

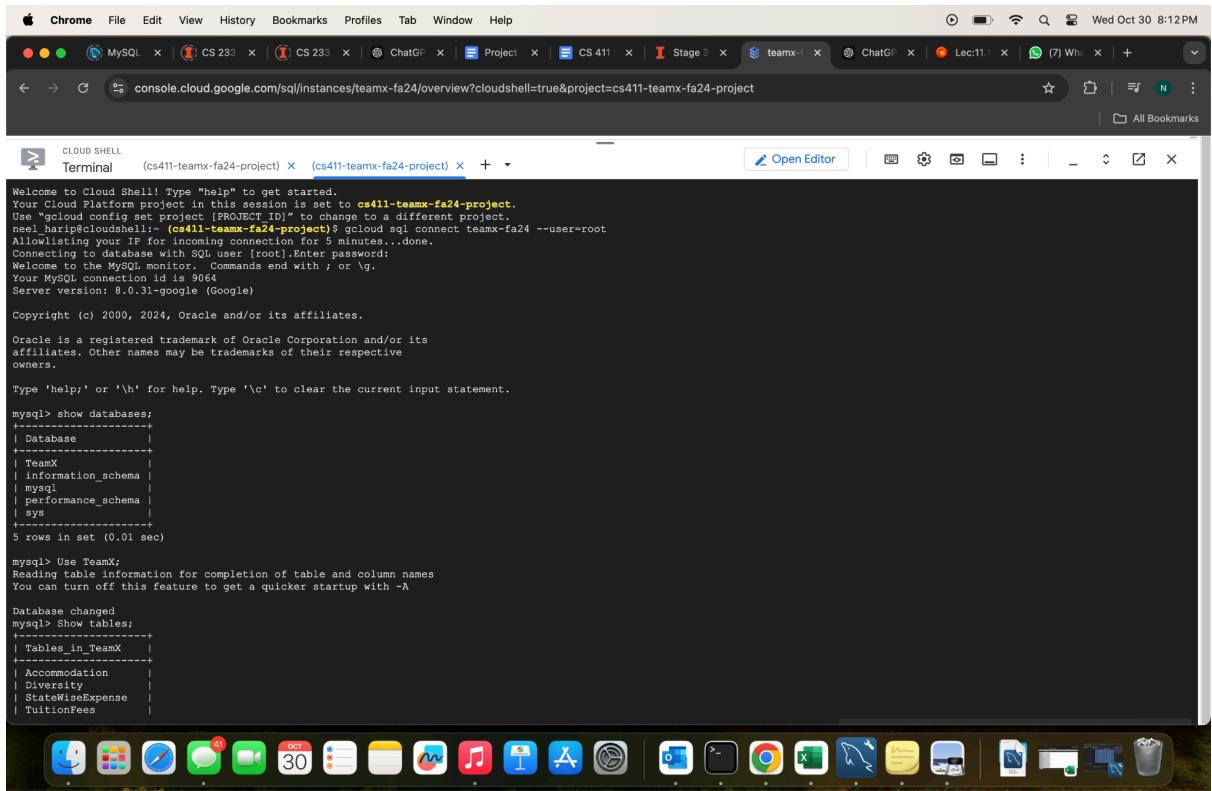
# Stage 3: Database Design

## Project Title: GlobalScholar: International Student Financial Navigator

Group Members: Neel Harip(nhari7), Sakshil Verma(sakshil2),  
Sejal Pekam(spekam2), Sharvari Gadiwan(sgadi5)

### 1. Database Implementation

#### 1.1 GCP Connection



```
Welcome to Cloud Shell! Type "help" to get started.  
Your Cloud Platform project in this session is set to cs411-teamx-fa24-project.  
Use "gcloud config set project [PROJECT_ID]" to change to a different project.  
neel.harip@cloudshell:  
[cs411-teamx-fa24-project]$ gcloud sql connect teamx-fa24 --user=root  
Allow root login. IP or port: 3306  
Connecting to database with SQL user [root].Enter password:  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 9064  
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 databases;  
+-----+  
| Database |  
+-----+  
| TeamX |  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
+-----+  
5 rows in set (0.01 sec)  
  
mysql> Use TeamX;  
Reading table information for completion of table and column names  
You can turn off this feature to get a quicker startup with -A  
  
Database changed  
mysql> Show tables;  
+-----+  
| Tables_in_TeamX |  
+-----+  
| Accommodation |  
| Diversity |  
| StatewiseExpense |  
| TuitionFees |
```

#### 1.2 DDL Commands For Table Creation

##### 1. User

```
CREATE TABLE User (  
    Id INT PRIMARY KEY,  
    FirstName VARCHAR(255),  
    LastName VARCHAR(255),  
    EmailId VARCHAR(255),  
    Password VARCHAR(255),
```

```
TuitionFeeBudget INT,  
AccommodationBudget INT  
);
```

## 2. University

```
CREATE TABLE University (  
    UniversityName VARCHAR(255) PRIMARY KEY,  
    State VARCHAR(100)  
);
```

## 3. Diversity

```
CREATE TABLE Diversity (  
    UniversityName VARCHAR(255),  
    TotalEnrollment INT,  
    RaceCategory VARCHAR(100),  
    RaceWiseEnrollment INT,  
    PRIMARY KEY (UniversityName, RaceCategory),  
    FOREIGN KEY (UniversityName) REFERENCES University(UniversityName)  
);
```

## 4. StatewiseExpense

```
CREATE TABLE StateWiseExpense (  
    State VARCHAR(100) PRIMARY KEY,  
    HousingCost INT,  
    FoodCost INT,  
    TransportationCost INT,  
    HealthcareCost INT  
);
```

## 5. Accommodation

```
CREATE TABLE Accommodation (  
    UniversityName VARCHAR(255),  
    DegreeLength VARCHAR(255),  
    RoomAndBoardCost INT,  
    PRIMARY KEY (UniversityName),  
    FOREIGN KEY (UniversityName) REFERENCES University(UniversityName)  
);
```

## 6. TuitionFees

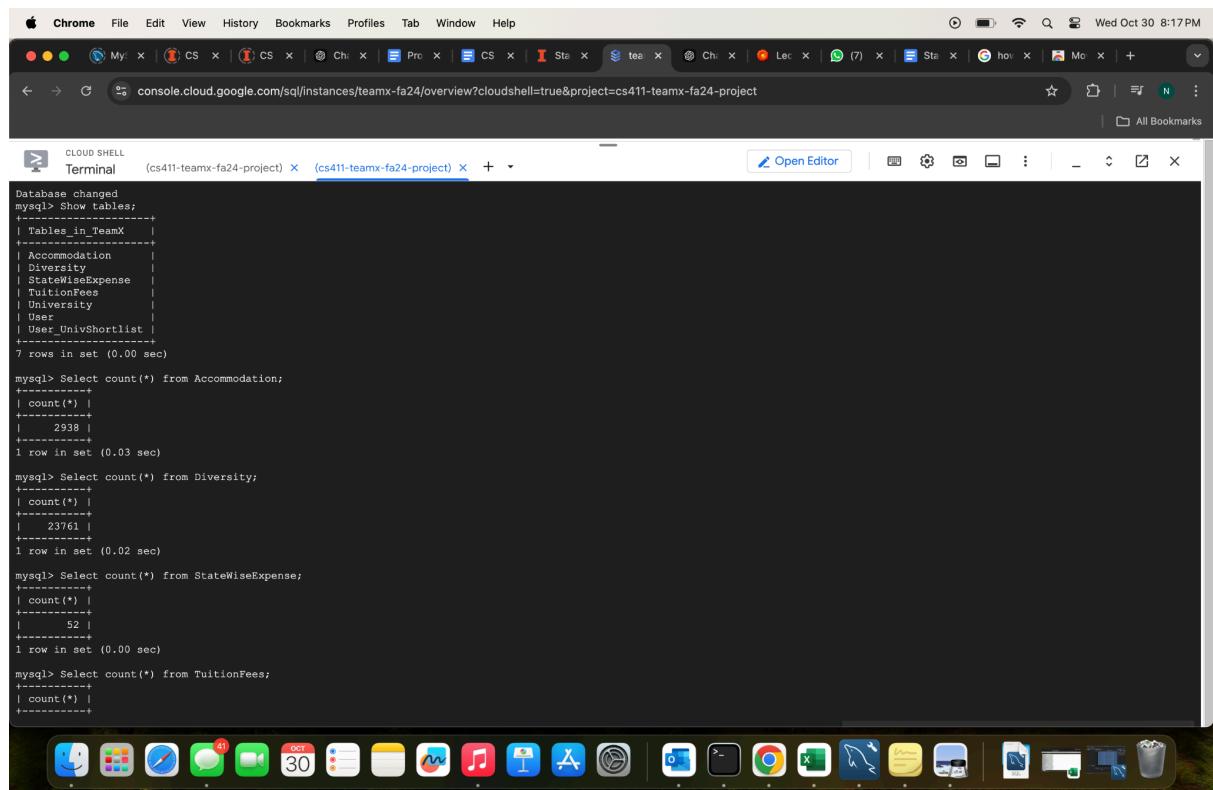
```
CREATE TABLE TuitionFees (  
    UniversityName VARCHAR(255),  
    UniversityType VARCHAR(100),  
    DegreeLength INT,  
    InStateTuitionFees INT,  
    OutOfStateTuitionFees INT,  
    PRIMARY KEY (UniversityName),
```

```
FOREIGN KEY (UniversityName) REFERENCES University(UniversityName)
);
```

## 7. User\_UnivShortlist

```
CREATE TABLE User_UnivShortlist (
    UserId INT,
    UniversityName VARCHAR(255),
    PRIMARY KEY (UserId, UniversityName),
    FOREIGN KEY (UserId) REFERENCES User(Id),
    FOREIGN KEY (UniversityName) REFERENCES University(UniversityName)
);
```

## 1.3 Count Query for Tables



The screenshot shows a macOS desktop with a terminal window open in a browser-based cloud shell environment. The terminal is titled 'Terminal' and shows a MySQL session connected to a database named 'TeamX'. The user is executing several 'SELECT COUNT(\*)' queries against tables 'Accommodation', 'Diversity', 'StateWiseExpense', 'TuitionFees', and 'User'. The results for each query are displayed in the terminal.

```
Database changed
mysql> Show tables;
+----------------+
| Tables_in_TeamX |
+----------------+
| Accommodation   |
| Diversity       |
| StateWiseExpense |
| TuitionFees     |
| University      |
| User             |
| User_UnivShortlist |
+----------------+
7 rows in set (0.00 sec)

mysql> Select count(*) from Accommodation;
+-----+
| count(*) |
+-----+
| 2938 |
+-----+
1 row in set (0.03 sec)

mysql> Select count(*) from Diversity;
+-----+
| count(*) |
+-----+
| 23761 |
+-----+
1 row in set (0.02 sec)

mysql> Select count(*) from StateWiseExpense;
+-----+
| count(*) |
+-----+
| 52 |
+-----+
1 row in set (0.00 sec)

mysql> Select count(*) from TuitionFees;
+-----+
| count(*) |
+-----+
1 row in set (0.00 sec)

mysql> Select count(*) from User;
+-----+
| count(*) |
+-----+
```

```

CLOUD SHELL (cs411-teamx-fa24-project) x (cs411-teamx-fa24-project) x + v
1 row in set (0.02 sec)
mysql> Select count(*) from StateWiseExpense;
+-----+
| count(*) |
+-----+
|      52 |
+-----+
1 row in set (0.00 sec)

1 row in set (0.08 sec)
mysql> Select count(*) from TuitionFees;
+-----+
| count(*) |
+-----+
|    2938 |
+-----+
1 row in set (0.08 sec)

mysql> Select count(*) from University;
+-----+
| count(*) |
+-----+
|    2938 |
+-----+
1 row in set (0.01 sec)

mysql> Select count(*) from User;
+-----+
| count(*) |
+-----+
|     991 |
+-----+
1 row in set (0.04 sec)

mysql> Select count(*) from User_UnivShortlist;
+-----+
| count(*) |
+-----+
|    1380 |
+-----+
1 row in set (0.07 sec)

mysql> []

