Adam Aguilar

Reese Metcalf

Brandon Hohn

Schema Squad

Sprint 2 – Week 4

Project Status Summary (Brandon):

With the benefit of hindsight, we have realized how much more detailed and verbose the user story summaries need to be; we have begun to add in the more niche but needed assumptions and specifications in order to adhere to that requirement, taking into account things like user interface and services. We believe we have finally decided on a group name and are proud to call ourselves the Scheme Squad!

While user story summary 1 was nearly complete, we have been refining the code throughout the week, looking for weak points and beginning the process of stress testing, identifying any potentially large Achilles heels. While we are about half way through the project, there still are some features we would like to implement and refine across the board should there be time, and if we are able to, we will need to modify some of the code and assumption for all the rest of the repository in order to work with it.

We have also begun experimenting with AI's potential for discovering more niche stress tests and user cases that could potentially catch flaws that a student's eye may not find; it has already helped quite a bit, though we have had some issues with it assuming and making too many inferences on its own accord, so that's something that we still need to ensure remains in check. AI's ability to find these more "hyper-niche" cases in a reasonable time is a powerful help.

We went through the setup phases for creating a public repository on GitHub to easily share code and documents in one organized space, allowing us to easily iterate over other group members code as well as easily check and stress test without needing to bug that user for their current iteration. We also have set up a Notion page for the group to be able to include "dirty notes"; if there's a half sentence that we just don't want to forget and read later, Notion is our place.

Going forward, collaboration will be much easier. With the majority of the base being implemented, we now get to spend time refining and giving each other feedback.

Sprint retrospective meeting summary (Reese):

During this sprint, the team made significant progress across several core development tasks within the database project. All four sprint items, SCRUM-19, SCRUM-20, SCRUM-21, and SCRUM-22 were successfully marked. This sprint largely focused on completing foundational code work for multiple user stories and ensuring that the logical database model evolved in alignment with new system insights.

What Went Well

Adam and Brandon successfully completed user stories 2 and 3 which involved finishing the core code logic and accompanying tests for User Stories 2 and 3. These efforts provided the project with concrete, working code that future testing and validation will rely on. The clear structure provided also makes these components easier to expand and integrate into future features. Reese finalized executing stress tests for User Story 1. The results from this testing phase gave the team an early understanding of scalability, performance limitations, and potential architectural adjustments within User Story 1 and other model limitations. The team now has actionable data for identifying bottlenecks before further deployment or integration. Adam reviewed and updated the logical database model to account for new user story details and changing system requirements. This refinement ensures that the underlying schema remains structurally sound, normalized, and aligned with the growing set of defined business rules. This work also reduces the risk of schema-related rework later.

What Could Be Improved

The sprint tasks were completed successfully, though some responsibilities naturally concentrated on code implementation while others required deep conceptual work such as modeling or testing. In future sprints, distributing these responsibilities more evenly across the team could improve involvement and ensure that knowledge is shared rather than isolated. Although stress testing was completed, it became clear that some performance considerations would have been easier to address if found earlier. This was most seen in User Story #1. There were a multitude of issues that range from logical (like rollback points) to conceptual (like how we check if hours change). There has already entered the “In Progress” phase for the next sprint. A clearer roadmap for how these in-progress tasks integrate with future deliverables may help the team maintain progress and avoid overlapping work.

What We Will Do Next

Focus on stress-testing User Stories 2 and 3. These results will help determine whether the newly completed base code behaves correctly under heavier loads and varied input conditions. With base code and stress testing nearing completion, the next sprint will focus on combining elements of the system into integrated workflows. This will include validating relationships

between tables, enforcing constraints, and confirming that the logical model and implementation remain aligned as complexity grows.

Epic Summary (Brandon):

Our group selected the CBU Tutoring Center Management System as the focus for our database project. This project is designed to support the Academic Success Center at California Baptist University by improving the way tutoring operations are organized and managed. The database will centralize data related to tutors, students, appointments, and session feedback, eliminating the need for manual tracking and allowing staff to efficiently coordinate tutoring services across departments.

