**A close-up of a logo

Description automatically generated**

**IST 659 Data Admin Concepts and Database Management Project**

**Swim Lessons Database Management Project**

**Group 3 Members**

**Akshay Joshi**

**Janelle**

**Goutham Sri Vishwesh Bikkumalla**

**Akash**

TABLE OF CONTENTS

|  |  |
| --- | --- |
| INDEX | PAGE NO |
| Introduction | 3 |
| Objective | 4 |
| ER Data Requirement | 5 |
| Conceptual Data Model | 6 |
| Logical Data Model | 7 |
| SQL Code | 8-17 |
| Business Model | 18 |
| Frontend- Power Apps | 19-21 |
| Conclusion and Limitations | 22 |
| Team Log | 23 |

**Introduction:**

The Swim Lesson Management System is a comprehensive solution designed to streamline and enhance the administration of swim lesson programs. This system caters to the distinct needs of both Instructors and Parents, offering a user-friendly platform that ensures the efficient organization and execution of swim lessons. Through this system, users can easily view and manage lesson schedules, gaining access to detailed class information, timing specifics, and participant details. This feature facilitates a seamless coordination of lesson logistics, contributing to the overall effectiveness of the swim program.

One of the key functionalities of the system is its capability to monitor and track the progress of each participant. Instructors can record and evaluate individual performances, allowing for a thorough assessment of skill development. Parents, in turn, benefit from the ability to monitor the progress of their children through a user-friendly interface. This transparency fosters communication and engagement between Instructors, Parents, and Participants, creating a collaborative learning environment.

Moreover, the Swim Lesson Management System goes beyond basic scheduling and progress tracking. It encourages communication and feedback exchange by providing a platform for sharing updates, personalized recommendations, and constructive feedback. This feature enhances the overall learning experience, creating a supportive ecosystem that promotes continuous improvement and ensures that each participant receives tailored guidance for optimal skill development. In essence, the Swim Lesson Management System is poised to revolutionize the way swim lesson programs are conducted, fostering efficiency, transparency, and collaboration among Instructors, Parents, and Participants.

**Objectives:**

The Swim Lesson Management System project is driven by a set of primary objectives that collectively aim to revolutionize the way data is collected, managed, and accessed within the swim lesson programs. The foremost objective is to streamline the process of data collection, ensuring that information related to lesson schedules, class details, timing, participant progress, and feedback is efficiently gathered and organized. By centralizing this data, the system contributes to a more systematic and coherent approach to swim lesson management.

Enhancing information security is another pivotal objective of the project. The Swim Lesson Management System is designed to implement robust security measures, safeguarding sensitive participant data, progress records, and other critical information. This focus on information security not only protects the privacy of individuals but also builds trust among parents and participants in the reliability of the system.

Improving data accessibility is a key goal to empower both Instructors and Parents. The system is structured to provide convenient and secure access to relevant information, allowing Instructors to efficiently manage lesson schedules and evaluate participant progress while enabling Parents to monitor the development of their children. This increased accessibility ensures that stakeholders have the information they need at their fingertips, contributing to a more informed and engaged swim lesson community.

Additionally, a critical objective is to ensure compliance with data protection regulations. Given the sensitive nature of participant information, the Swim Lesson Management System adheres to data protection standards, safeguarding against potential legal and ethical implications. This commitment to compliance not only reflects responsible data management but also establishes the project as a trustworthy and ethical solution.

By achieving these objectives, the project aims to bring about significant improvements in customer service, internal operations, and the overall efficiency and security of the swim lesson programs. Furthermore, the implementation of these objectives will help mitigate data-related risks, fostering a more resilient and reliable environment for managing swim lessons. In essence, the Swim Lesson Management System seeks to set a new standard for data-driven efficiency, security, and compliance in the realm of swim lesson program administration.

**E-R Data Requirements:**

An Entity-Relationship (ER) model is essential for defining the data needs and creating relationships between various entities while designing the database for the Swim Lesson Management System. We'll define entities for Student, Parent, Instructor, Group, and Level in this context, along with the properties that correspond to each.