```

## 2. Advanced Queries

### 2.1 Query 1

This query helps GlobalScholar identify users whose tuition budgets are below the average in-state university tuition, highlighting potential financial challenges.

```

SELECT U.FirstName, U.LastName, U.TuitionFeeBudget,
       (SELECT AVG(InStateTuitionFees) FROM TuitionFees) as avg_state_tuition
FROM User U
JOIN User_UnivShortlist US ON U.Id = US.UserId
JOIN TuitionFees T ON US.UniversityName = T.UniversityName
WHERE U.TuitionFeeBudget < (
    SELECT AVG(InStateTuitionFees)
    FROM TuitionFees
);

```

```

mysql> SELECT U.FirstName, U.LastName, U.TuitionFeeBudget, (SELECT AVG(InStateTuitionFees) FROM TuitionFees) as avg_state_tuition FROM User U JOIN User_UnivShortlist US ON U.Id = US.UserId JOIN TuitionFees T ON US.UniversityName = T.UniversityName WHERE U.TuitionFeeBudget < (    SELECT AVG(InStateTuitionFees)    FROM TuitionFees ) Limit 15;
+-----+-----+-----+-----+
| FirstName | LastName | TuitionFeeBudget | avg_state_tuition |
+-----+-----+-----+-----+
| William14 | Anderson |        14837 |      16404.2063 |
| Robert116 | Thomas   |         10907 |      16404.2063 |
| Laura     | Martin   |        14645 |      16404.2063 |
| Patricia126 | Moore |        12284 |      16404.2063 |
| Sarah     | Davis   |        16000 |      16404.2063 |
| William103 | Thomas   |        15863 |      16404.2063 |
| Lisa      | Miller   |        16200 |      16404.2063 |
| William114 | Anderson |        14837 |      16404.2063 |
| William   | Jones   |         11251 |      16404.2063 |
| Laura     | Martin   |        14645 |      16404.2063 |
| Sarah     | Davis   |        14831 |      16404.2063 |
| Alex113   | Martinez |        12969 |      16404.2063 |
| Robert    | Garcia   |        16339 |      16404.2063 |
| Joseph    | Hernandez |        12245 |      16404.2063 |
| William   | Lee     |         11010 |      16404.2063 |
+-----+-----+-----+-----+
15 rows in set (0.00 sec)

```

## 2.2 Query 2

This query calculates the average living costs for universities shortlisted by a specific user and checks if they fall within their budget.

```
SELECT U.FirstName, U.LastName, Univ.UniversityName,
       Round(Avg(S.HousingCost + S.FoodCost + S.TransportationCost + S.HealthcareCost +
A.RoomAndBoardCost), 0) AS TotalLivingCost,
       U.TuitionFeeBudget + U.AccommodationBudget AS TotalBudget
FROM User U
JOIN User_UnivShortlist US ON U.Id = US.UserId
JOIN University Univ ON US.UniversityName = Univ.UniversityName
JOIN StateWiseExpense S ON Univ.State = S.State
JOIN Accommodation A ON Univ.UniversityName = A.UniversityName
WHERE U.Id = 42
      AND (S.HousingCost + S.FoodCost + S.TransportationCost + S.HealthcareCost +
A.RoomAndBoardCost) <= (U.TuitionFeeBudget + U.AccommodationBudget)
GROUP BY U.Id, U.FirstName, U.LastName, Univ.UniversityName, S.HousingCost,
S.FoodCost, S.TransportationCost, S.HealthcareCost, A.RoomAndBoardCost;
```

```
mysql> SELECT U.FirstName, U.LastName, Univ.UniversityName,
->       Round(Avg(S.HousingCost + S.FoodCost + S.TransportationCost + S.HealthcareCost + A.RoomAndBoardCost), 0) AS TotalLivingCost,
->       U.TuitionFeeBudget + U.AccommodationBudget AS TotalBudget
->   FROM User U
->   JOIN User_UnivShortlist US ON U.Id = US.UserId
->   JOIN University Univ ON US.UniversityName = Univ.UniversityName
->   JOIN StateWiseExpense S ON Univ.State = S.State
->   JOIN Accommodation A ON Univ.UniversityName = A.UniversityName
->   WHERE U.Id = 42
->   AND (S.HousingCost + S.FoodCost + S.TransportationCost + S.HealthcareCost + A.RoomAndBoardCost) <= (U.TuitionFeeBudget + U.AccommodationBudget)
->   GROUP BY U.Id, U.FirstName, U.LastName, Univ.UniversityName, S.HousingCost, S.FoodCost, S.TransportationCost, S.HealthcareCost, A.RoomAndBoardCost;
+-----+-----+-----+-----+-----+
| FirstName | LastName | UniversityName | TotalLivingCost | TotalBudget |
+-----+-----+-----+-----+-----+
| David     | Lee      | City University of New York: Lehman College |    40880 |    49759 |
| David     | Lee      | Del Mar College |    26256 |    49759 |
| David     | Lee      | Dutchess Community College |    39778 |    49759 |
| David     | Lee      | East-West University |    25752 |    49759 |
| David     | Lee      | Lehigh Carbon Community College |    28731 |    49759 |
| David     | Lee      | Memphis College of Art |    34314 |    49759 |
| David     | Lee      | Northwest State Community College |    24590 |    49759 |
| David     | Lee      | Oklahoma Panhandle State University |    31574 |    49759 |
| David     | Lee      | UW Colleges Rock County |    25471 |    49759 |
| David     | Lee      | Virginia Marti College of Art and Design |    24590 |    49759 |
+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

Explanation for 10 rows - Each user is corresponded ( allowed) to shortlist only 10 universities, hence it only displays 10 rows.

## 2.3 Query 3

The query retrieves the first and last names of students, along with their shortlisted universities, in-state tuition fees, and room and board costs. It filters the results to include only those students whose budgets for tuition and accommodation meet or exceed the respective costs associated with their selected universities.

```
SELECT
  u.FirstName,
```

```

u.LastName,
tf.UniversityName,
tf.InStateTuitionFees,
(
    SELECT a.RoomAndBoardCost
    FROM Accommodation a
    WHERE a.UniversityName = tf.UniversityName
) AS RoomAndBoardCost
FROM
User u
JOIN
User_UnivShortlist uu ON u.Id = uu.UserId
JOIN
TuitionFees tf ON uu.UniversityName = tf.UniversityName
WHERE
u.TuitionFeeBudget >= tf.InStateTuitionFees
AND u.AccommodationBudget >= (
    SELECT a.RoomAndBoardCost
    FROM Accommodation a
    WHERE a.UniversityName = tf.UniversityName
);

```

```

mysql> SELECT      u.FirstName,      u.LastName,      tf.UniversityName,      tf.InStateTuitionFees,      (
      SELECT a.RoomAndBoardCost
      FROM Accommodation a
      WHERE
         TuitionFees tf ON uu.UniversityName = tf
      AND u.AccommodationBudget >= (
          SELECT a.RoomAndBoardCost
          FROM Accommodation a
          WHERE a.UniversityName = tf.UniversityName
      )
      ) AS RoomAndBoardCost
      FROM User u JOIN User_UnivShortlist uu ON u.Id = uu.UserId JOIN
      TuitionFees tf ON uu.UniversityName = tf
      AND u.AccommodationBudget >= (
          SELECT a.RoomAndBoardCost
          FROM Accommodation a
          WHERE a.UniversityName = tf.UniversityName
      )
      WHERE
         u.TuitionFeeBudget >= tf.InStateTuitionFees
         AND u.AccommodationBudget >= (
             SELECT a.RoomAndBoardCost
             FROM Accommodation a
             WHERE a.UniversityName = tf.UniversityName
         )
      )Limit 15;
+-----+-----+-----+-----+
| FirstName | LastName | UniversityName | InStateTuitionFees | RoomAndBoardCost |
+-----+-----+-----+-----+
| William3  | Jones   | Anilin Nakoda College | 2380 | 0 |
| William14 | Anderson | Anilin Nakoda College | 2380 | 0 |
| Alex     | Moore   | Adrian College | 37097 | 11318 |
| Jane     | Jones   | Adventist University of Health Sciences | 15150 | 4200 |
| Joseph138 | Hernandez | Adventist University of Health Sciences | 15150 | 4200 |
| Patricia | Johnson | Aims Community College | 2281 | 0 |
| Robert116 | Thomas   | Alabama Southern Community College | 4440 | 0 |
| Alex144  | Smith   | Alabama Southern Community College | 4440 | 0 |
| Laura    | Martin   | Alabama State University | 11068 | 5422 |
| Michael  | Anderson | Alabama State University | 11068 | 5422 |
| Lee      | Taylor   | Albany Community College | 2310 | 0 |
| Patricia92 | Brown   | Albany Community College | 2310 | 0 |
| Michael  | Anderson | Albany Technical College | 3246 | 0 |
| Chris139 | Martin   | Albany Technical College | 3246 | 0 |
| William  | Williams | Albertus Magnus College | 32060 | 13200 |
+-----+-----+-----+-----+
15 rows in set (0.01 sec)

```

## 2.4 Query 4

The query retrieves data on university enrollments for a specified race category, calculating the total enrollment for that category and the overall enrollment for each university. It computes the percentage of enrollment for the race category and orders the results to display universities with the highest representation first.

```

SELECT
d.UniversityName,
d.RaceCategory,
SUM(d.RaceWiseEnrollment) AS RaceWiseTotal,
(SELECT SUM(d2.RaceWiseEnrollment)
FROM Diversity d2
WHERE d2.UniversityName = d.UniversityName) AS TotalEnrollment,
(SUM(d.RaceWiseEnrollment) /
(SELECT SUM(d2.RaceWiseEnrollment)

```

```

FROM Diversity d2
WHERE d2.UniversityName = d.UniversityName)) * 100 AS PercentageOfEnrollment
FROM
Diversity d
WHERE
d.RaceCategory = 'Asian' -- Replace with the user's desired race category
GROUP BY
d.UniversityName,
d.RaceCategory
ORDER BY
PercentageOfEnrollment DESC;

```

```

mysql> SELECT d.UniversityName, d.RaceCategory, SUM(d.RaceWiseEnrollment) AS RaceWiseTotal, (SELECT SUM(d2.RaceWiseEnrollment) FROM Diversity d2 WHERE d2.UniversityName = d.UniversityName) AS TotalEnrollment, (SUM(d.RaceWiseEnrollment) / (SELECT SUM(d2.RaceWiseEnrollment) FROM Diversity d2 WHERE d2.UniversityName = d.UniversityName)) * 100 AS PercentageOfEnrollment FROM Diversity d WHERE d.RaceCategory = 'Asian' GROUP BY d.UniversityName, d.RaceCategory ORDER BY PercentageOfEnrollment DESC Limit 15;
+-----+-----+-----+-----+-----+
| UniversityName | RaceCategory | RaceWiseTotal | TotalEnrollment | PercentageOfEnrollment |
+-----+-----+-----+-----+-----+
| World Mission University | Asian | 172 | 626 | 27.4760 |
| Bethesda University of California | Asian | 191 | 749 | 25.3007 |
| Mission College | Asian | 3723 | 19713 | 18.9367 |
| Long Island Business Institute | Asian | 172 | 941 | 18.2784 |
| Art Center College of Design | Asian | 666 | 4045 | 16.4648 |
| Skyline College | Asian | 3693 | 22564 | 16.3668 |
| Evergreen Valley College | Asian | 3522 | 21807 | 16.1508 |
| University of the Pacific | Asian | 2101 | 13248 | 15.8590 |
| De Anza College | Asian | 7956 | 50305 | 15.8155 |
| Divine Word College | Asian | 27 | 14 | 15.5714 |
| California Institute of Technology | Asian | 550 | 3741 | 15.3059 |
| Ohlone College | Asian | 3554 | 23480 | 15.1789 |
| City College of San Francisco | Asian | 7972 | 52950 | 15.0557 |
| San Jose State University | Asian | 10381 | 69613 | 14.9124 |
| Lawrence Technological University | Asian | 925 | 6310 | 14.6593 |
+-----+-----+-----+-----+-----+
15 rows in set (0.10 sec)

```

## 2.5 Query 5

The query calculates and ranks the top 10 universities by enrollment percentage for a specified race category, providing valuable insights into university diversity. It serves as a resource for students seeking institutions with significant representation of their racial group and aids administrators in assessing and improving diversity efforts.

