920.46..3920.46 rows=860) (actual time=28.160..28.431 rows=1717 loops=1)
      -> Union materialize with deduplication (cost=3920.46..3920.46 rows=860) (actual time=28.156..28.156 rows=1717 loops=1)
        -> Table scan on <temporary> (cost=1909.39..1917.24 rows=430) (actual time=14.673..14.853 rows=1132 loops=1)
          -> Temporary table with deduplication (cost=1909.37..1909.37 rows=430) (actual time=14.670..14.670 rows=1132 loops=1)
            -> Nested loop inner join (cost=1866.38 rows=430) (actual time=1.186..12.736 rows=1721 loops=1)
              -> Filter: ((g.DepartmentCode = 'CS') and (g.CourseNum like '5%') and (g.Avg_Grade >= 3.50) and (g.CourseNum is not null)) (cost=1807.30 rows=64) (actual time=1.151..11.345 rows=70 loops=1)
                -> Table scan on g (cost=1807.30 rows=17223) (actual time=0.089..9.665 rows=17294 loops=1)
                  -> Index lookup on c using PRIMARY (CourseNum=g.CourseNum) (cost=0.26 rows=7) (actual time=0.004..0.018 rows=25 loops=70)
                -> Table scan on <temporary> (cost=1909.39..1917.24 rows=430) (actual time=11.541..11.639 rows=585 loops=1)
                  -> Temporary table with deduplication (cost=1909.37..1909.37 rows=430) (actual time=11.538..11.538 rows=585 loops=1)
                    -> Nested loop inner join (cost=1866.38 rows=430) (actual time=0.254..10.836 rows=659 loops=1)
                      -> Filter: ((g.DepartmentCode = 'ECE') and (g.CourseNum like '5%') and (g.Avg_Grade >= 3.50) and (g.CourseNum is not null)) (cost=1807.30 rows=64) (actual time=0.231..10.137 rows=56 loops=1)
                        -> Table scan on g (cost=1807.30 rows=17223) (actual time=0.071..8.507 rows=17294 loops=1)
                          -> Index lookup on c using PRIMARY (CourseNum=g.CourseNum) (cost=0.26 rows=7) (actual time=0.004..0.011 rows=12 loops=56)
```

Limit cost: 4025.55

Indexing Design 1

Indexing GPAHistory(Avg_grade) for this query:

```
CREATE INDEX idx_GPAHistory_Avg_Grade ON GPAHistory(Avg_Grade);
```

After running for GPAHistory(Avg_grade)

```
| -> Limit: 15 row(s) (cost=4224.39..4224.39 rows=15) (actual time=28.233..28.236 rows=15 loops=1)
  -> Sort: Semester, limit input to 15 row(s) per chunk (cost=4224.39..4224.39 rows=15) (actual time=28.231..28.233 rows=15 loops=1)
    -> Table scan on <union temporary> (cost=4070.94..4089.54 rows=1290) (actual time=27.578..27.871 rows=1717 loops=1)
      -> Union materialize with deduplication (cost=4070.92..4070.92 rows=1290) (actual time=27.577..27.577 rows=1717 loops=1)
        -> Table scan on <temporary> (cost=1960.43..1970.97 rows=645) (actual time=13.801..13.977 rows=1132 loops=1)
          -> Temporary table with deduplication (cost=1960.42..1960.42 rows=645) (actual time=13.797..13.797 rows=1132 loops=1)
            -> Nested loop inner join (cost=1895.92 rows=645) (actual time=1.175..12.099 rows=1721 loops=1)
              -> Filter: ((g.DepartmentCode = 'CS') and (g.CourseNum like '5%') and (g.Avg_Grade >= 3.50) and (g.CourseNum is not null)) (cost=1807.30 rows=96) (actual time=1.150..10.847 rows=70 loops=1)
                -> Table scan on g (cost=1807.30 rows=17223) (actual time=0.126..9.259 rows=17294 loops=1)
                  -> Index lookup on c using PRIMARY (CourseNum=g.CourseNum) (cost=0.26 rows=7) (actual time=0.004..0.016 rows=25 loops=70)
                -> Table scan on <temporary> (cost=1960.43..1970.97 rows=645) (actual time=11.839..11.950 rows=585 loops=1)
                  -> Temporary table with deduplication (cost=1960.42..1960.42 rows=645) (actual time=11.836..11.836 rows=585 loops=1)
                    -> Nested loop inner join (cost=1895.92 rows=645) (actual time=0.244..11.112 rows=659 loops=1)
                      -> Filter: ((g.DepartmentCode = 'ECE') and (g.CourseNum like '5%') and (g.Avg_Grade >= 3.50) and (g.CourseNum is not null)) (cost=1807.30 rows=96) (actual time=0.223..10.503 rows=56 loops=1)
                        -> Table scan on g (cost=1807.30 rows=17223) (actual time=0.068..8.840 rows=17294 loops=1)
                          -> Index lookup on c using PRIMARY (CourseNum=g.CourseNum) (cost=0.26 rows=7) (actual time=0.004..0.010 rows=12 loops=56)
```

Observations / Analysis

- Limit cost: 4224.39
- This index caused an increase in the cost by about 200.
- Many of the operations within the query increased their cost by a small amount, leading to the overall increase. This may be due to the increased cost of reading the index into memory, which for a relatively small dataset like this, is considerable relative to the overall cost of the query.

Indexing Design 2

Indexing GeneralCourse(CourseName) for this query:

```
CREATE INDEX idx_GeneralCourse_CourseName ON
GeneralCourse(CourseName);
```

After Running for GeneralCourse(CourseName)

```

| -> Limit: 15 row(s) (cost=4025.55..4025.55 rows=15) (actual time=29.135..29.138 rows=15 loops=1)
    -> Sort: Semester, limit input to 15 row(s) per chunk (cost=4025.55..4025.55 rows=15) (actual time=29.134..29.136 rows=15 loops=1)
        -> Table scan on <union temporary> (cost=3920.46..3933.70 rows=860) (actual time=28.486..28.791 rows=1717 loops=1)
            -> Union materialize with deduplication (cost=3920.46..3920.46 rows=860) (actual time=28.484..28.484 rows=1717 loops=1)
                -> Table scan on <temporary> (cost=1909.39..1917.24 rows=430) (actual time=14.672..14.847 rows=1132 loops=1)
                    -> Temporary table with deduplication (cost=1909.37..1909.37 rows=430) (actual time=14.668..14.668 rows=1132 loops=1)
                        -> Nested loop inner join (cost=1866.38 rows=430) (actual time=1.192..12.776 rows=1721 loops=1)
                            -> Filter: ((g.DepartmentCode = 'CS') and (g.CourseNum like '5%') and (g.Avg_Grade >= 3.50) and (g.CourseNum is not null)) (cost=1807.30 rows=64) (actual time=1.170..11.413 rows=70 loops=1)
                                -> Table scan on g (cost=1807.30 rows=17223) (actual time=0.196..9.376 rows=17294 loops=1)
                                    -> Index lookup on c using PRIMARY (CourseNum=g.CourseNum) (cost=0.26 rows=7) (actual time=0.004..0.017 rows=25 loops=70)
                                -> Table scan on <temporary> (cost=1909.39..1917.24 rows=430) (actual time=11.835..11.971 rows=585 loops=1)
                                    -> Temporary table with deduplication (cost=1909.37..1909.37 rows=430) (actual time=11.830..11.830 rows=585 loops=1)
                                        -> Nested loop inner join (cost=1866.38 rows=430) (actual time=0.248..11.164 rows=659 loops=1)
                                            -> Filter: ((g.DepartmentCode = 'ECE') and (g.CourseNum like '5%') and (g.Avg_Grade >= 3.50) and (g.CourseNum is not null)) (cost=1807.30 rows=64) (actual time=0.233..10.562 rows=56 loops=1)
                                                -> Table scan on g (cost=1807.30 rows=17223) (actual time=0.079..8.887 rows=17294 loops=1)
                                                    -> Index lookup on c using PRIMARY (CourseNum=g.CourseNum) (cost=0.26 rows=7) (actual time=0.004..0.010 rows=12 loops=56)

```

Observations / Analysis

- Limit cost: 4025.55
- This query costs the exact same as the original. Since CourseName is only used in the Group By clause, indexing by it did not induce any effect on the query.

Indexing Design 3

Indexing GeneralCourse(CreditHours) for this query:

```

CREATE INDEX idx_GeneralCourse_CreditHours ON
GeneralCourse(CreditHours);

```

After Running for GeneralCourse(CreditHours)

```

| -> Limit: 15 row(s) (cost=4025.55..4025.55 rows=15) (actual time=28.329..28.332 rows=15 loops=1)
    -> Sort: Semester, limit input to 15 row(s) per chunk (cost=4025.55..4025.55 rows=15) (actual time=28.328..28.330 rows=15 loops=1)
        -> Table scan on <union temporary> (cost=3920.46..3933.70 rows=860) (actual time=27.773..28.019 rows=1717 loops=1)
            -> Union materialize with deduplication (cost=3920.46..3920.46 rows=860) (actual time=27.771..27.771 rows=1717 loops=1)
                -> Table scan on <temporary> (cost=1909.39..1917.24 rows=430) (actual time=13.701..13.902 rows=1132 loops=1)
                    -> Temporary table with deduplication (cost=1909.37..1909.37 rows=430) (actual time=13.697..13.697 rows=1132 loops=1)
                        -> Nested loop inner join (cost=1866.38 rows=430) (actual time=1.125..11.897 rows=1721 loops=1)
                            -> Filter: ((g.DepartmentCode = 'CS') and (g.CourseNum like '5%') and (g.Avg_Grade >= 3.50) and (g.CourseNum is not null)) (cost=1807.30 rows=64) (actual time=1.105..10.542 rows=70 loops=1)
                                -> Table scan on g (cost=1807.30 rows=17223) (actual time=0.070..8.834 rows=17294 loops=1)
                                    -> Index lookup on c using PRIMARY (CourseNum=g.CourseNum) (cost=0.26 rows=7) (actual time=0.004..0.018 rows=25 loops=70)
                                -> Table scan on <temporary> (cost=1909.39..1917.24 rows=430) (actual time=12.066..12.193 rows=585 loops=1)
                                    -> Temporary table with deduplication (cost=1909.37..1909.37 rows=430) (actual time=12.063..12.063 rows=585 loops=1)
                                        -> Nested loop inner join (cost=1866.38 rows=430) (actual time=0.255..11.389 rows=659 loops=1)
                                            -> Filter: ((g.DepartmentCode = 'ECE') and (g.CourseNum like '5%') and (g.Avg_Grade >= 3.50) and (g.CourseNum is not null)) (cost=1807.30 rows=64) (actual time=0.224..10.781 rows=56 loops=1)
                                                -> Table scan on g (cost=1807.30 rows=17223) (actual time=0.070..9.028 rows=17294 loops=1)
                                                    -> Index lookup on c using PRIMARY (CourseNum=g.CourseNum) (cost=0.26 rows=7) (actual time=0.004..0.010 rows=12 loops=56)

```

Observations / Analysis

- Limit cost: 4025.55
- This index did not change the performance of the query at all. Once again, CreditHours is only used in the Group By clause, and we believe that due to this, it did not have an effect on the overall performance of the query.

Query 4 Final Analysis:

None of the proposed indices reduced the cost. Due to this, we would choose not to implement any of these indices. However, as we continue with the project, we can experiment with different indices for our finalized queries used in our application to see if any others could have a positive effect on this query.

Overall Index Design:

Most of our index choices did not improve the cost. However, one index, on GeneralCourse(CreditHours), did improve the cost of Query 3 a bit. Since that was the only index we saw improvement on, and also didn't reduce the performance of other queries, we will use this index going forward in our database. We will continue to evaluate and analyze the performance of different indices as we determine the final structure of the queries we will use in our application.