A screen shot of a white sheet

Description automatically generated

* A Many-to-One relationship exists between the Student and Parent entities, where each student is associated with one parent or guardian. This relationship is established through the foreign key student parent id in the Student entity.
* Another Many-to-One relationship exists between the Student and Group entities, signifying that each student is part of a specific group. This relationship is established through the foreign key group id in the Student entity.
* The Group entity has a Many-to-One relationship with the Level entity, indicating that each group is associated with a specific skill level. This relationship is established through the foreign key level id in the Group entity.

**Conceptual Model:**

The conceptual model of the Swim Lesson Management System serves as the foundational blueprint for understanding the structure and relationships between various entities within the system. The primary entities in the conceptual model are Student, Parent, Instructor, Group, and Level. These entities encapsulate the key components of the swim lesson program, allowing for a comprehensive representation of the data requirements and interactions within the system.

A diagram of a program

Description automatically generated

This conceptual model provides a clear representation of the key entities, their attributes, and the relationships that connect them. It serves as a high-level guide for designing the database schema and lays the groundwork for implementing a robust Swim Lesson Management System that effectively captures, organizes, and manages the essential information related to swim lessons, participants, and instructors. The conceptual model forms the basis for creating a structured and efficient database that aligns with the project's overarching objectives of streamlined data management, enhanced security, and improved accessibility.

**Logical Data Model:**

The logical model of the Swim Lesson Management System represents the detailed structure of the database, including tables, fields, relationships, and constraints. It is an abstraction that translates the conceptual model into a format that can be implemented in a relational database management system (RDBMS). In the context of this project, the logical model defines the schema for the database and specifies how data will be stored, organized, and related.

**A screenshot of a computer

Description automatically generated**

The logical model provides a detailed representation of how data will be organized in the database, defining tables, fields, constraints, and relationships. It serves as a crucial step in the database design process, transitioning from the high-level conceptual model to a format that can be implemented and utilized by a relational database management system.

**SQL Code: -**

IF EXISTS (SELECT \* from INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS

where

constraint\_name = 'fk\_students\_parent\_id')

alter table patients DROP

CONSTRAINT fk\_students\_parent\_id

IF EXISTS (SELECT \* from INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS

where

constraint\_name = 'fk\_students\_instructor\_id')

alter table patients DROP

CONSTRAINT fk\_students\_instructor\_id

IF EXISTS (SELECT \* from INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS

where

constraint\_name = 'fk\_students\_group\_id')

alter table patients DROP

CONSTRAINT fk\_students\_group\_id

IF EXISTS (SELECT \* from INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS

where

constraint\_name = 'fk\_groups\_level\_id')

alter table patients DROP

CONSTRAINT fk\_groups\_level\_id

IF EXISTS (SELECT \* from INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS

where

constraint\_name = 'fk\_groups\_instructor\_id')

alter table patients DROP

CONSTRAINT fk\_groups\_instructor\_id

GO

DROP TABLE IF EXISTS students

DROP TABLE IF EXISTS parents

DROP TABLE IF EXISTS instructors

DROP TABLE IF EXISTS groups

DROP TABLE IF EXISTS levels

GO

-- Creating tables

-- Students table

CREATE TABLE students (

  student\_id INT PRIMARY KEY,

  first\_name VARCHAR(30) NOT NULL,

  last\_name VARCHAR(30) NOT NULL,

  age INT NOT NULL,

  phone\_number CHAR(10),

  email VARCHAR(50),

  StreetAddress VARCHAR(255),

  City VARCHAR(100),

  StateOrRegion VARCHAR(50),

  PostalCode VARCHAR(20),

  Country VARCHAR(50),

  student\_parent\_id INT NOT NULL,

  student\_group\_id INT NOT NULL,

  instructor\_id INT NOT NULL

);

-- Parents table

CREATE TABLE parents (

  parent\_id INT PRIMARY KEY,

  first\_name VARCHAR(30) NOT NULL,

  last\_name VARCHAR(30) NOT NULL,

  phone\_number CHAR(10),

  email VARCHAR(50),

  StreetAddress VARCHAR(255),

  City VARCHAR(100),

  StateOrRegion VARCHAR(50),

  PostalCode VARCHAR(20),

  Country VARCHAR(50),

  UNIQUE (phone\_number, email)