```

SELECT
  UniversityName,
  RaceCategory,
  TotalEnrollment,
  RaceWiseEnrollment,
  EnrollmentPercentage
FROM (
  SELECT
    D.UniversityName,
    D.RaceCategory,
    SUM(D.TotalEnrollment) AS TotalEnrollment,
    SUM(D.RaceWiseEnrollment) AS RaceWiseEnrollment,
    (SUM(D.RaceWiseEnrollment) / SUM(D.TotalEnrollment) * 100) AS
    EnrollmentPercentage
  FROM
    Diversity D
  WHERE
    D.RaceWiseEnrollment > 0
  GROUP BY

```

```

        D.UniversityName, D.RaceCategory
    ) AS RankedDiversity
WHERE
    EnrollmentPercentage IN (
        SELECT
            EnrollmentPercentage
        FROM (
            SELECT
                EnrollmentPercentage,
                ROW_NUMBER() OVER (PARTITION BY RaceCategory ORDER BY
                    EnrollmentPercentage DESC) AS rn
            FROM (
                SELECT
                    D.UniversityName,
                    D.RaceCategory,
                    SUM(D.TotalEnrollment) AS TotalEnrollment,
                    SUM(D.RaceWiseEnrollment) AS RaceWiseEnrollment,
                    (SUM(D.RaceWiseEnrollment) / SUM(D.TotalEnrollment) * 100) AS
                    EnrollmentPercentage
                FROM
                    Diversity D
                WHERE
                    D.RaceWiseEnrollment > 0
                GROUP BY
                    D.UniversityName, D.RaceCategory
            ) AS InnerQuery
        ) AS Ranked
        WHERE rn <= 10
    )
ORDER BY
    RaceCategory, EnrollmentPercentage DESC;

```

```

mysql> SELECT      UniversityName,      RaceCategory,      TotalEnrollment,      RaceWiseEnrollment,      EnrollmentPercentage
   FROM (      SELECT      D.UniversityName,      D.RaceCategory,
   category,      SUM(D.TotalEnrollment) AS TotalEnrollment,      SUM(D.RaceWiseEnrollment) AS RaceWiseEnrollment,      (SUM(D.RaceWiseEnrollment) / SUM(D.TotalEnrollment) * 100)
   AS EnrollmentPercentage      FROM      Diversity D      WHERE      D.RaceWiseEnrollment > 0      GROUP BY      D.UniversityName, D.RaceCategory ) AS RankedDiversity
 WHERE
    EnrollmentPercentage IN (      SELECT      EnrollmentPercentage
   FROM (      SELECT
        EnrollmentPercentage,      ROW_NUMBER() OVER
        (PARTITION BY RaceCategory ORDER BY EnrollmentPercentage DESC) AS rn
       FROM (      SELECT
        D.UniversityName,      D.RaceCategory,
        SUM(D.TotalEnrollment) AS TotalEnrollment,      SUM(D.RaceWiseEnrollment) AS RaceWiseEnrollment,
        (SUM(D.RaceWiseEnrollment) / SUM(D.TotalEnrollment) * 100) AS EnrollmentPercentage
       FROM
        Diversity D
       WHERE
        D.RaceWiseEnrollment > 0
       GROUP BY
        D.UniversityName, D.RaceCategory
     ) AS InnerQuery
   ) AS Ranked
   WHERE rn <= 10
 ) ORDER BY
    RaceCategory, EnrollmentPercentage DESC;

```

UniversityName	RaceCategory	TotalEnrollment	RaceWiseEnrollment	EnrollmentPercentage
Haskell Indian Nations University	American Indian / Alaska Native	808	398	100.0000
College of the Muscogee Nation	American Indian / Alaska Native	138	104	73.5671
Navajo Technical University	American Indian / Alaska Native	2075	2031	98.8434
Dine College	American Indian / Alaska Native	1488	1467	98.5887
Little Big Horn College	American Indian / Alaska Native	263	257	97.7186
Leech Lake Tribal College	American Indian / Alaska Native	297	286	96.2963
Nebraska Indian Community College	American Indian / Alaska Native	120	115	95.8333
Oglala Lakota College	American Indian / Alaska Native	1427	1361	95.3749
Blackfeet Community College	American Indian / Alaska Native	495	470	94.9495
Chief Dull Knife College	American Indian / Alaska Native	192	179	93.2292
Wichita State University	Asian	232	191	82.1739
Bethesda University of California	Asian	331	191	57.7039
Mission College	Asian	8435	3733	44.2561
Long Island Business Institute	Asian	397	172	43.3249
Evergreen Valley College	Asian	9133	3522	38.5635

15 rows in set (0.19 sec)

### 3. Indexing Analysis

#### 3.1 Query 1

Explain & Analyze Pre-Indexing -

```
mysql> explain analyze SELECT U.FirstName, U.LastName, U.TuitionFeeBudget, (SELECT AVG(InStateTuitionFees) FROM TuitionFees) as avg_state_tuition
-> FROM User U
-> JOIN User_UniversityShortlist US ON U.Id = US.UserId
-> JOIN TuitionFees T ON US.UniversityName = T.UniversityName
-> WHERE U.TuitionFeeBudget < (
->     SELECT AVG(InStateTuitionFees)
->     FROM TuitionFees
-> );
+-----+
| EXPLAIN
+-----+
-----+
|   |
-----+
-----+
-----+
| -> Nested loop inner join (cost=783.98 rows=460) (actual time=2.873..6.265 rows=240 loops=1)
|   -> Nested loop inner join (cost=623.00 rows=460) (actual time=2.838..5.518 rows=240 loops=1)
|     -> Covering index scan on US using idx_user_univshortlist_uni_name (cost=140.00 rows=1380) (actual time=0.073..0.464 rows=1380 loops=1)
|       -> Filter: (U.TuitionFeeBudget < (select #3)) (cost=0.25 rows=0.3) (actual time=0.003..0.003 rows=0 loops=1380)
|         -> Single-row index lookup on U using PRIMARY (Id=US.UserId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1380)
|           -> Select #3 (subquery in condition; run only once)
|             -> Aggregate: avg(TuitionFees.InStateTuitionFees) (cost=591.35 rows=1) (actual time=1.616..1.617 rows=1 loops=1)
|               -> Table scan on TuitionFees (cost=297.55 rows=2938) (actual time=0.424..1.319 rows=2938 loops=1)
|     -> Single-row covering index lookup on T using PRIMARY (UniversityName=US.UniversityName) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=240)
|-> Select #2 (subquery in projection; run only once)
|   -> Aggregate: avg(TuitionFees.InStateTuitionFees) (cost=591.35 rows=1) (actual time=1.620..1.621 rows=1 loops=1)
|     -> Table scan on TuitionFees (cost=297.55 rows=2938) (actual time=0.125..1.173 rows=2938 loops=1)
|
```

##### 3.1.1 Index 1

Query - CREATE INDEX idx\_tuitionfees\_instate\_tuition ON TuitionFees  
(InStateTuitionFees);

Purpose - The index on InStateTuitionFees aims to optimize retrieval and comparison operations for queries filtering or sorting based on this column, especially when calculating average tuition fees in subqueries.

```

mysql> CREATE INDEX idx_tuitionfees_instate_tuition ON TuitionFees(InStateTuitionFees);
Query OK, 0 rows affected (0.37 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> explain analyze SELECT U.FirstName, U.LastName, U.TuitionFeeBudget, (SELECT AVG(InStateTuitionFees) FROM TuitionFees) as avg_state_tuition FROM User U JOIN User_UniversityShortlist US ON U.Id = US.UserId JOIN TuitionFees T ON US.UniversityName = T.UniversityName WHERE U.TuitionFeeBudget < (      SELECT AVG(InStateTuitionFees)      FROM TuitionFees );
+-----+
| EXPLAIN
+-----+
|> Nested loop inner join (cost=783.99 rows=460) (actual time=1.12..3.991 rows=240 loops=1)
|  -> Nested loop inner join (cost=623.00 rows=460) (actual time=1.104..3.447 rows=240 loops=1)
|    -> Covering index scan on US using idx_user_universityshortlist_uni_name (cost=140.00 rows=1380) (actual time=0.064..0.442 rows=1380 loops=1)
|      -> Filter: (U.TuitionFeeBudget < (select #3)) (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=1380)
|        -> Single-row index lookup on U using PRIMARY (id=US.UserId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1380)
|          -> Select #3 (subquery in condition; run only once)
|            -> Aggregate: avg(TuitionFees.InStateTuitionFees) (cost=591.35 rows=1) (actual time=0.987..0.987 rows=1 loops=1)
|              -> Covering index scan on TuitionFees using idx_tuitionfees_instate_tuition (cost=297.55 rows=2938) (actual time=0.039..0.715 rows=2938 loops=1)
-> Single-row covering index lookup on T using PRIMARY (UniversityName=US.UniversityName) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=240)
-> Select #2 (subquery in projection; run only once)
|  -> Aggregate: avg(TuitionFees.InStateTuitionFees) (cost=591.35 rows=1) (actual time=0.967..0.967 rows=1 loops=1)
|    -> Covering index scan on TuitionFees using idx_tuitionfees_instate_tuition (cost=297.55 rows=2938) (actual time=0.132..0.687 rows=2938 loops=1)
|>

```

### 3.1.2 Index 2

Query - CREATE INDEX idx\_user\_tuitionfeebudget ON User(TuitionFeeBudget);

Purpose - This index aims to enhance performance for queries involving the TuitionFeeBudget column in the User table. As this column is used in the WHERE clause to filter users based on their budget, indexing it can improve the speed of this operation.

```

mysql> CREATE INDEX idx_user_tuitionfeebudget ON User(TuitionFeeBudget);
Query OK, 0 rows affected (0.07 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> explain analyze SELECT U.FirstName, U.LastName, U.TuitionFeeBudget, (SELECT AVG(InStateTuitionFees) FROM TuitionFees) as avg_state_tuition FROM User U JOIN User_UniversityShortlist US ON U.Id = US.UserId JOIN TuitionFees T ON US.UniversityName = T.UniversityName WHERE U.TuitionFeeBudget < (      SELECT AVG(InStateTuitionFees)      FROM TuitionFees );
+-----+
| EXPLAIN
+-----+
|> Nested loop inner join (cost=707.81 rows=242) (actual time=0.110..2.818 rows=240 loops=1)
|  -> Nested loop inner join (cost=623.00 rows=242) (actual time=0.088..2.353 rows=240 loops=1)
|    -> Covering index scan on US using idx_user_universityshortlist_uni_name (cost=140.00 rows=1380) (actual time=0.065..0.423 rows=1380 loops=1)
|      -> Filter: (U.TuitionFeeBudget < (select #3)) (cost=0.25 rows=0.2) (actual time=0.001..0.001 rows=0 loops=1380)
|        -> Single-row index lookup on U using PRIMARY (id=US.UserId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1380)
|          -> Select #3 (subquery in condition; run only once)
|            -> Aggregate: avg(TuitionFees.InStateTuitionFees) (cost=591.35 rows=1) (actual time=1.077..1.078 rows=1 loops=1)
|              -> Covering index scan on TuitionFees (cost=297.55 rows=2938) (actual time=0.044..0.791 rows=2938 loops=1)
|                -> Single-row covering index lookup on T using PRIMARY (UniversityName=US.UniversityName) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=240)
-> Select #2 (subquery in projection; run only once)
|  -> Aggregate: avg(TuitionFees.InStateTuitionFees) (cost=591.35 rows=1) (actual time=1.042..1.042 rows=1 loops=1)
|    -> Table scan on TuitionFees (cost=297.55 rows=2938) (actual time=0.069..0.745 rows=2938 loops=1)
|>

```

### 3.1.3 Index 3

CREATE INDEX idx\_user\_fullname ON User(FirstName, LastName);

Purpose - The purpose of the idx\_user\_fullname index on (FirstName, LastName) is to speed up retrieval of rows based on these columns, optimizing queries that filter or sort by FirstName and LastName.

