universidade de aveiro

deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

UniversityAlma

Hugo Curto 107654

Nuno Leite Faria 112994

Base de Dados

P11G8

Professor regente: Carlos Manuel Azevedo Costa

2023/2024

Indíce

Introdução

Na cadeira de Base de Dados foi-nos proposto a realização de um projeto de um sistema funcional que tenha aplicação válida na vida real. Tendo como inspiração uma ideia de um projeto da cadeira de Interação Humano-Computador, decidimos fazer uma aplicação que permite frequentar e gerir cursos de meditação que têm diferentes objetivos, desde ajudar a relaxar a melhorar o foco. Neste relatório vai ser descrito brevemente acerca do nosso projeto e explica a função dos vários Procedures, Triggers, Views, UDF e indexes criados

Os entregáveis deste projeto são os ficheiros SQL "UniveristyAlmaDML", onde as tabelas, Standard Procedures, Triggers, Views, UDF e indexes estão identificados separadamente pela sua designação (por exemplo, as tabelas estão no ficheiro "UniversityAlmaDML_Tables", Views no ficheiro "UniversityAlmaDML_Views), e "UniversityAlma_DDL", onde contém os valores inseridos na tabela, e o diretório UniversityAlmaApp, onde contém a aplicação UniversityAlmaApp e o código desta para decorrer, realizado em C#.

Para ser possível testar a aplicação, deve aceder ao ficheiro "DatabaseHelper.cs" no diretório UniversityAlmaApp, e na variável "connectionString", mudar o que está presente em Server, Database, User Id e Password para as credenciais desejadas.

Fig. 1-Fragmento do código em DatabaseHelper.cs onde se encontra as credenciais a mudar

Análise de requerimentos

Para a aplicação ser considerada um sucesso, nós definimos os seguintes objetivos:

- O utilizador deve ser capaz de receber notificações, procurar e frequentar cursos, como também demonstrar que gosta de um curso
- Um utilizador pode ser um mentor ou um administrador. Enquanto um mentor pode criar cursos e modificar cursos que o próprio criou, apenas o administrador tem a autoridade de apagar qualquer curso ou sessão dentro do curso
- Novos utilizadores podem ser capazes de criar contas novas, mas o sistema deve reconhecer se a informação submetida, como o username ou o email, já existe no sistema.
- No âmbito de login, o sistema deve ser capaz de verificar se o username e a password estão corretas
- O sistema deve ser capaz de ajustar o número de sessões e cursos consoante mudanças são feitas

Entidades e Multiplicidade

- Todas as pessoas que usam a aplicação UniversityAlma são utilizadores, identificados a partir do seu UserID
- Utilizadores têm uma relação 1:1 com o perfil. O perfil contém toda a informação vital acerca do utilizador, com o nome, email, password, username, número de telefone, o sexo, aniversário, foto de perfil, Mail List e favoritos. O perfil de uma pessoa tem como chave primária ProfileId
- As notificações têm uma relação N:M com o perfil. É constituído pela chave primária NotificationID, e os atributos Username, Icon, Info e Título
- Admin e Mentor estão relacionadas com o User em 1:1 a partir de uma Restrição de Completude parcial. O admin é identificado pelo AdminId, e o mentor tem como atributos MentorId, Experience e Certificates, sendo estes dois últimos atributos multivalor. O admin tem relações N:M com o Profile, onde o Admin modera, e com mentor, onde N Admins podem verificar Mentores como também remover essa verificação
- Audits é uma entidade que tem como atributos Admin_id, Mentor_id, Course_id, User_id e Date. O Admin tem uma relação 1:1 com Audits
- Courses têm como atributos Course_Id, Title e Description. Um único Mentor é capaz de moderar múltiplos Courses
- Category é uma entidade que tem uma relação 1:1 obrigatória com Courses. É identificada pela sua chave primária Category_id e pelo atributo Type.
- Um Course contém múltiplas Sessions. Uma Session tem como atributos Courseid, Number, Title, Media, Duration e Fav Count

- History é uma entidade fraca, com atributos SessionNumber, ElapsedTime, ProfileId e CourseId, e tem uma relação 1:1 com Session

Diagrama Entidade-Relação (DER)