The main goal of this project is to create a relational database that enhances the efficiency and accuracy of tutoring management. Currently, many academic centers rely on spreadsheets or manual sign-in sheets to handle scheduling, student-tutor matching, and feedback collection. Our system will automate these processes, making it easier for administrators to schedule sessions, match students to the appropriate tutors based on subject areas, and monitor feedback trends. This will not only improve organization and data accuracy but also help the Academic Success Center identify areas for program improvement.

The system will include features such as tutor management to store and manage tutor profiles including their departments, subject specialties, and availability; student records to maintain student information such as academic departments/courses; appointment scheduling to create and track tutoring sessions while ensuring accurate scheduling and assignment of tutors; session feedback tracking to evaluate tutoring effectiveness.

Upon completion, the CBU Tutoring Center Management System will provide a streamlined and data-driven solution for managing the Academic Success Center's operations. The system will reduce manual work, improve data organization, and enhance decision-making through insights into tutoring effectiveness and student engagement. Ultimately, this project aims to strengthen the support for CBU students and contribute to a more efficient and responsive tutoring program.

User story summaries (All):

User story #1 (Reese):

The Tutor will login to Inside CBU's tutoring center and then click on the availability section to enter their availability. From that point they will enter the days of the week alongside the hours that they are available within that week (assumed on a weekly schedule). There is no limit on their days that they can tutor throughout the week (even weekends). The only limit is the number of hours that they can work in the week (10 hours). This is to ensure that they do not overload hours and simply work rather than do their own schoolwork (assumed student tutors only). After all, information is input and checked by the system for valid hours and availability; then the database will store their information.

User story #2 (Adam):

- Admin user – A staff member responsible for hiring new tutors and assigning them subjects and availability.
- Story Summary – A newly hired tutor needs to be added to the system. The admin must create their tutor profile, assign the subjects they can teach, and enter their initial availability. The system must accept tutor data, ensure that availability follows the 10-hour rule, and immediately make the tutor visible to students searching for help in those subjects.
- Problem Statement – Without a structured workflow, new tutors may not appear correctly in student searches, causing scheduling issues or tutors to be unavailable for booking.
- Functional Description – The admin logs into the system and opens the "Add new tutor" button. The admin enters the tutor's personal information, selects one or more subjects the tutor will teach, and enters the tutor's initial availability.
- Acceptance Criteria – A new row is added to tutor, at least one subject is assigned to the tutor via TutorSubject, availability slots are successfully inserted and total weekly availability should be greater than 10, no duplicate tutor emails allowed, the tutor appears in the available tutors query for the assigned subjects, the SQL script demonstrating this story runs successfully with no errors.

User story #3 (Brandon): Student Viewing Available Time Slots For A Class

Persona:

A Valid Student

Story Summary:

A student with a valid account, wants to view the current available tutoring time slots for a given class.

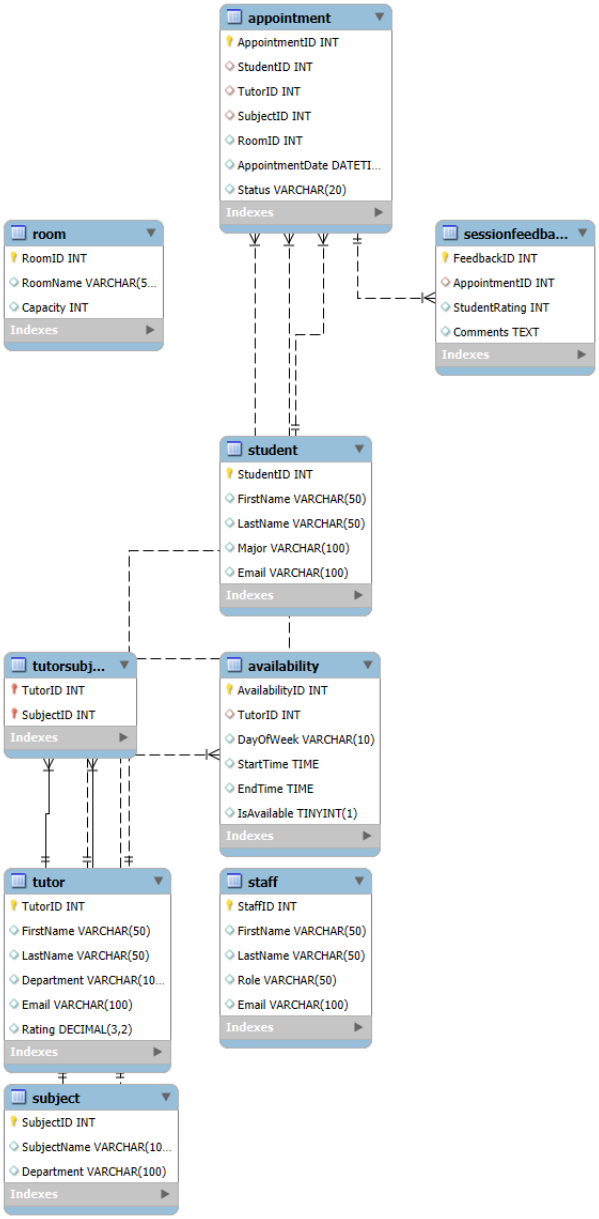
Functional Description:

The student with a valid account is able to log in, enter a class they desire to be tutored in. After submitting the request through a user interface that interacts with the backend, the available tutoring appointments are displayed to them.

Acceptance Criteria:

When the student is logged in, with a valid account, and queries using valid course information, a list of available tutoring appointments are displayed to them.

Graphical representation of the Logical Model (Adam):



All SQL scripts developed thus far (All):

User Story #1:

```
SET @TutorID := 8;

SET @MaxHoursPerWeek := 10;

-- 1. CREATE TEMPORARY

DROP TEMPORARY TABLE IF EXISTS TempNewSlots;
```

```
CREATE TEMPORARY TABLE TempNewSlots (
   DayOfWeek
ENUM('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'),
   StartTime TIME,
   EndTime   TIME
);
-- Example new slots
INSERT INTO TempNewSlots (DayOfWeek, StartTime, EndTime) VALUES
('Monday',    '09:00:00', '11:00:00'),
('Wednesday', '13:00:00', '15:00:00'),
('Friday',    '10:00:00', '12:00:00');
-- 2. CREATE REAL TABLE COPY FOR SELF-JOIN
DROP TABLE IF EXISTS TempSlotsCopy;
CREATE TABLE TempSlotsCopy AS
SELECT * FROM TempNewSlots;
ALTER TABLE TempSlotsCopy
ADD COLUMN SlotID INT AUTO_INCREMENT PRIMARY KEY;
-- 3. CALCULATE HOURS FOR NEW SLOTS
SELECT
   IFNULL(SUM(TIMESTAMPDIFF(MINUTE, StartTime, EndTime)) / 60.0, 0)
INTO @NewHours
FROM TempNewSlots;
-- 4. CALCULATE EXISTING HOURS
SELECT
   IFNULL(SUM(TIMESTAMPDIFF(MINUTE, StartTime, EndTime)) / 60.0, 0)
INTO @ExistingHours
FROM Availability
WHERE TutorID = @TutorID
```

```
  AND IsAvailable = TRUE;

-- 5. TOTAL HOURS AFTER UPDATE
SET @TotalHours := @NewHours + @ExistingHours;
-- 6. OVERLAP CHECK: NEW SLOTS AMONG THEMSELVES
SELECT EXISTS (
   SELECT 1
   FROM TempSlotsCopy A
   JOIN TempSlotsCopy B
    ON A.DayOfWeek = B.DayOfWeek
    AND A.StartTime < B.EndTime
    AND B.StartTime < A.EndTime
    AND A.SlotID < B.SlotID
) INTO @OverlapNew;
-- 7. OVERLAP CHECK: NEW SLOTS VS EXISTING AVAILABILITY
SELECT EXISTS (
   SELECT 1
   FROM TempNewSlots n
   JOIN Availability a
    ON a.TutorID = @TutorID
    AND a.IsAvailable = TRUE
    AND n.DayOfWeek = a.DayOfWeek
    AND n.StartTime < a.EndTime
    AND a.StartTime < n.EndTime
) INTO @OverlapExisting;
-- 8. VALIDATION LOGIC
SET @IsValid := (
   CASE
```

```
      WHEN @TotalHours <= @MaxHoursPerWeek
       AND @OverlapNew = 0
       AND @OverlapExisting = 0
      THEN 1
      ELSE 0
   END
);
-- 9. CONDITIONAL UPDATE (ATOMIC)
START TRANSACTION;
-- Delete existing availability only if valid
DELETE FROM Availability
WHERE TutorID = @TutorID
  AND @IsValid = 1;
-- Insert new availability only if valid
INSERT INTO Availability (TutorID, DayOfWeek, StartTime, EndTime, IsAvailable)
SELECT @TutorID, DayOfWeek, StartTime, EndTime, TRUE
FROM TempNewSlots
WHERE @IsValid = 1;
COMMIT;
-- 10. STATUS MESSAGE
SELECT
   CASE
      WHEN @IsValid = 1 THEN
         'SUCCESS: Availability updated.'
      WHEN @OverlapNew = 1 THEN
         'ERROR: New time slots overlap each other.'
      WHEN @OverlapExisting = 1 THEN
         'ERROR: New time slots overlap existing availability.'
```

```
      WHEN @TotalHours > @MaxHoursPerWeek THEN
        CONCAT('ERROR: Total hours (', @TotalHours,
            ') exceed limit of ', @MaxHoursPerWeek, '.')
      ELSE
        'ERROR: Unknown validation failure.'
   END AS Message;
-- 11. SHOW UPDATED SCHEDULE (ONLY IF VALID)
SELECT
   a.TutorID,
   CONCAT(t.FirstName, ' ', t.LastName) AS Tutor,
   a.DayOfWeek,
   a.StartTime,
   a.EndTime,
   TIMESTAMPDIFF(MINUTE, a.StartTime, a.EndTime) / 60.0 AS Hours
FROM Availability a
JOIN Tutor t ON a.TutorID = t.TutorID
WHERE a.TutorID = @TutorID
  AND @IsValid = 1
ORDER BY FIELD(a.DayOfWeek,
   'Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'),
   a.StartTime;
-- 12. CLEANUP
DROP TABLE TempSlotsCopy;
```

