

Stage 4

Constraint

We have multiple foreign keys and primary keys constraints in our database according to the schema. Additionally, we have identified and implemented this constraint.

ALTER table Users add constraint chk_userType check (userType IN (0, 1, 2, 3));

```
mysql> ALTER table Users add constraint chk_userType check (userType IN (0, 1, 2, 3));
Query OK, 1600 rows affected (0.53 sec)
Records: 1600  Duplicates: 0  Warnings: 0
```

Procedure

We have written a procedure to fetch the below average students. We create a cursor to find students with low attendance. Low attendance is attendance lower than the provided attendance threshold percentage. We also find students whose grades are lower than average grades. We join these two tables as our below average students.

DELIMITER //

CREATE PROCEDURE FetchBelowAverageUsersOfClassroomAndClassGroup(IN in_classroomId
VARCHAR(10), IN in_classGroupId VARCHAR(10), IN in_attendanceThresholdPercentage
DECIMAL(5,2))

BEGIN

DECLARE totalAttendanceDaysCount INT;

DECLARE avgGradesCount INT;

DECLARE curStudentId varchar(10);

DECLARE curStudentFirstName varchar(255);

DECLARE curStudentLastName varchar(255);

DECLARE curAttendanceDaysCount INT;

DECLARE curAttendanceRate DECIMAL(5,2);

DECLARE attendanceDone INT DEFAULT 0;

DECLARE studentAttendanceCursor CURSOR FOR SELECT studentId, firstName, lastName,
sum(isPresent) AS attendanceDays from Attendance JOIN Users on studentId = userId WHERE
classroomId = in_classroomId GROUP BY studentId;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET attendanceDone = 1;

SELECT COUNT(DISTINCT attendanceDate) into totalAttendanceDaysCount FROM Attendance
WHERE classroomId=in_classroomId;

SELECT avg(maximumGrade) into avgGradesCount from Assignment where
classGroupId=in_classGroupId;

DROP TABLE IF EXISTS LowAttendanceStudents;

```
CREATE TABLE LowAttendanceStudents (userId VARCHAR(10) PRIMARY KEY, firstName
varchar(255), lastName varchar(255));
```

```
DROP TABLE IF EXISTS LowGradeStudents;
```

```
CREATE TABLE LowGradeStudents (userId VARCHAR(10) PRIMARY KEY, firstName
varchar(255), lastName varchar(255));
```

```
OPEN studentAttendanceCursor;
```

```
attendanceLoop: REPEAT
```

```
    FETCH studentAttendanceCursor into curStudentId, curStudentFirstName,
curStudentLastName, curAttendanceDaysCount;
```

```
    IF attendanceDone = 1 THEN
```

```
        LEAVE attendanceLoop;
```

```
    END IF;
```

```
    SET curAttendanceRate = curAttendanceDaysCount/totalAttendanceDaysCount*100;
```

```
    IF curAttendanceRate < in_attendanceThresholdPercentage THEN
```

```
        insert into LowAttendanceStudents values(curStudentId, curStudentFirstName,
curStudentLastName);
```

```
    END IF;
```

```
UNTIL attendanceDone END REPEAT;
```

```
CLOSE studentAttendanceCursor;
```

```
INSERT INTO LowGradeStudents (userId, firstName, lastName) SELECT userId, firstName,
lastName from Grades NATURAL JOIN Users where classGroupId=in_classGroupId GROUP BY
userId having avg(grade) < avgGradesCount;
```

```
SELECT * from LowAttendanceStudents NATURAL JOIN LowGradeStudents;
END //
```

```
DELIMITER ;
```