```
mysql> explain analyze SELECT U.FirstName, U.LastName, U.TuitionFeeBudget, (SELECT AVG(InStateTuitionFees) FROM TuitionFees) as avg_state_tuition
-- FROM User U
-- JOIN User.UnivShortlist US ON US.UserId = US.UserId
-- JOIN TuitionFees T ON US.UniversityName = T.UniversityName
-- WHERE T.TuitionFeeBudget < (
--     SELECT AVG(InStateTuitionFees)
--     FROM TuitionFees
-- );
+-----+
| EXPLAIN
+-----+
+-----+
| > Nested loop inner join (cost=788.27 rows=460) (actual time=1.181..5.096 rows=240 loops=1)
  > Nested loop inner join (cost=627.29 rows=460) (actual time=1.160..4.124 rows=240 loops=1)
    > Covering index scan on US using idx_user_univshortlist_uni_name (cost=144.29 rows=1380) (actual time=0.055..0.631 rows=1380 loops=1)
    > Filter: (U.TuitionFeeBudget < (select #3)) (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=1380)
      > Single-row index lookup on PRIMARY (cost=0.00..0.00 rows=1) (actual time=0.001..0.001 rows=1 loops=1380)
      > Select #3 (subquery; run only once)
        > Aggregate: avg(TuitionFees.InStateTuitionFees) (cost=591.35 rows=1) (actual time=1.049..1.049 rows=1 loops=1)
          > Table scan on TuitionFees (cost=297.55 rows=2938) (actual time=0.057..0.759 rows=2938 loops=1)
    > Single-row covering index lookup on T using PRIMARY (UniversityName=US.UniversityName) (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=240)
  > Select #2 (subquery in projection; run only once)
    > Aggregate: avg(TuitionFees.InStateTuitionFees) (cost=591.35 rows=1) (actual time=1.445..1.446 rows=1 loops=1)
      > Table scan on TuitionFees (cost=297.55 rows=2938) (actual time=0.042..0.982 rows=2938 loops=1)
+-----+
```

### 3.1.4 Index Analysis

Index Configuration	Query Cost	Explanation
Original Query	783.98	Working without any index
With idx_user_tuitionfeebudget	707.81	the index allows for faster lookups rather than performing a full table scan.
With idx_tuitionfees_instate_tuition	783.98	may be because the average tuition fee is computed using a subquery that scans the entire TuitionFees table each time it is executed.
With idx_user_fullname	788.27	The slight increase in cost after indexing is likely due to the optimizer using additional index lookups and

		nested loop joins, which slightly raised the cost for retrieving specific rows compared to the simpler initial plan without indexing.
--	--	---

### 3.1.5 Report Final Design

**Selected Index:** idx\_user\_tuitionfeebudget

**Column:** User.TuitionFeeBudget

#### Rationale

The index was chosen to optimize queries filtering users based on their tuition fee budgets, particularly to identify those whose budgets are below the average in-state tuition fees.

#### Analysis

1. **Before Indexing:** The query cost was high (783.98) due to full table scans on the User table, leading to slow performance.
2. **After Indexing:** Implementing the index reduced the query cost to 707.81, significantly improving performance by allowing faster lookups on the TuitionFeeBudget.

## 3.2 Query 2

### 3.2.1 Index 1

Query - CREATE INDEX idx\_statewise\_housing\_cost ON StateWiseExpense(HousingCost);

Purpose: This index supports quick access to HousingCost in StateWiseExpense, essential in calculating average living costs. Since the WHERE clause checks the sum of various living costs (including HousingCost), this index reduces full scans on StateWiseExpense by quickly retrieving HousingCost values.

Cost Impact: By eliminating the need to scan the entire StateWiseExpense table for HousingCost, the cost is brought down significantly from full scan levels.

```

mysql> explain analyze SELECT U.FirstName, U.LastName, Univ.UniversityName,
   >     Round(Avg(S.HousingCost + S.FoodCost + S.TransportationCost + S.HealthcareCost + A.RoomAndBoardCost), 0) AS TotalLivingCost,
   >     Round(U.TuitionFeeBudget + U.AccommodationBudget AS TotalBudget
--> FROM User U
--> JOIN User.UnivShortlist US ON U.Id = US.UserId
--> JOIN University Univ ON US.UniversityName = Univ.UniversityName
--> JOIN StateWiseExpense S ON Univ.State = S.State
--> JOIN Accommodation A ON Univ.UniversityName = A.UniversityName
--> WHERE U.Id = 42
--> AND (S.HousingCost + S.FoodCost + S.TransportationCost + S.HealthcareCost + A.RoomAndBoardCost) <= (U.TuitionFeeBudget + U.AccommodationBudget)
--> GROUP BY U.Id, U.FirstName, U.LastName, Univ.UniversityName, S.HousingCost, S.FoodCost, S.TransportationCost, S.HealthcareCost, A.RoomAndBoardCost;
+-----+
| EXPLAIN
+-----+
|> Table scan on <temporary> (actual time=0.681..0.688 rows=10 loops=1)
  -> Aggregate using temporary table (actual time=0.671..0.671 rows=10 loops=1)
    -> Nested loop inner join (cost=8.81 rows=10) (actual time=0.671..0.671 rows=10 loops=1)
      -> Nested loop inner join (cost=5.31 rows=10) (actual time=0.199..0.25 rows=10 loops=1)
        -> Covering index lookup on US using PRIMARY (UserId=42) (cost=1.81 rows=10) (actual time=0.0318..0.0545 rows=10 loops=1)
          -> Filter: Univ.State is not null (cost=0.26 rows=1) (actual time=0.0178..0.0189 rows=1 loops=1)
            -> Single-row index lookup on Univ using PRIMARY (UniversityName=Univ.UniversityName) (cost=0.26 rows=1) (actual time=0.0165..0.0166 rows=1 loops=10)
              -> Single-row index lookup on S using PRIMARY (State=Univ.State) (cost=0.26 rows=1) (actual time=0.0077..0.0079 rows=1 loops=10)
                -> Filter: (((($1.HousingCost + $1.FoodCost) + $1.TransportationCost) + $1.HealthcareCost) + A.RoomAndBoardCost) <= <cache>('38917' + '18842')) (cost=0.26 rows=1) (actual time=0.0148..0.0152 rows=1 loops=10)
s=10)
  -> Single-row index lookup on A using PRIMARY (UniversityName=US.UniversityName) (cost=0.26 rows=1) (actual time=0.013..0.0131 rows=1 loops=10)
+-----+
1 row in set (0.01 sec)

mysql> ■

```

### 3.2.2 Index 2

Query - CREATE INDEX idx\_statewise\_healthcare\_cost ON StateWiseExpense(HealthcareCost);

Purpose: Similar to idx\_statewise\_housing\_cost, this index enables faster retrieval of HealthcareCost values from StateWiseExpense for living cost calculations.

Cost Impact: This index similarly brings down the cost by removing the need for a full table scan when accessing HealthcareCost, facilitating quicker computations in the aggregation.

```

mysql> CREATE INDEX idx_statewise_healthcare_cost ON StateWiseExpense(HealthcareCost);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze SELECT U.FirstName, U.LastName, Univ.UniversityName,
   >     Round(Avg(S.HousingCost + S.FoodCost + S.TransportationCost + S.HealthcareCost + A.RoomAndBoardCost), 0) AS TotalLivingCost,
   >     Round(U.TuitionFeeBudget + U.AccommodationBudget AS TotalBudget
--> FROM User U
--> JOIN User.UnivShortlist US ON U.Id = US.UserId
--> JOIN University Univ ON US.UniversityName = Univ.UniversityName
--> JOIN StateWiseExpense S ON Univ.State = S.State
--> JOIN Accommodation A ON Univ.UniversityName = A.UniversityName
--> WHERE U.Id = 42
--> AND (S.HousingCost + S.FoodCost + S.TransportationCost + S.HealthcareCost + A.RoomAndBoardCost) <= (U.TuitionFeeBudget + U.AccommodationBudget)
--> GROUP BY U.Id, U.FirstName, U.LastName, Univ.UniversityName, S.HousingCost, S.FoodCost, S.TransportationCost, S.HealthcareCost, A.RoomAndBoardCost;
+-----+
| EXPLAIN
+-----+
|> Table scan on <temporary> (actual time=0.746..0.752 rows=10 loops=1)
  -> Aggregate using temporary table (actual time=0.734..0.734 rows=10 loops=1)
    -> Nested loop inner join (cost=12.3 rows=10) (actual time=0.19..0.577 rows=10 loops=1)
      -> Nested loop inner join (cost=8.1 rows=10) (actual time=0.143..0.466 rows=10 loops=1)
        -> Nested loop inner join (cost=5.31 rows=10) (actual time=0.111..0.288 rows=10 loops=1)
          -> Covering index lookup on US using PRIMARY (UserId=42) (cost=1.81 rows=10) (actual time=0.0256..0.0493 rows=10 loops=1)
            -> Filter: (Univ.State is not null) (cost=0.26 rows=1) (actual time=0.022..0.0224 rows=1 loops=10)
              -> Single-row index lookup on Univ using PRIMARY (UniversityName=US.UniversityName) (cost=0.26 rows=1) (actual time=0.0060..0.00706 rows=1 loops=10)
                -> Single-row index lookup on S using PRIMARY (State=Univ.State) (cost=0.26 rows=1) (actual time=0.0060..0.00706 rows=1 loops=10)
                  -> Filter: (((($1.HousingCost + $1.FoodCost) + $1.TransportationCost) + $1.HealthcareCost) + A.RoomAndBoardCost) <= <cache>('38917' + '18842')) (cost=0.26 rows=1) (actual time=0.0199..0.0202 rows=1 loops=10)
s=10)
  -> Single-row index lookup on A using PRIMARY (UniversityName=US.UniversityName) (cost=0.26 rows=1) (actual time=0.0179..0.018 rows=1 loops=10)
+-----+
1 row in set (0.01 sec)

mysql> ■

```

### 3.2.3 Index 3

Query - CREATE INDEX idx\_accommodation\_room\_and\_board\_cost ON Accommodation(RoomAndBoardCost);

Purpose: This index expedites access to RoomAndBoardCost within Accommodation, a key factor in calculating the total living costs for each university. By indexing this column, the query can avoid scanning the entire Accommodation table, improving the efficiency of aggregating living costs.

Cost Impact: Like the previous indexes, this significantly reduces the cost associated with aggregation by targeting RoomAndBoardCost directly without a full table scan.