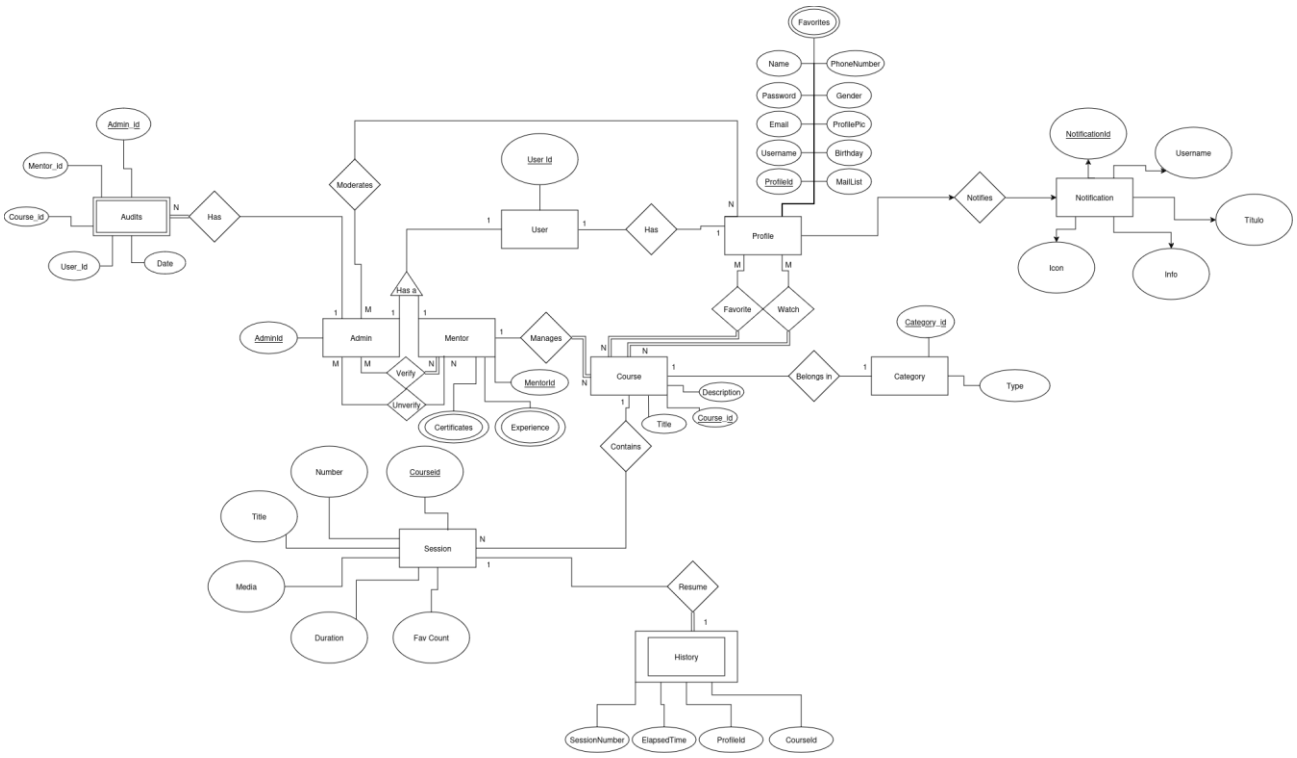

Fig. 2 - Diagrama ER

Esquema Relacional da BD

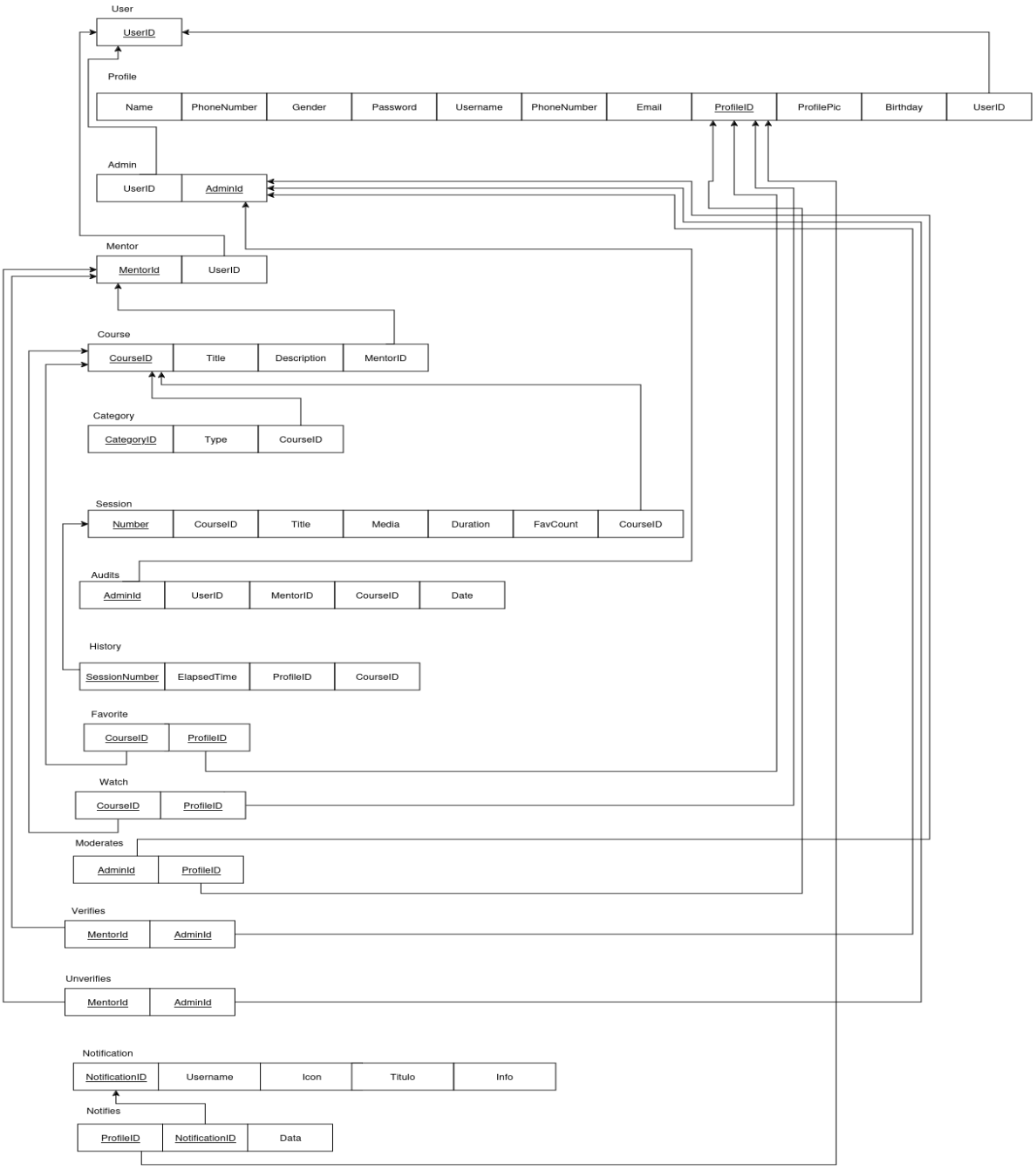

Fig. 3- Esquema Relacional

Stored Procedures (SP)

Resequenciação do número de sessões ao apagar

Numa aplicação em que o mentor e o admin têm a autoridade de criar ou apagar cursos, o mesmo se pode fazer com o número de sessões de um curso.

Considerando um exemplo, um curso tem 7 sessões. Se, por algum motivo, um utilizador com a autoridade decidir apagar a sessão número 3, será preciso reajustar os números que as sessões têm. Este processo permite a mudança simples do número das sessões afetadas

```
DROP PROCEDURE IF EXISTS UniversityAlma.ReSequenceSessionNumbers;
GO

CREATE PROCEDURE UniversityAlma.ReSequenceSessionNumbers
        @CourseId INT
AS
BEGIN
        SET NOCOUNT ON;

        CREATE TABLE #TempSession (
                NewNumber INT IDENTITY(1,1),
                SessionId INT
        );
        -- Insert existing sessions into temp table
        INSERT INTO #TempSession (SessionId)
        SELECT SessionId
        FROM UniversityAlma.Session
        WHERE CourseId = @CourseId
        ORDER BY SessionId;
        -- Update session numbers based on temp table
        UPDATE s
        SET s.Number = t.NewNumber
        FROM UniversityAlma.Session s
        JOIN #TempSession t ON s.SessionId = t.SessionId;
        -- Drop temp table
        DROP TABLE #TempSession;
END;
GO
```