);

-- Instructors table

CREATE TABLE instructors (

  instructor\_id INT PRIMARY KEY,

  first\_name VARCHAR(30) NOT NULL,

  last\_name VARCHAR(30) NOT NULL,

  phone\_number CHAR(10),

  email VARCHAR(50),

  UNIQUE (phone\_number, email)

);

-- Groups table

CREATE TABLE groups (

  group\_id INT PRIMARY KEY,

  level\_id INT NOT NULL,

  total\_students INT NOT NULL,

  instructor\_id INT NOT NULL,

  meeting\_day VARCHAR(20) NOT NULL,

  meeting\_time TIME NOT NULL

);

-- Levels (Lookup Table)

CREATE TABLE levels (

  level\_id INT PRIMARY KEY,

  level\_name VARCHAR(30) NOT NULL

);

-- Creating relationships

-- Student-Parent relationship

ALTER TABLE students

ADD CONSTRAINT fk\_students\_parent\_id FOREIGN KEY (student\_parent\_id)

REFERENCES parents(parent\_id);

-- Student-Instructor relationship

ALTER TABLE students

ADD CONSTRAINT fk\_students\_instructor\_id FOREIGN KEY (instructor\_id)

REFERENCES instructors(instructor\_id);

-- Student-Group relationship

ALTER TABLE students

ADD CONSTRAINT fk\_students\_group\_id FOREIGN KEY (student\_group\_id)

REFERENCES groups(group\_id);

-- Group-Level relationship

ALTER TABLE groups

ADD CONSTRAINT fk\_groups\_level\_id FOREIGN KEY (level\_id)

REFERENCES levels(level\_id);

-- Instructor-Group relationship

ALTER TABLE groups

ADD CONSTRAINT fk\_groups\_instructor\_id FOREIGN KEY (instructor\_id)

REFERENCES instructors(instructor\_id);

-- inserting values

go

INSERT INTO students(student\_id,first\_name,last\_name,age,phone\_number,email,streetAddress,city,StateOrRegion,postalcode,country,student\_parent\_id,student\_group\_id,instructor\_id) VALUES

(1, 'John', 'Doe', 20, '1234567890', 'john.doe@example.com', '456 Oak St', 'Anytown', 'CA', '12345', 'USA', 101, 401, 301),

(2, 'Jane', 'Smith', 22, '9876543210', 'jane.smith@example.com', '789 Maple Ave', 'Othertown', 'NY', '54321', 'USA', 102, 402, 302),

(3, 'Bob', 'Johnson', 19, '5551234567', 'bob.johnson@example.com', '321 Pine Ln', 'Smalltown', 'TX', '67890', 'USA', 103, 403, 303),

(4, 'Emily', 'Williams', 21, '4447890123', 'emily.williams@example.com', '987 Birch Rd', 'Newtown', 'FL', '98765', 'USA', 104, 404, 304),

(5, 'Alex', 'Davis', 23, '2223334444', 'alex.davis@example.com', '654 Cedar Blvd', 'Sometown', 'WA', '54321', 'USA', 105, 405, 305),

(6, 'Megan', 'Brown', 20, '1112223333', 'megan.brown@example.com', '789 Elm St', 'Cityville', 'IL', '87654', 'USA', 106, 406, 306),

(7, 'Chris', 'Jones', 22, '9998887777', 'chris.jones@example.com', '234 Oak Ave', 'Metroville', 'CA', '34567', 'USA', 107, 407, 307),

(8, 'Amanda', 'Miller', 19, '7778889999', 'amanda.miller@example.com', '876 Pine Rd', 'Villagetown', 'NY', '23456', 'USA', 108, 408, 308),

(9, 'Daniel', 'Clark', 21, '3334445555', 'daniel.clark@example.com', '543 Maple Ln', 'Hometown', 'TX', '76543', 'USA', 109, 409, 309),