```
mysql> CREATE INDEX idx_accommodation_room_and_board_cost ON Accommodation(RoomAndBoardCost);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze SELECT U.FirstName, U.LastName, Univ.UniversityName,
   >          Round(Avg(S.HousingCost + S.FoodCost + S.TransportationCost + S.HealthcareCost + A.RoomAndBoardCost), 0) AS TotalLivingCost,
   >          U.TuitionFeeBudget + U.AccommodationBudget AS TotalBudget
   > FROM User U
   > JOIN UniversityChartList US ON U.Id = US.UserId
   > JOIN University Univ ON US.UniversityName = Univ.UniversityName
   > JOIN StateWiseExpense S ON Univ.State = S.State
   > JOIN Accommodation A ON Univ.UniversityName = A.UniversityName
   > WHERE U.Id = 42
   > AND (S.HousingCost + S.FoodCost + S.TransportationCost + S.HealthcareCost + A.RoomAndBoardCost) <= (U.TuitionFeeBudget + U.AccommodationBudget)
   > GROUP BY U.Id, U.FirstName, U.LastName, Univ.UniversityName, S.HousingCost, S.FoodCost, S.TransportationCost, S.HealthcareCost, A.RoomAndBoardCost;
+-----+
| EXPLAIN
+-----+
|> Table scan on <temporary>  (actual time=1.38..1.39 rows=10 loops=1)
|  --> Aggregate using temporary table  (actual time=1.37..1.37 rows=10 loops=1)
|     --> Nested loop inner join  (cost=12.3 rows=10) (actual time=0.589..1.12 rows=10 loops=1)
|        --> Nested loop inner join  (cost=1.81 rows=10) (actual time=0.138..0.264 rows=10 loops=1)
|           --> Nested loop inner join  (cost=0.189 rows=10) (actual time=0.018..0.019 rows=10 loops=1)
|              --> Covering index lookup on US using PRIMARY (UserId=42)  (cost=1.91 rows=10) (actual time=0.0258..0.05 rows=10 loops=1)
|                 --> Filter: (Univ.State is not null)  (cost=0.26 rows=1) (actual time=0.0219..0.0222 rows=1 loops=10)
|                    --> Single-row index lookup on Univ using PRIMARY (UniversityName=US.UniversityName)  (cost=0.26 rows=1) (actual time=0.0207..0.0209 rows=1 loops=10)
|                       --> Filter: (((S.HousingCost + S.FoodCost) + S.TransportationCost + S.HealthcareCost) + A.RoomAndBoardCost) <= <cache>('36917' + '18842')  (cost=0.26 rows=1) (actual time=0.0744..0.0748 rows=1 loops=10)
|                         --> Single-row index lookup on A using PRIMARY (UniversityName=US.UniversityName)  (cost=0.26 rows=1) (actual time=0.0724..0.0726 rows=1 loops=10)
|                --> Single-row index lookup on A using PRIMARY (UniversityName=US.UniversityName)  (cost=0.26 rows=1) (actual time=0.0724..0.0726 rows=1 loops=10)
|      --> Table scan on <temporary>  (actual time=1.38..1.39 rows=10 loops=1)
|        --> Aggregate using temporary table  (actual time=1.37..1.37 rows=10 loops=1)
|           --> Nested loop inner join  (cost=12.3 rows=10) (actual time=0.589..1.12 rows=10 loops=1)
|              --> Nested loop inner join  (cost=1.81 rows=10) (actual time=0.138..0.264 rows=10 loops=1)
|                 --> Nested loop inner join  (cost=0.189 rows=10) (actual time=0.018..0.019 rows=10 loops=1)
|                    --> Covering index lookup on US using PRIMARY (UserId=42)  (cost=1.91 rows=10) (actual time=0.0258..0.05 rows=10 loops=1)
|                       --> Filter: (Univ.State is not null)  (cost=0.26 rows=1) (actual time=0.0219..0.0222 rows=1 loops=10)
|                          --> Single-row index lookup on Univ using PRIMARY (UniversityName=US.UniversityName)  (cost=0.26 rows=1) (actual time=0.0207..0.0209 rows=1 loops=10)
|                             --> Filter: (((S.HousingCost + S.FoodCost) + S.TransportationCost + S.HealthcareCost) + A.RoomAndBoardCost) <= <cache>('36917' + '18842')  (cost=0.26 rows=1) (actual time=0.0744..0.0748 rows=1 loops=10)
|                                --> Single-row index lookup on A using PRIMARY (UniversityName=US.UniversityName)  (cost=0.26 rows=1) (actual time=0.0724..0.0726 rows=1 loops=10)
|          --> Table scan on <temporary>  (actual time=1.38..1.39 rows=10 loops=1)
|            --> Aggregate using temporary table  (actual time=1.37..1.37 rows=10 loops=1)
|               --> Nested loop inner join  (cost=12.3 rows=10) (actual time=0.589..1.12 rows=10 loops=1)
|                  --> Nested loop inner join  (cost=1.81 rows=10) (actual time=0.138..0.264 rows=10 loops=1)
|                     --> Nested loop inner join  (cost=0.189 rows=10) (actual time=0.018..0.019 rows=10 loops=1)
|                        --> Covering index lookup on US using PRIMARY (UserId=42)  (cost=1.91 rows=10) (actual time=0.0258..0.05 rows=10 loops=1)
|                           --> Filter: (Univ.State is not null)  (cost=0.26 rows=1) (actual time=0.0219..0.0222 rows=1 loops=10)
|                              --> Single-row index lookup on Univ using PRIMARY (UniversityName=US.UniversityName)  (cost=0.26 rows=1) (actual time=0.0207..0.0209 rows=1 loops=10)
|                                 --> Filter: (((S.HousingCost + S.FoodCost) + S.TransportationCost + S.HealthcareCost) + A.RoomAndBoardCost) <= <cache>('36917' + '18842')  (cost=0.26 rows=1) (actual time=0.0744..0.0748 rows=1 loops=10)
|                                    --> Single-row index lookup on A using PRIMARY (UniversityName=US.UniversityName)  (cost=0.26 rows=1) (actual time=0.0724..0.0726 rows=1 loops=10)
|                                     --> Single-row index lookup on A using PRIMARY (UniversityName=US.UniversityName)  (cost=0.26 rows=1) (actual time=0.0724..0.0726 rows=1 loops=10)
|                                         --> Single-row index lookup on A using PRIMARY (UniversityName=US.UniversityName)  (cost=0.26 rows=1) (actual time=0.0724..0.0726 rows=1 loops=10)

1 row in set (0.01 sec)

mysql> ||
```

### 3.2.4 Index Analysis

Index Configuration	Query Cost	Explanation
Original Query	12.3	Working without any index
With idx_statewise_housing_cost	12.3	Allows for faster aggregation of HousingCost by preventing a full scan on StateWiseExpense , improving the query cost when calculating living costs.

With idx_statewise_healthcare_cost	12.3	Allows for efficient lookups on HealthcareCost within StateWiseExpense , avoiding a full scan for this specific cost in each row.
With idx_accommodation_room_and_board_cost	12.3	Speeds up lookups and aggregations of RoomAndBoardCost within Accommodation, improving performance in calculating living costs for each shortlisted university.

### 3.3 Query 3

Database Content -

Explain & Analyze Pre-Indexing

```

--> FROM
-->   User u
--> JOIN
-->   User_UniversityShortlist uu ON u.Id = uu.UserId
--> JOIN
-->   TuitionFees tf ON uu.UniversityName = tf.UniversityName
--> WHERE
-->   u.TuitionFeeBudget >= tf.InStateTuitionFees
-->   AND u.AccommodationBudget >= (
-->     SELECT a.RoomAndBoardCost
-->     FROM Accommodation a
-->     WHERE a.UniversityName = tf.UniversityName
-->   );
+-----+
| EXPLAIN
+-----+
|                                         |
+-----+
| --> Nested loop inner join (cost=1106.00 rows=460) (actual time=0.287..8.817 rows=960 loops=1)
|   --> Nested loop inner join (cost=623.00 rows=1380) (actual time=0.249..2.737 rows=1380 loops=1)
|     --> Covering index scan on uu using idx_user_univshortlist uni name (cost=140.00 rows=1380) (actual time=0.227..0.710 rows=1380 loops=1)
|     --> Single-row index lookup on u using PRIMARY (Id=u.UserId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1380)
|   --> Filter: ((u.TuitionFeeBudget >= tf.InStateTuitionFees) and (u.AccommodationBudget >= (select #3))) (cost=0.25 rows=0..3) (actual time=0.004..0.004 rows=1 loops=1380)
|     --> Single-row index lookup on tf using PRIMARY (UniversityName=uu.UniversityName) (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=1380)
|       --> Select #3 (subquery in condition; dependent)
|         --> Single-row index lookup on a using PRIMARY (UniversityName=tf.UniversityName) (cost=0.35 rows=1) (actual time=0.003..0.003 rows=1 loops=1107)
|   --> Select #2 (subquery in projection; dependent)
|     --> Single-row index lookup on a using PRIMARY (UniversityName=tf.UniversityName) (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=960)
+-----+

```

### 3.3.1 Index 1

Query - CREATE INDEX idx\_tuitionfees\_instate ON TuitionFees(InStateTuitionFees);

Purpose - The index on InStateTuitionFees aims to optimize retrieval and comparison operations for queries filtering or sorting based on this column, especially when comparing tuition fees against user budgets.

```

mysql> CREATE INDEX idx_tuitionfees_instate ON TuitionFees(InStateTuitionFees);
Query OK, 0 rows affected (0.13 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> explain analyze SELECT      u.FirstName,      u.LastName,      tf.UniversityName,      tf.InStateTuitionFees,      (
      SELECT a.RoomAndBoardCost      FROM Accommodation
      WHERE a.UniversityName = tf.UniversityName ) AS RoomAndBoardCost FROM      User u JOIN      User_UniversityShortlist uu ON u.Id = uu.UserId JOIN      TuitionFees tf ON uu.Uni
versityName = tf.UniversityName WHERE      u.TuitionFeeBudget >= tf.InStateTuitionFees      AND u.AccommodationBudget >= (
      SELECT a.RoomAndBoardCost      FROM Accommodation
      WHERE a.UniversityName = tf.UniversityName );
+-----+
| EXPLAIN
+-----+
|                                         |
+-----+
| --> Nested loop inner join (cost=1106.00 rows=460) (actual time=0.138..8.059 rows=960 loops=1)
|   --> Nested loop inner join (cost=623.00 rows=1380) (actual time=0.097..2.247 rows=1380 loops=1)
|     --> Covering index scan on uu using idx_user_univshortlist uni name (cost=140.00 rows=1380) (actual time=0.075..0.433 rows=1380 loops=1)
|     --> Single-row index lookup on u using PRIMARY (Id=u.UserId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1380)
|   --> Filter: ((u.TuitionFeeBudget >= tf.InStateTuitionFees) and (u.AccommodationBudget >= (select #3))) (cost=0.25 rows=0..3) (actual time=0.004..0.004 rows=1 loops=1380)
|     --> Single-row index lookup on tf using PRIMARY (UniversityName=uu.UniversityName) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1380)
|       --> Select #3 (subquery in condition; dependent)
|         --> Single-row index lookup on a using PRIMARY (UniversityName=tf.UniversityName) (cost=0.35 rows=1) (actual time=0.003..0.003 rows=1 loops=1107)
|   --> Select #2 (subquery in projection; dependent)
|     --> Single-row index lookup on a using PRIMARY (UniversityName=tf.UniversityName) (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=960)
+-----+

```

### 3.3.2 Index 2

Query - CREATE INDEX idx\_user\_tuitionfeebudget ON User(TuitionFeeBudget);

Purpose - This index is designed to improve performance for queries that filter users based on their tuition fee budgets, as this column is used in the WHERE clause to determine if a user can afford the tuition fees.