Trigger

We have written a trigger which runs on removal of students in a classroom group. This checks if these students have any grades. If grades exist, they are removed for that classroom group.

```
DELIMITER //
```

```
CREATE TRIGGER DeleteUserGrades
AFTER DELETE ON ClassroomUsers
FOR EACH ROW
BEGIN
    IF EXISTS (
        SELECT 1 FROM Grades WHERE userId = OLD.userId and classGroupId =
OLD.classGroupId and classroomId = OLD.classroomId and courseId = OLD.courseId
    ) THEN
        DELETE FROM Grades where userId = OLD.userId and classGroupId = OLD.classGroupId and
classroomId = OLD.classroomId and courseId = OLD.courseId;
    END IF;
END //
```

```
DELIMITER ;
```

```
mysql> DELIMITER //
mysql> CREATE TRIGGER DeleteUserGrades
mysql> AFTER DELETE ON ClassroomUsers
-> FOR EACH ROW
-> BEGIN
-> IF EXISTS (
-> SELECT 1 FROM Grades WHERE userId = OLD.userId and classGroupId = OLD.classGroupId and classroomId = OLD.classroomId and courseId = OLD.courseId
-> ) THEN
-> DELETE FROM Grades where userId = OLD.userId and classGroupId = OLD.classGroupId and classroomId = OLD.classroomId and courseId = OLD.courseId;
-> END IF;
-> END //
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> DELIMITER ;
mysql>
```

Transactions

We have written a transaction to fetch the student analytics for a userId. By entering the userId, it first checks if the student exists in the database and throws an error. Otherwise, it fetched the user, assignments and attendance data.

```
DELIMITER //
```

```
CREATE PROCEDURE FetchStudentAnalytics(
    IN in_userId VARCHAR(255))
fetchStudentAnalytics: BEGIN
    DECLARE existUser INT DEFAULT 0;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;
```

```

SELECT count(*) into existUser from Users where userId = in_userId and userType = 2;
IF existUser = 0 THEN
    SELECT 'User not found' as Message;
    ROLLBACK;
    LEAVE fetchStudentAnalytics;
END IF;

```

```

SELECT * FROM Users where userId = in_userId;

```

```

SELECT co.subjectName as subjectName, cr.className as className, cg.classGroupId as
classGroupId, a.assignmentId as assignmentId, g.grade as grade, maxGrade.topperGrade as
topperGrade, a.maximumGrade as maximumPossibleGrade from ClassroomUsers cu NATURAL
JOIN ClassroomGroups cg NATURAL JOIN Courses co NATURAL JOIN Classrooms cr NATURAL
JOIN Assignment a NATURAL JOIN Grades g JOIN (select assignmentId, max(grade) as
topperGrade from Grades group by assignmentId) as maxGrade ON g.assignmentId =
maxGrade.assignmentId where cu.userId=in_userId;

```

```

SELECT a1.studentId, a1.attendanceDate, a1.isPresent AS userAttendance, (SELECT COUNT(*)
from Attendance as a2 where a2.attendanceDate = a1.attendanceDate and a2.classroomId =
a1.classroomId and a2.isPresent = 1) AS totalPresent, (SELECT COUNT(*) from Attendance as a2
where a2.attendanceDate = a1.attendanceDate and a2.classroomId = a1.classroomId and a2.isPresent
= 0) AS totalAbsent FROM Attendance a1 where a1.studentId = in_userId;

```

```

COMMIT;
END //

```

```

DELIMITER ;