(10, 'Sophia', 'Lee', 20, '6667778888', 'sophia.lee@example.com', '123 Cedar Ave', 'Suburbia', 'FL', '43210', 'USA', 110, 410, 310);

GO

INSERT INTO parents(parent\_id,first\_name,last\_name,phone\_number,email,streetAddress,city,StateOrRegion,postalcode,country) VALUES

(101, 'Alice', 'Doe', '1112223333', 'alice.doe@example.com', '123 Pine St', 'Anytown', 'CA', '12345', 'USA'),

(102, 'Bob', 'Smith', '2223334444', 'bob.smith@example.com', '456 Oak Ave', 'Othertown', 'NY', '54321', 'USA'),

(103, 'Charlie', 'Johnson', '3334445555', 'charlie.johnson@example.com', '789 Maple Ln', 'Smalltown', 'TX', '67890', 'USA'),

(104, 'David', 'Williams', '4445556666', 'david.williams@example.com', '987 Cedar Blvd', 'Newtown', 'FL', '98765', 'USA'),

(105, 'Eva', 'Davis', '5556667777', 'eva.davis@example.com', '654 Birch Rd', 'Sometown', 'WA', '54321', 'USA'),

(106, 'Frank', 'Brown', '6667778888', 'frank.brown@example.com', '321 Elm St', 'Cityville', 'IL', '87654', 'USA'),

(107, 'Grace', 'Jones', '7778889999', 'grace.jones@example.com', '876 Pine Ave', 'Metroville', 'CA', '34567', 'USA'),

(108, 'Harry', 'Miller', '8889990000', 'harry.miller@example.com', '234 Oak Ln', 'Villagetown', 'NY', '23456', 'USA'),

(109, 'Ivy', 'Clark', '9990001111', 'ivy.clark@example.com', '543 Cedar Rd', 'Hometown', 'TX', '76543', 'USA'),

(110, 'Jack', 'Lee', '1234567890', 'jack.lee@example.com', '789 Maple Blvd', 'Suburbia', 'FL', '43210', 'USA');

GO

INSERT INTO instructors(instructor\_id,first\_name,last\_name,phone\_number,email)VALUES

(301, 'Michael', 'Smith', '1112223333', 'michael.smith@example.com'),

(302, 'Jennifer', 'Johnson', '2223334444', 'jennifer.johnson@example.com'),

(303, 'Matthew', 'Williams', '3334445555', 'matthew.williams@example.com'),

(304, 'Emily', 'Davis', '4445556666', 'emily.davis@example.com'),

(305, 'Andrew', 'Brown', '5556667777', 'andrew.brown@example.com'),

(306, 'Olivia', 'Jones', '6667778888', 'olivia.jones@example.com'),

(307, 'Daniel', 'Miller', '7778889999', 'daniel.miller@example.com'),

(308, 'Sophia', 'Clark', '8889990000', 'sophia.clark@example.com'),

(309, 'James', 'Lee', '9990001111', 'james.lee@example.com'),

(310, 'Emma', 'Smith', '1234567890', 'emma.smith@example.com');

GO

INSERT INTO groups(group\_id,level\_id,total\_students,instructor\_id,meeting\_day,meeting\_time) VALUES

(401, 501, 20, 301, 'Monday', '10:00 AM'),

(402, 502, 18, 302, 'Tuesday', '2:00 PM'),

(403, 503, 22, 303, 'Wednesday', '4:30 PM'),

(404, 504, 15, 304, 'Thursday', '1:00 PM'),

(405, 505, 25, 305, 'Friday', '3:45 PM'),

(406, 506, 21, 306, 'Saturday', '9:30 AM'),

(407, 507, 19, 307, 'Sunday', '11:15 AM'),

(408, 508, 23, 308, 'Monday', '8:00 AM'),

(409, 509, 17, 309, 'Tuesday', '12:45 PM'),

(410, 510, 24, 310, 'Wednesday', '6:15 PM');

GO

INSERT INTO levels(level\_id,level\_name)VALUES

(501, 'Beginner'),

(502, 'Intermediate'),

(503, 'Advanced'),

(504, 'Expert'),

(505, 'Intermediate'),

(506, 'Beginner'),

(507, 'Advanced'),

(508, 'Expert'),

(509, 'Beginner'),

(510, 'Intermediate');