```

mysql> CREATE INDEX idx_user_tuitionfeebudget ON User(TuitionFeeBudget);
Query OK, 0 rows affected (0.20 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> explain analyze SELECT u.FirstName, u.LastName, tf.UniversityName, tf.InStateTuitionFees, (
    SELECT a.RoomAndBoardCost
    FROM Accommodation
    WHERE a.UniversityName = tf.UniversityName ) AS RoomAndBoardCost FROM User u JOIN User_UniversityShortlist uu ON u.Id = uu.UserId JOIN TuitionFees tf ON uu.UniversityName = tf.UniversityName WHERE u.TuitionFeeBudget >= tf.InStateTuitionFees AND u.AccommodationBudget >= (
    SELECT a.RoomAndBoardCost
    FROM Accommodation
    WHERE a.UniversityName = tf.UniversityName );
+-----+
| EXPLAIN
+-----+
|> Nested loop inner join (cost=1106.00 rows=460) (actual time=0.100..8.568 rows=960 loops=1)
|   -> Nested loop inner join (cost=623.00 rows=1380) (actual time=0.073..2.469 rows=1380 loops=1)
|     -> Covering index scan on uu using idx_user_univshortlist_uni_name (cost=140.00 rows=1380) (actual time=0.060..0.535 rows=1380 loops=1)
|     -> Filter: ((u.TuitionFeeBudget >= tf.InStateTuitionFees) and (u.AccommodationBudget >= (select #3))) (cost=0.25 rows=0.3) (actual time=0.004..0.004 rows=1 loops=1380)
|     -> Single-row index lookup on tf using PRIMARY (UniversityName=uu.UniversityName) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1380)
|       -> Select #3 (subquery in condition; dependent)
|         -> Single-row index lookup on a using PRIMARY (UniversityName=tf.UniversityName) (cost=0.35 rows=1) (actual time=0.003..0.003 rows=1 loops=1380)
-> Select #2 (subquery in projection; dependent)
-> Single-row index lookup on a using PRIMARY (UniversityName=tf.UniversityName) (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=960)
|

```

### 3.3.3 Index 3

Query - CREATE INDEX idx\_user\_accommodationbudget ON User(AccommodationBudget);

Purpose - The index on AccommodationBudget helps optimize queries that filter users based on their accommodation budget, further enhancing the performance of operations that require budget checks.

```

mysql> CREATE INDEX idx_user_accommodationbudget ON User(AccommodationBudget);
Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> explain analyze SELECT u.FirstName, u.LastName, tf.UniversityName, tf.InStateTuitionFees, (
    SELECT a.RoomAndBoardCost
    FROM Accommodation
    WHERE a.UniversityName = tf.UniversityName ) AS RoomAndBoardCost FROM User u JOIN User_UniversityShortlist uu ON u.Id = uu.UserId JOIN TuitionFees tf ON uu.UniversityName = tf.UniversityName WHERE u.TuitionFeeBudget >= tf.InStateTuitionFees AND u.AccommodationBudget >= (
    SELECT a.RoomAndBoardCost
    FROM Accommodation
    WHERE a.UniversityName = tf.UniversityName );
+-----+
| EXPLAIN
+-----+
|> Nested loop inner join (cost=1106.00 rows=460) (actual time=0.102..9.142 rows=960 loops=1)
|   -> Nested loop inner join (cost=623.00 rows=1380) (actual time=0.074..2.610 rows=1380 loops=1)
|     -> Covering index scan on uu using idx_user_univshortlist_uni_name (cost=140.00 rows=1380) (actual time=0.062..0.538 rows=1380 loops=1)
|     -> Single-row index lookup on tf using PRIMARY (Id=u.UserId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1380)
|       -> Filter: ((u.TuitionFeeBudget >= tf.InStateTuitionFees) and (u.AccommodationBudget >= (select #3))) (cost=0.25 rows=0.3) (actual time=0.004..0.005 rows=1 loops=1380)
|       -> Single-row index lookup on tf using PRIMARY (UniversityName=uu.UniversityName) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1380)
|         -> Select #3 (subquery in condition; dependent)
|           -> Single-row index lookup on a using PRIMARY (UniversityName=tf.UniversityName) (cost=0.35 rows=1) (actual time=0.003..0.003 rows=1 loops=1380)
-> Select #2 (subquery in projection; dependent)
-> Single-row index lookup on a using PRIMARY (UniversityName=tf.UniversityName) (cost=0.35 rows=1) (actual time=0.003..0.003 rows=1 loops=960)
|

```

### 3.3.4 Index Analysis

Index Configuration	Query Cost	Explanation
Original Query	1106.00	The query executed without any index, leading to full table scans for user and tuition fee data, resulting in high cost.

With idx_user_tuitionfeebudget	1106.00	No performance improvement; the cost remains the same due to the multiple nested subqueries that may still require full scans.
With idx_tuitionfees_instate_tuition	1106.00	No performance gain observed; the structure of the query still necessitates comprehensive scans due to its design.
With idx_user_accommodationbudget	1106.00	Similar to previous indices, no change in performance; the nested subqueries may still cause extensive scanning of tables.

### 3.3.5 Report Final Design

**Selected Index:** idx\_user\_tuitionfeebudget

**Column:** User.TuitionFeeBudget

#### Rationale

The index was selected to optimize the query for filtering users based on their tuition fee budgets, especially for determining affordability concerning tuition and accommodation costs.

#### Analysis

- **Before Indexing:** The original query cost was 1106.00, attributed to full table scans on both the User and TuitionFees tables, leading to sluggish performance.
- **After Indexing:** Despite implementing indices, the query cost remained at 1106.00, indicating that the nested subqueries may still force full scans, limiting the effectiveness of the indices.

#### Summary

In this case, the complexity of the nested subqueries and the overall query structure impacted the expected performance improvements from indexing. Further optimization may be necessary to restructure the query or explore alternative indexing strategies for better results.

## 3.4 Query 4

Explain & Analyze Pre-Indexing -

```

--> GROUP BY
-->     d.UniversityName,
-->     d.RaceCategory
--> ORDER BY
-->     PercentageOfEnrollment DESC;
+-----+
| EXPLAIN
+-----+
|-----+
|  --> Sort: PercentageOfEnrollment DESC (actual time=84.704..85.249 rows=2160 loops=1)
|    --> Stream results (cost=1829.06 rows=1597) (actual time=0.319..79.284 rows=2160 loops=1)
|      --> Group aggregate: sum(d.RaceWiseEnrollment), sum(d.RaceWiseEnrollment) (cost=1829.06 rows=1597) (actual time=0.186..13.924 rows=2160 loops=1)
|        --> Filter: (d.RaceCategory = 'Asian') (cost=1669.35 rows=1597) (actual time=0.144..12.412 rows=2160 loops=1)
|          --> Index scan on d using PRIMARY (UniversityName=d.UniversityName) (cost=1669.35 rows=15971) (actual time=0.134..9.302 rows=23761 loops=1)
|-----+
|  --> Select #3 (subquery in projection; dependent)
|    --> Aggregates: sum(d2.RaceWiseEnrollment) (cost=1..48 rows=1) (actual time=0.014..0.014 rows=1 loops=2160)
|      --> Index lookup on d2 using PRIMARY (UniversityName=d.UniversityName) (cost=0.86 rows=6) (actual time=0.008..0.013 rows=11 loops=2160)
|-----+
|  --> Select #3 (subquery in projection; dependent)
|    --> Aggregates: sum(d2.RaceWiseEnrollment) (cost=1..48 rows=1) (actual time=0.014..0.014 rows=1 loops=2160)
|      --> Index lookup on d2 using PRIMARY (UniversityName=d.UniversityName) (cost=0.86 rows=6) (actual time=0.008..0.013 rows=11 loops=2160)
|-----+
|  --> Select #2 (subquery in projection; dependent)
|    --> Aggregates: sum(d2.RaceWiseEnrollment) (cost=1..48 rows=1) (actual time=0.013..0.013 rows=1 loops=2160)
|      --> Index lookup on d2 using PRIMARY (UniversityName=d.UniversityName) (cost=0.86 rows=6) (actual time=0.007..0.012 rows=11 loops=2160)
|-----+

```

### 3.4.1 Index 1

Query - CREATE INDEX idx\_diversity\_racecategory ON Diversity(RaceCategory);

Purpose - The index on RaceCategory is aimed at reducing the cost of query performance by enabling more efficient lookups and filtering of records based on specified race categories.

```

mysql> CREATE INDEX idx_diversity_racecategory ON Diversity(RaceCategory);
Query OK, 0 rows affected (0.83 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> explain analyze SELECT      d.UniversityName,      d.RaceCategory,      SUM(d.RaceWiseEnrollment) AS RaceWiseTotal,      (SELECT SUM(d2.RaceWiseEnrollment)      FROM Diversity d2
  WHERE d2.UniversityName = d.UniversityName) AS TotalEnrollment,      (SUM(d.RaceWiseEnrollment) /      (SELECT SUM(d2.RaceWiseEnrollment)      FROM Diversity d2
  WHERE d2.UniversityName = d.UniversityName)) * 100 AS PercentageOfEnrollment FROM      Diversity d WHERE      d.RaceCategory = 'Asian' GROUP BY      d.UniversityName,      d.RaceCategory ORDER
 BY      PercentageOfEnrollment DESC;
+-----+
| EXPLAIN
+-----+
|-----+
|  --> Sort: PercentageOfEnrollment DESC (actual time=79.510..79.953 rows=2160 loops=1)
|    --> Stream results (cost=648.75 rows=2160) (actual time=0.679..73.990 rows=2160 loops=1)
|      --> Group aggregate: sum(d.RaceWiseEnrollment), sum(d.RaceWiseEnrollment) (cost=648.75 rows=2160) (actual time=0.604..13.248 rows=2160 loops=1)
|        --> Index lookup on d using idx_diversity_racecategory (RaceCategory='Asian') (cost=432.75 rows=2160) (actual time=0.590..11.965 rows=2160 loops=1)
|-----+
|  --> Select #3 (subquery in projection; dependent)
|    --> Aggregates: sum(d2.RaceWiseEnrollment) (cost=1..48 rows=1) (actual time=0.013..0.013 rows=1 loops=2160)
|      --> Index lookup on d2 using PRIMARY (UniversityName=d.UniversityName) (cost=0.86 rows=6) (actual time=0.008..0.012 rows=11 loops=2160)
|-----+
|  --> Select #3 (subquery in projection; dependent)
|    --> Aggregates: sum(d2.RaceWiseEnrollment) (cost=1..48 rows=1) (actual time=0.013..0.013 rows=1 loops=2160)
|      --> Index lookup on d2 using PRIMARY (UniversityName=d.UniversityName) (cost=0.86 rows=6) (actual time=0.008..0.012 rows=11 loops=2160)
|-----+
|  --> Select #2 (subquery in projection; dependent)
|    --> Aggregates: sum(d2.RaceWiseEnrollment) (cost=1..48 rows=1) (actual time=0.013..0.013 rows=1 loops=2160)
|      --> Index lookup on d2 using PRIMARY (UniversityName=d.UniversityName) (cost=0.86 rows=6) (actual time=0.007..0.011 rows=11 loops=2160)
|-----+

```

### 3.4.2 Index 2

Query - CREATE INDEX idx\_user\_tuitionfeebudget ON User(TuitionFeeBudget);

Purpose - This index is intended to enhance query performance for filtering users based on their tuition fee budgets; however, it does not directly relate to the diversity query.