```

```

mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE FetchStudentAnalytics(
mysql>     IN in_userId VARCHAR(255))
mysql>     fetchStudentAnalytics: BEGIN
mysql>     DECLARE existUser INT DEFAULT 0;
mysql>     SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
mysql>     START TRANSACTION;
mysql>     SELECT count(*) into existUser from Users where userId = in_userId and userType = 2;
mysql>     IF existUser = 0 THEN
mysql>         SELECT 'User not found' as Message;
mysql>         ROLLBACK;
mysql>         LEAVE fetchStudentAnalytics;
mysql>     END IF;
mysql>     SELECT * FROM Users where userId = in_userId;
mysql>     SELECT co.subjectName as subjectName, cr.className as className, cg.classGroupId as classGroupId, a.assignmentId as assignmentId, g.grade as grade, maxGrade.topperGrade as topperGr
ade, a.maximumGrade as maximumPossibleGrade from ClassroomUsers cu NATURAL JOIN ClassroomGroups cg NATURAL JOIN Courses co NATURAL JOIN Classrooms cr NATURAL JOIN Assignment a NATURAL JOIN G
rades g JOIN (select assignmentId, max(grade) as topperGrade from Grades group by assignmentId) as maxGrade ON g.assignmentId = maxGrade.assignmentId where cu.userId=in_userId;
mysql>     SELECT a1.studentId, a1.attendanceDate, a1.isPresent AS userAttendance, (SELECT COUNT(*) from Attendance as a2 where a2.attendanceDate = a1.attendanceDate and a2.classroomId = a1.clas
sroomId and a2.isPresent = 1) AS totalPresent, (SELECT COUNT(*) from Attendance as a2 where a2.attendanceDate = a1.attendanceDate and a2.classroomId = a1.classroomId and a2.isPresent = 0) AS
totalAbsent FROM Attendance a1 where a1.studentId = in_userId;
mysql>     COMMIT;
mysql> END //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql>

```

Stage 3 Improvements

As discussed in our previous group chat on ED, we have resubmitted the Query 4 Index 2 to receive full points on the previous stage due to the miscommunication in the previous stage. Previous find the index and results below.

Query4:

We want the admins to get a list of toppers from each classgroup. For this, we write a query which selects classGroupId, userId, firstName, lastName of the topper, and Average grade of the topper across all assignments as TopperAverage.

```
select g.classGroupId, u.userId as TopperUserId, u.firstName AS TopperFirstName,
u.lastName AS TopperLastName, Round(max(g.grade/a.maximumGrade*100), 2) as
TopperAverage from Assignment a NATURAL JOIN Grades g NATURAL JOIN Users u
group by g.classGroupId, u.userId having (g.classGroupId, TopperAverage) IN (select
g.classGroupId, Round(max(g.grade/a.maximumGrade*100), 2) as TopperAverage from
Assignment a JOIN Grades g on a.assignmentId=g.assignmentId group by g.classGroupId
order by g.classGroupId) order by g.classGroupId;
```

Index 2:

Before Indexing:

```
| -> Sort: g.classGroupId, u.userId (actual time=13.596..13.607 rows=106 loops=1)
|   -> Filter: <in_optimizer>((g.classGroupId,TopperAverage) in (select #2)) (actual time=12.289..13.518 rows=106 loops=1)
|     -> Table scan on <temporary> (actual time=9.261..9.433 rows=701 loops=1)
|       -> Aggregate using temporary table (actual time=9.258..9.258 rows=701 loops=1)
|         -> Nested loop inner join (cost=1390.80 rows=1681) (actual time=0.148..6.604 rows=1681 loops=1)
|           -> Nested loop inner join (cost=762.45 rows=1681) (actual time=0.130..5.471 rows=1681 loops=1)
|             -> Table scan on g (cost=174.10 rows=1681) (actual time=0.100..0.945 rows=1681 loops=1)
|             -> Single-row index lookup on a using PRIMARY (classroomId=g.classroomId, courseId=g.courseId, classGroupId=g.classGroupId, assignmentId=g.assignmentId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1681)
|           -> Single-row index lookup on u using PRIMARY (userId=g.userId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1681)
|         -> Select #2 (subquery in condition; run only once)
|           -> Filter: ((g.classGroupId = <materialized_subquery>'.classGroupId) and (TopperAverage = <materialized_subquery>'.TopperAverage)) (cost=0.00..0.00 rows=0) (actual time=0.005..0.005 rows=0 loops=70)
|         2)
|           -> Limit: 1 row(s) (cost=0.00..0.00 rows=0) (actual time=0.005..0.005 rows=0 loops=702)
|           -> Index lookup on <materialized_subquery> using <auto distinct key> (classGroupId=g.classGroupId, TopperAverage=TopperAverage) (actual time=0.005..0.005 rows=0 loops=702)
|           -> Materialize with deduplication (cost=0.00..0.00 rows=0) (actual time=2.957..2.957 rows=87 loops=1)
|           -> Table scan on <temporary> (actual time=2.863..2.879 rows=87 loops=1)
|             -> Aggregate using temporary table (actual time=2.861..2.861 rows=87 loops=1)
|               -> Inner hash join (g.assignmentId = a.assignmentId) (cost=48616.85 rows=48581) (actual time=0.444..1.779 rows=1681 loops=1)
|                 -> Table scan on g (cost=0.08 rows=1681) (actual time=0.066..0.822 rows=1681 loops=1)
|                 -> Hash
|                   -> Table scan on a (cost=29.65 rows=289) (actual time=0.128..0.230 rows=289 loops=1)
```