User story #2:

```
-- ================================================================
-- User Story #2: Admin Adds a New Tutor
-- Persona: Admin User
```

```
-- Goal: Add a new tutor profile, assign subjects, and set initial
--      availability while enforcing the 10-hour-per-week rule and
--      preventing overlapping availability slots.
-- ========================================================================

SET @MaxHoursPerWeek := 10;

-- 1. INSERT NEW TUTOR PROFILE
INSERT INTO Tutor (FirstName, LastName, Department, Email, Rating)
VALUES ('Sarah', 'Ramirez', 'Mathematics', 'sarah.ramirez@calbaptist.edu', 0.00);

SET @TutorID := LAST_INSERT_ID();

-- 2. ASSIGN SUBJECTS TO TUTOR (example: Algebra = 1, Calculus = 2)
INSERT INTO TutorSubject (TutorID, SubjectID)
VALUES
  (@TutorID, 1),  -- Algebra
  (@TutorID, 2);  -- Calculus



-- 3. DEFINE INITIAL AVAILABILITY SLOTS IN TEMP TABLE
DROP TEMPORARY TABLE IF EXISTS TempNewSlots;

CREATE TEMPORARY TABLE TempNewSlots (
  DayOfWeek
ENUM('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'),
  StartTime TIME,
  EndTime   TIME
);
```

```
-- Example: 6 total hours
INSERT INTO TempNewSlots (DayOfWeek, StartTime, EndTime) VALUES
  ('Monday',    '09:00:00', '11:00:00'),
  ('Wednesday', '10:00:00', '12:00:00'),
  ('Friday',    '14:00:00', '16:00:00');


-- 4. CREATE REAL TABLE COPY FOR SELF-JOIN (OVERLAP CHECK)
DROP TABLE IF EXISTS TempSlotsCopy;

CREATE TABLE TempSlotsCopy AS
SELECT * FROM TempNewSlots;

ALTER TABLE TempSlotsCopy
ADD COLUMN SlotID INT AUTO_INCREMENT PRIMARY KEY;


-- 5. CALCULATE HOURS FOR NEW SLOTS (IN HOURS)
SELECT
  IFNULL(SUM(TIMESTAMPDIFF(MINUTE, StartTime, EndTime)) / 60.0, 0)
INTO @NewHours
FROM TempNewSlots;

-- 6. CALCULATE EXISTING HOURS FOR THIS TUTOR (SHOULD BE 0 FOR NEW TUTOR, BUT KEPT FOR CONSISTENCY)
SELECT
  IFNULL(SUM(TIMESTAMPDIFF(MINUTE, StartTime, EndTime)) / 60.0, 0)
INTO @ExistingHours
```

```
FROM Availability
WHERE TutorID = @TutorID
  AND IsAvailable = TRUE;

-- 7. TOTAL HOURS AFTER ADDING NEW SLOTS
SET @TotalHours := @NewHours + @ExistingHours;


-- 8. OVERLAP CHECK: NEW SLOTS AMONG THEMSELVES
SELECT EXISTS (
    SELECT 1
    FROM TempSlotsCopy A
    JOIN TempSlotsCopy B
      ON A.DayOfWeek = B.DayOfWeek
     AND A.StartTime < B.EndTime
     AND B.StartTime < A.EndTime