```

mysql> CREATE INDEX idx_user_tuitionfeebudget ON User(TuitionFeeBudget);
Query OK, 0 rows affected (0.10 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> explain analyze SELECT      d.UniversityName,      d.RaceCategory,      SUM(d.RaceWiseEnrollment) AS RaceWiseTotal,      (SELECT SUM(d2.RaceWiseEnrollment)      FROM Diversity d2      WHERE d2.UniversityName = d.UniversityName) AS TotalEnrollment,      (SUM(d.RaceWiseEnrollment) /      (SELECT SUM(d2.RaceWiseEnrollment)      FROM Diversity d2      WHERE d2.UniversityName = d.UniversityName)) * 100 AS PercentageOfEnrollment FROM      Diversity d WHERE      d.RaceCategory = 'Asian' GROUP BY      d.UniversityName,      d.RaceCategory ORDER BY      PercentageOfEnrollment DESC;
+-----+
| EXPLAIN
+--+
+-----+
|   | 
+-----+
| -> Sort: PercentageOfEnrollment DESC (actual time=77.263..77.680 rows=2160 loops=1)
|   -> Stream results (cost=1829.06 rows=1597) (actual time=0.452..72.730 rows=2160 loops=1)
|       -> Group aggregate: sum(d.RaceWiseEnrollment), sum(d.RaceWiseEnrollment) (cost=1829.06 rows=1597) (actual time=0.518..13.312 rows=2160 loops=1)
|           -> Filter: (d.RaceCategory = 'Asian') (cost=1669.35 rows=1597) (actual time=0.505..11.997 rows=2160 loops=1)
|               -> Index scan on d using PRIMARY (UniversityName=d.UniversityName) (cost=1669.35 rows=15971) (actual time=0.487..9.184 rows=23761 loops=1)
-> Select #3 (subquery in projection; dependent)
|   -> Index lookup on d2 using PRIMARY (UniversityName=d.UniversityName) (cost=0.86 rows=6) (actual time=0.007..0.012 rows=11 loops=2160)
-> Select #3 (subquery in projection; dependent)
|   -> Aggregate: sum(d2.RaceWiseEnrollment) (cost=1.48 rows=1) (actual time=0.013..0.013 rows=1 loops=2160)
|       -> Index lookup on d2 using PRIMARY (UniversityName=d.UniversityName) (cost=0.86 rows=6) (actual time=0.007..0.012 rows=11 loops=2160)
+-----+

```

### 3.4.3 Index 3

**Query - CREATE INDEX idx\_diversity\_racewiseenrollment ON Diversity(RaceWiseEnrollment);**

**Purpose -** The index on RaceWiseEnrollment is designed to reduce the cost of aggregation operations like SUM, facilitating more efficient calculations of total enrollments.

```

mysql> CREATE INDEX idx_diversity_racewiseenrollment ON Diversity(RaceWiseEnrollment);
Query OK, 0 rows affected (0.49 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> explain analyze SELECT      d.UniversityName,      d.RaceCategory,      SUM(d.RaceWiseEnrollment) AS RaceWiseTotal,      (SELECT SUM(d2.RaceWiseEnrollment)      FROM Diversity d2      WHERE d2.UniversityName = d.UniversityName) AS TotalEnrollment,      (SUM(d.RaceWiseEnrollment) /      (SELECT SUM(d2.RaceWiseEnrollment)      FROM Diversity d2      WHERE d2.UniversityName = d.UniversityName)) * 100 AS PercentageOfEnrollment FROM      Diversity d WHERE      d.RaceCategory = 'Asian' GROUP BY      d.UniversityName,      d.RaceCategory ORDER BY      PercentageOfEnrollment DESC;
+-----+
| EXPLAIN
+--+
+-----+
|   | 
+-----+
| -> Sort: PercentageOfEnrollment DESC (actual time=92.297..92.863 rows=2160 loops=1)
|   -> Stream results (cost=1829.06 rows=1597) (actual time=0.181..87.069 rows=2160 loops=1)
|       -> Group aggregate: sum(d.RaceWiseEnrollment), sum(d.RaceWiseEnrollment) (cost=1829.06 rows=1597) (actual time=0.108..15.380 rows=2160 loops=1)
|           -> Filter: (d.RaceCategory = 'Asian') (cost=1669.35 rows=1597) (actual time=0.096..13.831 rows=2160 loops=1)
|               -> Index scan on d using PRIMARY (cost=1669.35 rows=15971) (actual time=0.086..10.546 rows=23761 loops=1)
-> Select #3 (subquery in projection; dependent)
|   -> Aggregate: sum(d2.RaceWiseEnrollment) (cost=1.48 rows=1) (actual time=0.016..0.016 rows=1 loops=2160)
-> Select #3 (subquery in projection; dependent)
|   -> Aggregate: sum(d2.RaceWiseEnrollment) (cost=1.48 rows=1) (actual time=0.016..0.016 rows=1 loops=2160)
+-----+

```

### 3.4.4 Index Analysis

Index Configuration	Query Cost	Explanation
Original Query	1829.00	The query executed without any index
With idx_diversity_racecategory	648.75	The index improves filtering efficiency for the race category, significantly reducing query cost through better access paths.
With idx_user_tuitionfeebudget	11829.00	This index does not impact the current query's performance as it relates to user budgets, not diversity data.
With idx_diversity_racewiseenrollment	1829.00	No performance improvement observed due to the complexity of nested subqueries; extensive scans remain necessary.

### 3.4.5 Report Final Design

**Selected Index:** idx\_diversity\_racecategory

**Column:** Diversity.RaceCategory

**Rationale:** This index was selected to optimize the filtering of diversity data based on race categories, which is crucial for this query's performance.

**Analysis:**

- **Before Indexing:** The original query cost was 1829.00, indicating reliance on full table scans for both filtering and aggregation.
- **After Indexing:** Implementing the idx\_diversity\_racecategory index reduced the query cost to 648.75, showcasing a substantial performance improvement by allowing efficient access to records based on the specified race category.

## 3.5 Query 5

### Explain & Analyze Pre-Indexing

```
| -> Sort: RankedDiversity.RaceCategory, RankedDiversity.EnrollmentPercentage DESC (actual time=208.930..208.942 rows=124 loops=1)
    -> Stream results (cost=948797.36 rows=9443240) (actual time=208.076..208.793 rows=124 loops=1)
        -> Nested loop inner join (cost=948797.36 rows=9443240) (actual time=208.063..208.711 rows=124 loops=1)
            -> Filter: ('<subquery>'.EnrollmentPercentage is not null) (cost=778.33..35.46 rows=1774) (actual time=102.237..102.274 rows=92 loops=1)
                -> Table scan on <subquery> (cost=778.77..803.43 rows=1774) (actual time=102.235..102.262 rows=92 loops=1)
                    -> Materialize with deduplication (cost=778.77..778.75 rows=1774) (actual time=102.235..102.239 rows=92 loops=1)
                        -> Filter: (Ranked.rn <= 10) (cost=0.34..601.34 rows=1774) (actual time=98.613..102.168 rows=110 loops=1)
                            -> Table scan on Ranked (cost=2.50..2.50 rows=0) (actual time=98.604..101.053 rows=22086 loops=1)
                                -> Materialize (cost=0.00..0.00 rows=0) (actual time=98.599..98.599 rows=22086 loops=1)
                                    -> Window aggregate: row_number() OVER (PARTITION BY InnerQuery.RaceCategory ORDER BY InnerQuery.EnrollmentPercentage desc ) (actual time=82.895..96.367 rows=22086 loops=1)
                                        -> Sort: InnerQuery.RaceCategory, InnerQuery.EnrollmentPercentage DESC (cost=9924.34..9924.34 rows=5323) (actual time=82.874..85.811 rows=22086 loops=1)
                                            -> Table scan on InnerQuery (cost=2733.99..2803.01 rows=5323) (actual time=51.568..55.209 rows=22086 loops=1)
                                                -> Materialize (cost=2733.98..2733.98 rows=5323) (actual time=51.562..51.562 rows=22086 loops=1)
                                                -> Group aggregate: sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment), sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment) (cost=2201.66 rows=5323) (actual time=0.135..32.278 rows=22086 loops=1)
                                                    -> Index lookup on RankedDiversity using <auto_key> (EnrollmentPercentage=<subquery>.EnrollmentPercentage) (actual time=1.156..1.156 rows=1 loops=92)
                                                        -> Materialize (cost=2733.98..2733.98 rows=5323) (actual time=105.803..105.803 rows=22086 loops=1)
                                                            -> Filter: (D.RaceWiseEnrollment > 0) (cost=1669.35 rows=5323) (actual time=0.109..13.464 rows=22086 loops=1)
                                                                -> Index scan on D using PRIMARY (cost=1669.35 rows=15971) (actual time=0.106..10.921 rows=23761 loops=1)
|
| -> Sort: RankedDiversity.RaceCategory, RankedDiversity.EnrollmentPercentage DESC (actual time=250.028..250.040 rows=124 loops=1)
    -> Stream results (cost=2131519.42 rows=249.926 rows=124 loops=1)
        -> Nested loop inner join (cost=2131519.42 rows=21248086) (actual time=249.524..249.852 rows=124 loops=1)
            -> Filter: ('<subquery>'.EnrollmentPercentage is not null) (cost=1166.95..1166.95 rows=2661) (actual time=151.001..151.908 rows=92 loops=1)
                -> Table scan on <subquery> (cost=1166.97..1202.72 rows=2661) (actual time=151.001..151.908 rows=92 loops=1)
                    -> Materialize with deduplication (cost=1166.95..1166.95 rows=2661) (actual time=148.206..151.811 rows=110 loops=1)
                        -> Filter: (Ranked.rn <= 10) (cost=0.34..900.81 rows=2661) (actual time=148.206..151.811 rows=110 loops=1)
                            -> Table scan on Ranked (cost=2.50..2.50 rows=0) (actual time=148.198..150.677 rows=22086 loops=1)
                                -> Materialize (cost=0.00..0.00 rows=0) (actual time=148.194..148.194 rows=22086 loops=1)
                                    -> Window aggregate: row_number() OVER (PARTITION BY InnerQuery.RaceCategory ORDER BY InnerQuery.EnrollmentPercentage desc ) (actual time=125.480..139.026 rows=22086 loops=1)
                                        -> Sort: InnerQuery.RaceCategory, InnerQuery.EnrollmentPercentage DESC (cost=14518.18..14518.18 rows=7985) (actual time=125.465..128.393 rows=22086 loops=1)
                                            -> Table scan on InnerQuery (cost=3266.36..3368.66 rows=7985) (actual time=77.149..81.015 rows=22086 loops=1)
                                                -> Materialize (cost=3266.35..3266.35 rows=7985) (actual time=77.143..77.143 rows=22086 loops=1)
                                                -> Group aggregate: sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment), sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment) (cost=2467.85 rows=7985) (actual time=0.117..47.085 rows=22086 loops=1)
                                                    -> Filter: (D.RaceWiseEnrollment > 0) (cost=1669.35 rows=7985) (actual time=0.100..16.470 rows=22086 loops=1)
                                                        -> Index scan on D using PRIMARY (cost=1669.35 rows=15971) (actual time=0.099..14.073 rows=23761 loops=1)
                                                        -> Index lookup on RankedDiversity using <auto_key> (EnrollmentPercentage=<subquery>.EnrollmentPercentage) (actual time=1.064..1.064 rows=1 loops=92)
                                                            -> Materialize (cost=3266.35..3266.35 rows=7985) (actual time=97.623..97.623 rows=22086 loops=1)
                                                                -> Group aggregate: sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment), sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment) (cost=2467.85 rows=7985) (actual time=0.97..34.542 rows=22086 loops=1)
                                                                    -> Filter: (D.RaceWiseEnrollment > 0) (cost=1669.35 rows=7985) (actual time=0.082..12.611 rows=22086 loops=1)
                                                                        -> Index scan on D using PRIMARY (cost=1669.35 rows=15971) (actual time=0.080..10.470 rows=23761 loops=1)
|
|
```

### 3.5.1 Index 1

Query - CREATE INDEX idx\_diversity\_racewiseenrollment ON Diversity(RaceWiseEnrollment);

Purpose - This index aims to enhance performance for queries involving the RaceWiseEnrollment column in the Diversity table. Since this column is utilized in the WHERE clause to filter records where RaceWiseEnrollment > 0, indexing it can significantly improve the speed of this operation and reduce cost.