Index used:

```
mysql> CREATE INDEX idx_ass_assignmentID ON Assignment(assignmentID);
Query OK, 0 rows affected (0.23 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Result: (Cost reduced 29.6 to 0.25)

We have performed indexing on the attribute assignmentID on the weak entity table Assignments. This results in a very significant reduction of cost from 29.6 to 0.25 due to the use of assignmentID of Assignments while performing joins on Assignments and Grades. The use of index can be seen on the last line of the result image below

```

-> Sort: g.classGroupId, u.userId (actual time=21.391..21.402 rows=106 loops=1)
-> Filter: <in_optimizer>{(g.classGroupId,TopperAverage),(g.classGroupId,TopperAverage) in (select #2)} (actual time=20.044..21.292 rows=106 loops=1)
-> Table scan on <temporary> (actual time=8.789..8.965 rows=701 loops=1)
-> Aggregate using temporary table (actual time=8.785..8.785 rows=701 loops=1)
-> Nested loop inner join (cost=1350.80 rows=1681) (actual time=0.106..6.270 rows=1681 loops=1)
-> Nested loop inner join (cost=762.45 rows=1681) (actual time=0.893..2.160 rows=1681 loops=1)
-> Table scan on g (cost=174.10 rows=1681) (actual time=0.071..0.842 rows=1681 loops=1)
-> Single-row index lookup on a using PRIMARY (classroomId=g.classroomId, courseId=g.courseId, classGroupId=g.classGroupId, assignmentId=g.assignmentId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1681)
-> Single-row index lookup on u using PRIMARY (userId=g.userId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1681)
-> Select #2 (subquery in condition; run only once)
-> Filter: ((g.classGroupId = <materialized_subquery>'.classGroupId) and (TopperAverage = <materialized_subquery>'.TopperAverage)) (cost=0.00..0.00 rows=0) (actual time=0.017..0.017 rows=0 loops=702)
2)
-> Limit: 1 row(s) (cost=0.00..0.00 rows=0) (actual time=0.017..0.017 rows=0 loops=702)
-> Index lookup on <materialized_subquery> using <auto distinct key> (classGroupId=g.classGroupId, TopperAverage=TopperAverage) (actual time=0.017..0.017 rows=0 loops=702)
-> Materialize with deduplication (cost=0.00..0.00 rows=0) (actual time=11.175..11.178 rows=87 loops=1)
-> Table scan on <temporary> (actual time=11.106..11.122 rows=87 loops=1)
-> Aggregate using temporary table (actual time=11.103..11.103 rows=87 loops=1)
-> Nested loop inner join (cost=762.45 rows=1681) (actual time=0.084..9.776 rows=1681 loops=1)
-> Table scan on g (cost=174.10 rows=1681) (actual time=0.057..0.886 rows=1681 loops=1)
-> Index lookup on a using idx_ess_assignId (assignmentId=g.assignmentId) (cost=0.25 rows=1) (actual time=0.004..0.005 rows=1 loops=1681)

```