     AND A.SlotID < B.SlotID
) INTO @OverlapNew;


-- 9. OVERLAP CHECK: NEW SLOTS VS EXISTING AVAILABILITY (SHOULD BE NONE FOR
NEW TUTOR BUT INCLUDED)
SELECT EXISTS (
    SELECT 1
    FROM TempNewSlots n
    JOIN Availability a
      ON a.TutorID = @TutorID
     AND a.IsAvailable = TRUE
     AND n.DayOfWeek = a.DayOfWeek
```

```
    AND n.StartTime < a.EndTime
    AND a.StartTime < n.EndTime
) INTO @OverlapExisting;


-- 10. VALIDATION LOGIC
SET @IsValid := (
  CASE
    WHEN @TotalHours <= @MaxHoursPerWeek
     AND @OverlapNew = 0
     AND @OverlapExisting = 0
    THEN 1
    ELSE 0
  END
);


-- 11. CONDITIONAL INSERT OF AVAILABILITY (ATOMIC)
START TRANSACTION;

-- For a brand-new tutor this DELETE does nothing, but keeps logic consistent.
DELETE FROM Availability
WHERE TutorID = @TutorID
  AND @IsValid = 1;

INSERT INTO Availability (TutorID, DayOfWeek, StartTime, EndTime, IsAvailable)
SELECT @TutorID, DayOfWeek, StartTime, EndTime, TRUE
FROM TempNewSlots
```

```
WHERE @IsValid = 1;


COMMIT;



-- 12. STATUS MESSAGE
SELECT
  CASE
    WHEN @IsValid = 1 THEN
      'SUCCESS: New tutor added with initial availability.'
    WHEN @OverlapNew = 1 THEN
      'ERROR: New availability time slots overlap each other.'
    WHEN @OverlapExisting = 1 THEN
      'ERROR: New availability time slots overlap existing schedule.'
    WHEN @TotalHours > @MaxHoursPerWeek THEN
      CONCAT('ERROR: Total hours (', @TotalHours,
          ') exceed weekly limit of ', @MaxHoursPerWeek, '.')
    ELSE
      'ERROR: Unknown validation failure while adding tutor.'
  END AS Message;



-- 13. VERIFICATION: SHOW NEW TUTOR, THEIR SUBJECTS, AND AVAILABILITY (ONLY IF
VALID)
SELECT
  t.TutorID,
  CONCAT(t.FirstName, ' ', t.LastName) AS Tutor,
  s.SubjectName,
  a.DayOfWeek,
```

```
    a.StartTime,
    a.EndTime,
    TIMESTAMPDIFF(MINUTE, a.StartTime, a.EndTime) / 60.0 AS Hours
FROM Tutor t
LEFT JOIN TutorSubject ts ON t.TutorID = ts.TutorID
LEFT JOIN Subject s ON ts.SubjectID = s.SubjectID
LEFT JOIN Availability a ON t.TutorID = a.TutorID
WHERE t.TutorID = @TutorID
  AND @IsValid = 1
ORDER BY FIELD(a.DayOfWeek,
    'Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'),
    a.StartTime;



-- 14. CLEANUP
DROP TABLE TempSlotsCopy;
-- (TempNewSlots is TEMPORARY and will be dropped automatically at end of session)
```