-- View for instructors

CREATE VIEW instructor\_view AS

SELECT

    i.instructor\_id,

    i.first\_name AS instructor\_first\_name,

    i.last\_name AS instructor\_last\_name,

    i.phone\_number AS instructor\_phone\_number,

    i.email AS instructor\_email,

    g.group\_id,

    g.meeting\_day,

    g.meeting\_time,

    l.level\_name

FROM

    instructors i

JOIN

    groups g ON i.instructor\_id = g.instructor\_id

JOIN

    levels l ON g.level\_id = l.level\_id;

select \* from instructor\_view

CREATE VIEW parent\_view AS

SELECT

    s.student\_id,

    s.first\_name + ' ' + s.last\_name as student\_name,

    s.age as student\_age,

    s.phone\_number as student\_phone\_number,

    s.email as student\_email,

    s.StreetAddress,

    s.StateOrRegion,

    s.PostalCode,

    s.country,

    i.first\_name + ' ' + i.last\_name as instructor\_name,

    i.phone\_number as instructor\_phone\_number,

    i.email as instructor\_email,

    g.group\_id,

    l.level\_name,

    CONVERT(VARCHAR(10), g.meeting\_day) + ' ' + CONVERT(VARCHAR(8), g.meeting\_time, 108) as weekly\_group\_meeting

FROM students s

JOIN parents p ON s.student\_parent\_id = p.parent\_id

JOIN instructors i ON s.instructor\_id = i.instructor\_id

JOIN groups g ON g.group\_id = s.student\_group\_id

JOIN levels l ON g.level\_id = l.level\_id;

GO

select \* from parent\_view

-- drop procedure if exists p\_upsert\_parent\_details

-- GO

-- create procedure p\_upsert\_parent\_details (

--     @student\_email varchar(50),

--     @student\_phonenumber char(3),

--     @student\_id VARCHAR(20)

-- ) as begin

--     if exists(select \* from parent\_view where student\_email =@parent\_email ) begin

--         update parent\_view set student\_phone\_number =   @student\_phonenumber, student\_id = @student\_id

--             where student\_id = @student\_id

--     end

--     else begin

--         insert into parent\_view  (student\_id,student\_email, student\_phone\_number)

--             values (@student\_id@student\_email,@student\_phonenumber )

--     end

-- end

-- GO

CREATE VIEW student\_view AS

SELECT

    s.student\_id,

    s.first\_name AS student\_first\_name,

    s.last\_name AS student\_last\_name,

    s.age AS student\_age,

    s.phone\_number AS student\_phone\_number,

    s.email AS student\_email,

    s.StreetAddress AS student\_StreetAddress,

    s.City AS student\_City,

    s.StateOrRegion AS student\_StateOrRegion,

    s.PostalCode AS student\_PostalCode,

    s.Country AS student\_Country,

    p.parent\_id,

    p.first\_name AS parent\_first\_name,

    p.last\_name AS parent\_last\_name,

    p.phone\_number AS parent\_phone\_number,

    p.email AS parent\_email,

    i.instructor\_id,

    i.first\_name AS instructor\_first\_name,

    i.last\_name AS instructor\_last\_name,

    g.group\_id,

    g.level\_id,

    g.total\_students,

    g.meeting\_day,

    g.meeting\_time,

    l.level\_name

FROM

    students s

JOIN

    parents p ON s.student\_parent\_id = p.parent\_id

JOIN

    instructors i ON s.instructor\_id = i.instructor\_id

JOIN

    groups g ON s.student\_group\_id = g.group\_id

JOIN

    levels l ON g.level\_id = l.level\_id;

SELECT \* FROM student\_view;

select \* from parent\_view

-- Declare variables

DECLARE @student\_email VARCHAR(50) = 'john.doe@example.com';

DECLARE @student\_phonenumber CHAR(10) = '1234567890';

DECLARE @student\_id VARCHAR(20) = '1';