```
| -> Sort: RankedDiversity.RaceCategory, RankedDiversity.EnrollmentPercentage DESC (actual time=250.028..250.040 rows=124 loops=1)
    -> Stream results (cost=2131519.42 rows=249.926 rows=124 loops=1)
        -> Nested loop inner join (cost=2131519.42 rows=21248086) (actual time=249.524..249.852 rows=124 loops=1)
            -> Filter: ('<subquery>'.EnrollmentPercentage is not null) (cost=1166.95..1166.95 rows=2661) (actual time=151.001..151.908 rows=92 loops=1)
                -> Table scan on <subquery> (cost=1166.97..1202.72 rows=2661) (actual time=151.001..151.908 rows=92 loops=1)
                    -> Materialize with deduplication (cost=1166.95..1166.95 rows=2661) (actual time=148.206..151.811 rows=110 loops=1)
                        -> Filter: (Ranked.rn <= 10) (cost=0.34..900.81 rows=2661) (actual time=148.206..151.811 rows=110 loops=1)
                            -> Table scan on Ranked (cost=2.50..2.50 rows=0) (actual time=148.198..150.677 rows=22086 loops=1)
                                -> Materialize (cost=0.00..0.00 rows=0) (actual time=148.194..148.194 rows=22086 loops=1)
                                    -> Window aggregate: row_number() OVER (PARTITION BY InnerQuery.RaceCategory ORDER BY InnerQuery.EnrollmentPercentage desc ) (actual time=125.480..139.026 rows=22086 loops=1)
                                        -> Sort: InnerQuery.RaceCategory, InnerQuery.EnrollmentPercentage DESC (cost=14518.18..14518.18 rows=7985) (actual time=125.465..128.393 rows=22086 loops=1)
                                            -> Table scan on InnerQuery (cost=3266.36..3368.66 rows=7985) (actual time=77.149..81.015 rows=22086 loops=1)
                                                -> Materialize (cost=3266.35..3266.35 rows=7985) (actual time=77.143..77.143 rows=22086 loops=1)
                                                -> Group aggregate: sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment), sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment) (cost=2467.85 rows=7985) (actual time=0.117..47.085 rows=22086 loops=1)
                                                    -> Filter: (D.RaceWiseEnrollment > 0) (cost=1669.35 rows=7985) (actual time=0.100..16.470 rows=22086 loops=1)
                                                        -> Index scan on D using PRIMARY (cost=1669.35 rows=15971) (actual time=0.099..14.073 rows=23761 loops=1)
                                                        -> Index lookup on RankedDiversity using <auto_key> (EnrollmentPercentage=<subquery>.EnrollmentPercentage) (actual time=1.064..1.064 rows=1 loops=92)
                                                            -> Materialize (cost=3266.35..3266.35 rows=7985) (actual time=97.623..97.623 rows=22086 loops=1)
                                                                -> Group aggregate: sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment), sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment) (cost=2467.85 rows=7985) (actual time=0.97..34.542 rows=22086 loops=1)
                                                                    -> Filter: (D.RaceWiseEnrollment > 0) (cost=1669.35 rows=7985) (actual time=0.082..12.611 rows=22086 loops=1)
                                                                        -> Index scan on D using PRIMARY (cost=1669.35 rows=15971) (actual time=0.080..10.470 rows=23761 loops=1)
|
|
```

### 3.5.2 Index 2

Query - CREATE INDEX idx\_diversity\_totallenrollment ON Diversity(TotalEnrollment);

Purpose - This index aims to reduce query execution costs associated with operations involving the TotalEnrollment column in the Diversity table. By indexing this column, the

database can quickly locate and access the necessary records for aggregation and filtering, minimizing the need for full table scans

```
| -> Sort: RankedDiversity.RaceCategory, RankedDiversity.EnrollmentPercentage DESC (actual time=229.609..229.625 rows=124 loops=1)
    -> Stream results (cost=948797.36 rows=9443240) (actual time=229.018..229.481 rows=124 loops=1)
        -> Nested loop inner join (cost=948797.36 rows=9443240) (actual time=229.011..229.408 rows=124 loops=1)
            -> Filter: ('<subquery>'.EnrollmentPercentage is not null) (cost=778.33..35.46 rows=1774) (actual time=121.111..121.142 rows=92 loops=1)
                -> Table scan on <subquery> (cost=778.77..78.75 rows=1774) (actual time=121.109..121.129 rows=92 loops=1)
                    -> Materialize with deduplication (cost=778.75..77.75 rows=1774) (actual time=121.106..121.106 rows=92 loops=1)
                        -> Filter: (Ranked.EnrollmentPercentage <= 10) (cost=0.34..0.34 rows=1774) (actual time=17.310..121.042 rows=110 loops=1)
                            -> Table scan on Ranked (cost=2.50..2.50 rows=0) (actual time=17.306..119.762 rows=22086 loops=1)
                                -> Materialize (cost=0.00..0.00 rows=0) (actual time=17.302..117.302 rows=22086 loops=1)
                                    -> Window aggregate: row_number() OVER (PARTITION BY InnerQuery.RaceCategory ORDER BY InnerQuery.EnrollmentPercentage desc ) (actual time=99.345..114.802 rows=22086 loops=1)
                                        -> Sort: InnerQuery.RaceCategory, InnerQuery.EnrollmentPercentage DESC (cost=9924.34..9924.34 rows=5323) (actual time=99.327..102.571 rows=22086 loops=1)
                                            -> Table scan on InnerQuery (cost=2733.99..2803.01 rows=5323) (actual time=54.785..58.224 rows=22086 loops=1)
                                                -> Group aggregate: sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment), sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment) (cost=2201.66 rows=5323) (actual time=0.102..33.993 rows=22086 loops=1)
                                                    -> Filter: (D.RaceWiseEnrollment > 0) (cost=1669.35 rows=5323) (actual time=0.089..12.392 rows=22086 loops=1)
                                                        -> Index lookup on RankedDiversity using <auto_key0> (EnrollmentPercentage=<subquery>.EnrollmentPercentage) (actual time=1.176..1.176 rows=1 loops=92)
                                                            -> Materialize (cost=2733.98..2733.98 rows=5323) (actual time=107.881..107.881 rows=22086 loops=1)
                                                                -> Group aggregate: sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment), sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment) (cost=2201.66 rows=5323) (actual time=0.96..35.841 rows=22086 loops=1)
                                                                    -> Filter: (D.RaceWiseEnrollment > 0) (cost=1669.35 rows=5323) (actual time=0.081..13.101 rows=22086 loops=1)
                                                                        -> Index scan on D using PRIMARY (cost=1669.35 rows=15971) (actual time=0.079..10.816 rows=23761 loops=1)
|
```

### 3.5.3 Index 3

Query - CREATE INDEX idx\_diversity\_enrollmentpercentage ON Diversity(RaceWiseEnrollment, TotalEnrollment);

Purpose - This index aims to optimize the performance of queries that involve both the RaceWiseEnrollment and TotalEnrollment columns in the Diversity table. By indexing these two columns together, the database can efficiently perform calculations and aggregations that rely on both attributes, such as determining enrollment percentages.

```
| -> Sort: RankedDiversity.RaceCategory, RankedDiversity.EnrollmentPercentage DESC (actual time=225.961..225.973 rows=124 loops=1)
    -> Stream results (cost=2131519.42 rows=21248086) (actual time=225.411..225.855 rows=124 loops=1)
        -> Nested loop inner join (cost=2131519.42 rows=21248086) (actual time=225.403..225.797 rows=124 loops=1)
            -> Filter: ('<subquery>'.EnrollmentPercentage > 0) (cost=1166.53..53.19 rows=2661) (actual time=124.261..124.291 rows=92 loops=1)
                -> Table scan on <subquery> (cost=1166.53..1202.72 rows=2661) (actual time=124.260..124.277 rows=92 loops=1)
                    -> Materialize with deduplication (cost=1166.53..1202.72 rows=2661) (actual time=124.260..124.277 rows=92 loops=1)
                        -> Filter: (Ranked.EnrollmentPercentage is not null) (cost=0.34..0.34 rows=2661) (actual time=120.708..124.199 rows=110 loops=1)
                            -> Table scan on Ranked (cost=2.50..2.50 rows=0) (actual time=120.706..124.188 rows=110 loops=1)
                                -> Materialize (cost=0.00..0.00 rows=0) (actual time=120.698..120.698 rows=22086 loops=1)
                                    -> Window aggregate: row_number() OVER (PARTITION BY InnerQuery.RaceCategory ORDER BY InnerQuery.EnrollmentPercentage desc ) (actual time=105.078..118.553 rows=22086 loops=1)
                                        -> Sort: InnerQuery.RaceCategory, InnerQuery.EnrollmentPercentage DESC (cost=14518.18..14518.18 rows=7985) (actual time=105.062..107.960 rows=22086 loops=1)
                                            -> Table scan on InnerQuery (cost=3266.36..3368.66 rows=7985) (actual time=54.512..58.044 rows=22086 loops=1)
                                                -> Materialize (cost=3266.35..3266.35 rows=7985) (actual time=54.508..54.508 rows=22086 loops=1)
                                                    -> Group aggregate: sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment), sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment) (cost=2467.85 rows=7985) (actual time=0.135..34.041 rows=22086 loops=1)
                                                        -> Filter: (D.RaceWiseEnrollment > 0) (cost=1669.35 rows=7985) (actual time=0.122..12.502 rows=22086 loops=1)
                                                            -> Index scan on D using PRIMARY (cost=1669.35 rows=15971) (actual time=0.120..10.434 rows=23761 loops=1)
                                                        -> Index lookup on RankedDiversity using <auto_key0> (EnrollmentPercentage=<subquery>.EnrollmentPercentage) (actual time=1.103..1.103 rows=1 loops=92)
                                                            -> Materialize (cost=3266.35..3266.35 rows=7985) (actual time=101.127..101.127 rows=22086 loops=1)
                                                                -> Group aggregate: sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment), sum(D.TotalEnrollment), sum(D.RaceWiseEnrollment) (cost=2467.85 rows=7985) (actual time=0.96..37.254 rows=22086 loops=1)
                                                                    -> Filter: (D.RaceWiseEnrollment > 0) (cost=1669.35 rows=7985) (actual time=0.081..13.420 rows=22086 loops=1)
                                                                        -> Index scan on D using PRIMARY (cost=1669.35 rows=15971) (actual time=0.080..11.027 rows=23761 loops=1)
|
```

### 3.5.4 Index Analysis

Index Configuration	Query Cost	Explanation
Original Query	948797.36	The query executed without any index, leading to full table scans for the Diversity table, resulting in a high cost.
With idx_diversity_racecategory	2131519.42	The query cost increased significantly, indicating that the index did not help

		optimize the query performance. The presence of multiple nested subqueries may still require full scans, negating the benefits of the index.
With idx_user_tuitionfeebudget	948797.36	No performance improvement observed; the cost remained the same, indicating that the index on TotalEnrollment did not contribute to efficiency in this case.
With idx_diversity_racewiseenrollment	2131519.42	Similar to the index on RaceWiseEnrollment, this index also resulted in increased cost, suggesting that the complexity of the query structure continues to require comprehensive scans..

### 3.5.5 Report Final Design

**Selected Index:** None

**Column:** N/A

**Rationale:**

Given the performance analysis, no effective indexing design was found to improve query performance significantly. The existing indices either resulted in increased costs or no change at all, primarily due to the complex nature of the query involving nested subqueries, which necessitated full scans of the table.

**Analysis:**

- **Before Indexing:** The original query cost was 948797.36, attributed to full table scans on the Diversity table. This high cost indicates that the query was not optimized for performance.
- **After Indexing:** The attempts to implement indices resulted in increased costs, especially with idx\_diversity\_racewiseenrollment and idx\_diversity\_enrollmentpercentage, which drove costs up to 2131519.42. This suggests that the nested subqueries may still force full scans, limiting the effectiveness of the indices.

**Summary:**

The analysis revealed that the query's structure and complexity hindered the anticipated performance improvements from indexing. The increased costs associated with certain indices indicate that further optimization may be required, either by restructuring the query to minimize the reliance on nested subqueries or by exploring different indexing strategies to enhance efficiency.