User story #3:

```sql
-- We are assuming we are in the past to work with the
-- dummy data we currently have.

SET @SubjectName := 'Intro to Programming';

SELECT
    s.SubjectName                                AS `Course`,
    s.Department                                 AS `Department`,
    CONCAT(t.FirstName, ' ', t.LastName)         AS `Tutor`,
    DATE_FORMAT(a.AppointmentDate, '%W')         AS `Day`,
    DATE_FORMAT(a.AppointmentDate, '%Y-%m-%d')   AS `Date`,
    DATE_FORMAT(a.AppointmentDate, '%h:%i %p')   AS `Starts`,
    r.RoomName                                   AS `Location`,
    a.Status                                     AS `Status`
FROM Subject     s
JOIN Appointment a  ON a.SubjectID = s.SubjectID
JOIN Tutor       t  ON t.TutorID   = a.TutorID
LEFT JOIN Room   r  ON r.RoomID    = a.RoomID
WHERE
    s.SubjectName = @SubjectName
    AND a.Status = 'scheduled'
    AND a.AppointmentDate >= '2025-01-01'
ORDER BY
    a.AppointmentDate;
```

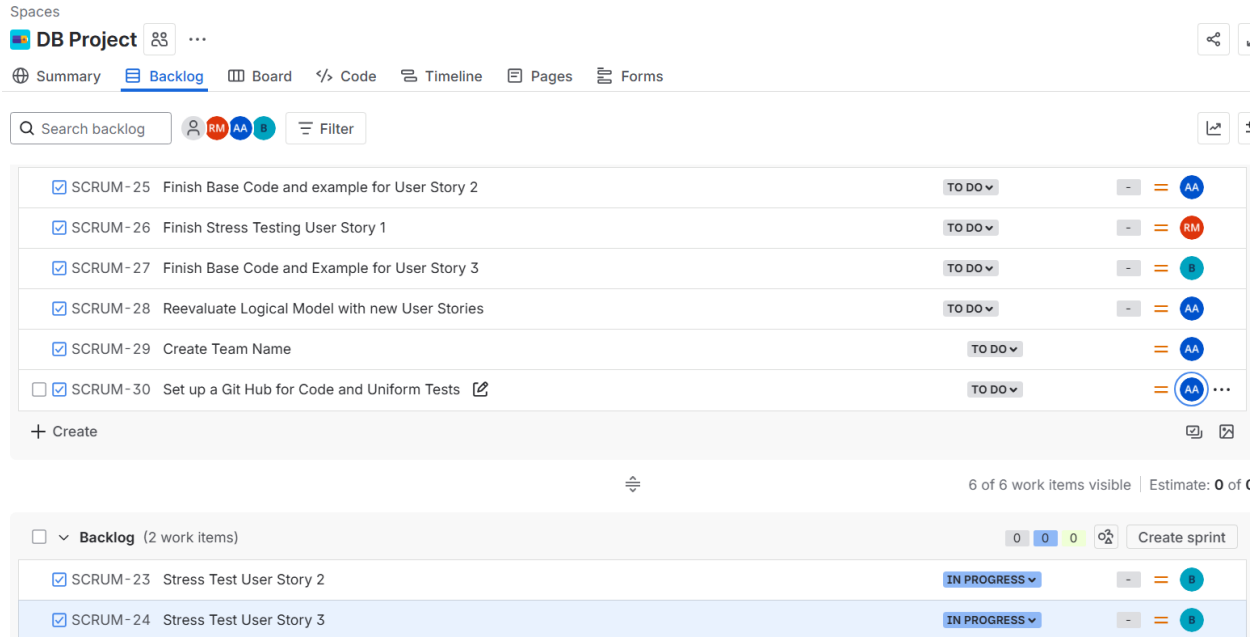
Sprint planning meeting summary (Adam):

Week 4:

- Everyone will contribute to user stories
- Make a repo to keep all our files clean and in order
- Stress test all of the user stories
- Update the table/er graph if needed
- Make a team name

Screenshot of Jira page for sprint that was just completed (Reese):

All Scrums from 25 – 30 are done, they are not marked done because they disappear.



Screenshot of Jira page for upcoming sprint (Reese):