-- Execute the stored procedure

EXEC p\_upsert\_parent\_details @student\_email, @student\_phonenumber, @student\_id;

DROP PROCEDURE IF EXISTS p\_upsert\_parent\_details;

GO

CREATE PROCEDURE p\_upsert\_parent\_details (

    @student\_email VARCHAR(50),

    @student\_phonenumber CHAR(10),

    @student\_id VARCHAR(20)

) AS

BEGIN

    IF EXISTS (SELECT \* FROM parent\_view WHERE student\_email = @student\_email)

    BEGIN

        UPDATE parent\_view

        SET student\_phone\_number = @student\_phonenumber

        WHERE student\_id = @student\_id;

    END

    ELSE

    BEGIN

        INSERT INTO parent\_view (student\_id, student\_email, student\_phone\_number)

        VALUES (@student\_id, @student\_email, @student\_phonenumber);

    END

END;

GO

--List of students in a group

SELECT s.student\_id, s.first\_name, s.last\_name, s.age, s.phone\_number, s.email

FROM students s

WHERE s.student\_group\_id = 401;

--parent information for a students

SELECT p.parent\_id, p.first\_name, p.last\_name, p.phone\_number, p.email

FROM parents p

JOIN students s ON p.parent\_id = s.student\_parent\_id

WHERE s.student\_id = student\_id;

--List of group with instructor Details

SELECT g.group\_id, g.meeting\_day, g.meeting\_time, i.instructor\_id, i.first\_name AS instructor\_first\_name, i.last\_name AS instructor\_last\_name

FROM groups g

JOIN instructors i ON g.instructor\_id = i.instructor\_id;

--Total Students per Level

SELECT l.level\_name, COUNT(s.student\_id) AS total\_students

FROM levels l

JOIN groups g ON l.level\_id = g.level\_id

JOIN students s ON g.group\_id = s.student\_group\_id

GROUP BY l.level\_name;

-- --update Students Details

-- UPDATE students

-- SET age = <new\_age>, phone\_number = '<new\_phone\_number>', email = '<new\_email>'

-- WHERE student\_id = <student\_id>;

-- --Delete a Group

-- DELETE FROM groups

-- WHERE group\_id = <group\_id>;

--Student Information Retrieval

SELECT \* FROM students WHERE student\_id = student\_id;

--Parent Contact Information/--list of parents in a group

SELECT p.parent\_id, p.first\_name, p.last\_name, p.phone\_number, p.email

FROM parents p

JOIN students s ON p.parent\_id = s.student\_parent\_id

WHERE s.student\_group\_id = student\_group\_id;

--Group Schedule

SELECT g.group\_id, g.meeting\_day, g.meeting\_time

FROM groups g

WHERE g.group\_id = group\_id;

-- List of Instructors and Assigned Groups

SELECT i.instructor\_id, i.first\_name, i.last\_name, g.group\_id

FROM instructors i

JOIN groups g ON i.instructor\_id = g.instructor\_id;

-- Parental Contact Duplication Check

SELECT phone\_number, email, COUNT(\*)

FROM parents

GROUP BY phone\_number, email

HAVING COUNT(\*) > 1;

--Students without Assigned Parents

SELECT s.student\_id, s.first\_name, s.last\_name

FROM students s

LEFT JOIN parents p ON s.student\_parent\_id = p.parent\_id

WHERE p.parent\_id IS NULL;

--Meeting Conflict Check

SELECT g.group\_id, g.meeting\_day, g.meeting\_time, COUNT(\*)

FROM groups g

GROUP BY g.group\_id, g.meeting\_day, g.meeting\_time

HAVING COUNT(\*) > 1;

--group summary

CREATE VIEW GroupSummary AS

SELECT

  g.group\_id, g.level\_id, l.level\_name, g.total\_students,

  g.instructor\_id, i.first\_name AS instructor\_first\_name, i.last\_name AS instructor\_last\_name,

  g.meeting\_day, g.meeting\_time

FROM

  groups g

JOIN levels l ON g.level\_id = l.level\_id

JOIN instructors i ON g.instructor\_id = i.instructor\_id;

SELECT \* from GroupSummary

CREATE PROCEDURE EditInstructorDetails(

    @instructor\_id INT,

    @new\_meeting\_day VARCHAR(20) = NULL,

    @new\_meeting\_time TIME = NULL,

    @new\_email VARCHAR(50) = NULL,

    @new\_level\_name VARCHAR(30) = NULL

)

AS

BEGIN

    -- Update meeting schedule if provided

    IF @new\_meeting\_day IS NOT NULL OR @new\_meeting\_time IS NOT NULL

    BEGIN

        UPDATE groups

        SET

            meeting\_day = ISNULL(@new\_meeting\_day, meeting\_day),

            meeting\_time = ISNULL(@new\_meeting\_time, meeting\_time)

        WHERE instructor\_id = @instructor\_id;

    END;

    -- Update email if provided

    IF @new\_email IS NOT NULL

    BEGIN

        UPDATE instructors

        SET email = @new\_email

        WHERE instructor\_id = @instructor\_id;

    END;

    -- Update level name if provided

    IF @new\_level\_name IS NOT NULL

    BEGIN

        UPDATE levels

        SET level\_name = @new\_level\_name

        WHERE level\_id = (SELECT level\_id FROM instructor\_view WHERE instructor\_id = @instructor\_id);

    END;

END;

-- Edit meeting schedule for instructor with ID 301

EXEC EditInstructorDetails @instructor\_id = 301, @new\_meeting\_day = 'Friday', @new\_meeting\_time = '2:30 PM';

select \* from instructor\_view

-- Edit email for instructor with ID 302

EXEC EditInstructorDetails @instructor\_id = 302, @new\_email = 'new\_email@example.com';

-- Edit level name for instructor with ID 303

EXEC EditInstructorDetails @instructor\_id = 303, @new\_level\_name = 'Advanced';

--it will show the phone number is different or same

SELECT

    s.student\_id,

    s.first\_name AS student\_first\_name,

    s.last\_name AS student\_last\_name,

    s.phone\_number AS student\_phone\_number,

    i.instructor\_id,

    i.first\_name AS instructor\_first\_name,

    i.last\_name AS instructor\_last\_name,

    i.phone\_number AS instructor\_phone\_number,

    CASE

        WHEN s.phone\_number = i.phone\_number THEN 'Same'

        ELSE 'Different'

    END AS phone\_numbers\_comparison

FROM

    students s

JOIN

    instructors i ON s.instructor\_id = i.instructor\_id;

**Business Model:**

The Swim Lesson Management System project is built upon a business model that focuses on creating value for both the swim company, the provider of swim lessons, and its customers, primarily the participants and their families. The business model is centered around three key pillars: efficient data management, improved customer service, and a commitment to data security and compliance.

* Efficient Data Management:

The project addresses the need for centralized and efficient data management within the swim company. By offering a comprehensive Swim Lesson Management System, the project eliminates the inefficiencies associated with disparate methods of collecting, storing, and managing swim lesson data. This results in a more streamlined and organized approach, reducing the likelihood of errors and enhancing overall operational efficiency.

* Improved Customer Service:

Accurate and real-time information is a cornerstone of improved customer service. The project ensures that the swim company can respond promptly to inquiries, track participant progress, and provide a personalized experience to participants and their families. The availability of up-to-date data allows for better communication and engagement, fostering a positive relationship between the swim company and its customers.

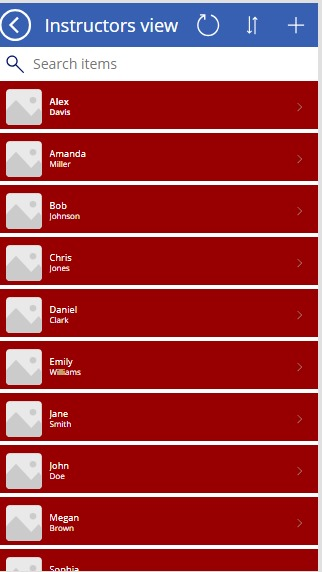
* Data Security and Compliance:

The Swim Lesson Management System places a strong emphasis on data security and compliance with data protection regulations. This commitment helps the swim company mitigate legal and reputational risks associated with mishandling sensitive participant information. By adhering to established data protection standards, the system builds trust with customers and reinforces the swim company's commitment to responsible and ethical data management practices.

The business model for the Swim Lesson Management System project revolves around creating value through efficient data management, improved customer service, and a strong focus on data security and compliance. This model not only enhances the operational effectiveness of the swim company but also enriches the experience for swim lesson participants, establishing a foundation for long-term customer satisfaction and trust.

**A screenshot of a phone

Description automatically generatedFrontend – Implementation Screens**

A person swimming in a pool

Description automatically generated A screenshot of a red box

Description automatically generated

**Conclusion and Limitations:**

In conclusion, the Swim Lesson Management System project represents a significant step forward in addressing the complexities associated with managing swim lesson data. By introducing a comprehensive solution that streamlines data collection, enhances information security, and improves accessibility, the project is positioned to revolutionize the management of swim lessons. The envisioned system not only promises increased efficiency for the swim company but also aims to elevate the overall experience for participants and their families. Through centralized data management and improved customer service, the project sets the stage for a more streamlined, responsive, and secure swim lesson program.

However, it is essential to acknowledge and address certain limitations and challenges that may impact the successful implementation and sustainability of the project:

* **Scalability Challenges:**

As the swim company grows or undergoes changes, scalability challenges may emerge. The system should be designed with scalability in mind to accommodate an increasing volume of data and users without compromising performance. This requires careful consideration of the architecture and infrastructure to ensure that the system can seamlessly adapt to the evolving needs of the swim company over time.

* **Budget and Resource Allocation:**

Adhering to the estimated budget and resource allocation is crucial for the success of the project. Striking a balance between project requirements and available resources is essential to avoid potential setbacks and ensure that the system is delivered within the specified constraints. Effective project management and continuous monitoring of resource utilization will be critical to meeting project objectives while staying within the designated budget.

By acknowledging these limitations and proactively addressing challenges, the project team can enhance the likelihood of a successful implementation and sustainable operation of the Swim Lesson Management System. Regular assessments and adjustments throughout the project lifecycle will contribute to its overall success and effectiveness in meeting the objectives set forth.

Team Log - Project Report Timeline:

**September 30th: Project Proposal Outline and Summary**

The team established a clear vision for the project and defined the value proposition it aimed to deliver to the swim company and its participants.

**November 15th: Data Analysis and Attributes List, Conceptual Data Model Diagram, and Logical Data Model Diagram**

Conducted thorough data analysis to identify the necessary attributes for the Swim Lesson Management System. This involved defining the data requirements for entities such as Student, Parent, Instructor, Group, and Level.team created both the conceptual and logical data model diagrams. The conceptual model outlined the high-level entities and their relationships, providing a visual representation of the project's data structure. The logical model then delved into the specifics of tables, fields, and relationships, preparing the groundwork for the database implementation.

**November 29th: User Interface Views**

By November 29th, the team shifted focus to the user interface (UI) design, creating views that would facilitate a user-friendly experience for both Instructors and Parents.

Janelle:

- Created the entity relationship (ER) diagram, which models the key entities, attributes, and relationships in the database based on the requirements.

- Also created the logical data model, which is a technical diagram translating the ER diagram into database schema with tables, columns, keys, etc.

Akshay:

- Developed the conceptual data model, which involves identifying the main entities, attributes, relationships and mapping the requirements at a high level.

- Contributed to the team's presentation, likely explaining the conceptual model or other aspects.

Goutham:

- Worked on writing the SQL code to create the database schema, insert data, write queries, etc to make the database operational.

- Assisted with the PowerApp, which could involve creating an application to interact with the database for CRUD operations.

Akash:

- Took the lead on the PowerApp to surface the database data and functionality for users to leverage.

- Also contributed to the team presentation, potentially demonstrating the PowerApp, conceptual model, or other elements.

In summary, the team collaborated across database modeling, SQL coding, PowerApp development, and presentation creation. Each member played to their strengths while also assisting others to collectively build out the full database management project deliverables.