Fig. 4-Código do Procedure ReSequenceSessionNumbers

Registar conta

Uma das funcionalidades base de cada aplicação é a capacidade de se poder registar uma conta. Antes de criar a conta, a Procedure verifica se existe algum utilizador na sua base de dados que tenham o mesmo Email ou o mesmo nome de utilizador que foi submetido no registo, e impede que a procedure conclua case exista. Se não existir conflito, a conta é criada, e a informação acerca do utilizador é posta na base de dados de perfis.

```
-- Register
CREATE PROCEDURE UniversityAlma.RegisterUser
	@Name VARCHAR(100),
	@Password VARCHAR(120),
	@Email VARCHAR(100),
	@Username VARCHAR(50),
	@PhoneNumber VARCHAR(15),
	@Gender VARCHAR(10),
	@ProfilePic VARBINARY(MAX) = NULL, --Default is null
	@Birthday DATE,
	@MailList BIT,
	@UserId INT OUTPUT,
	@ReturnCode INT OUTPUT
AS
BEGIN
	SET NOCOUNT ON;

	-- Check if username exists
	IF EXISTS (SELECT 1 FROM UniversityAlma.Profile WHERE Username = @Username)
	BEGIN
		SET @ReturnCode = -1;
		RETURN; -- Username exists
	END
	-- Check if email exists
	IF EXISTS (SELECT 1 FROM UniversityAlma.Profile WHERE Email = @Email)
	BEGIN
		SET @ReturnCode = -2;
		RETURN; -- Email exists
	END

	-- Insert the new profile
	INSERT INTO UniversityAlma.Profile(Name, Password, Email, Username, PhoneNumber, Gender, ProfilePic, Birthday, MailList)
	VALUES(@Name, @Password, @Email, @Username, @PhoneNumber, @Gender, @ProfilePic, @Birthday, @MailList);
	-- Retrieve the new profileId
	DECLARE @ProfileId INT;
	SET @ProfileId = SCOPE_IDENTITY();
	-- Retrieve the UserId based on this new ProfileId
	SELECT @UserId = UserId FROM UniversityAlma.[User] WHERE ProfileId = @ProfileId;
	-- Return the new userId
	SET @ReturnCode = 0; -- Registration successful
END;
GO
```

Fig. 5- Código do Procedure RegisterUser

Login

Acompanhando o procedure de registo, login é outra das funcionalidades base. Este código mostra o processo de login, onde verifica na tabela de perfis se existe algum cujo nome de utilizador e password coincidam o que foi utilizado

```
--Login
DROP PROCEDURE IF EXISTS UniversityAlma.LoginUser;
GO
CREATE PROCEDURE UniversityAlma.LoginUser
	@Username VARCHAR(50),
	@Password VARCHAR(120),
	@UserId INT OUTPUT,
	@ReturnCode INT OUTPUT
AS
BEGIN
	SET NOCOUNT ON

	SET @UserId= NULL;
	SET @ReturnCode = -1;
	-- Check if username and password match
	IF EXISTS (SELECT 1 FROM UniversityAlma.Profile WHERE Username = @Username AND Password = @Password)
	BEGIN
		-- Retrieve UserId
		SELECT @UserId = UserId
		FROM UniversityAlma.[User]
		WHERE ProfileId = (SELECT ProfileId FROM UniversityAlma.Profile WHERE Username = @Username);

		SET @ReturnCode = 0; -- Login successful
	END
	ELSE
	BEGIN
		-- Return null because the credentials are invalid
		SET @ReturnCode = -1; -- Login failed
	END
END;
GO
```

Fig. 6-Código do Procedure LoginUser

Adicionar e remover Favoritos

Um instrumento vital para avaliar a qualidade de um curso é um sistema de favoritos. Regularmente utilizado para demonstrar a qualidade do curso, também é uma forma de outros utilizadores serem capazes de obterem uma boa primeira impressão de um curso quando andam à procura.

```
-- Add favorite
DROP PROCEDURE IF EXISTS UniversityAlma.AddFavorite;
GO
CREATE PROCEDURE UniversityAlma.AddFavorite
	@ProfileId INT,
	@CourseId INT
AS
BEGIN
	SET NOCOUNT ON;

	IF NOT EXISTS (SELECT 1 FROM UniversityAlma.Favorites WHERE ProfileId = @ProfileId AND CourseId = @CourseId)
	BEGIN
		-- Insert favorite
		INSERT INTO UniversityAlma.Favorites(ProfileId, CourseId)
		VALUES (@ProfileId, @CourseId);
	END
END;
GO
-- Remove favorite
DROP PROCEDURE IF EXISTS UniversityAlma.RemoveFavorite;
GO
CREATE PROCEDURE UniversityAlma.RemoveFavorite
	@ProfileId INT,
	@CourseId INT
AS
BEGIN
	SET NOCOUNT ON

	-- Check if favorite exists
	IF EXISTS (SELECT 1 FROM UniversityAlma.Favorites WHERE ProfileId = @ProfileId AND CourseId = @CourseId)
	BEGIN
		-- Delete the favorite
		DELETE FROM UniversityAlma.Favorites
		WHERE ProfileId = @ProfileId AND CourseId = @CourseId;
	END
END;
GO
```

Fig. 7- Código dos Procedures AddFavorite e RemoveFavorite

Triggers

Notificação de auditoria

Este trigger tem como função notificar o utilizador acerca da auditoria em que está sujeito

```
DROP TRIGGER IF EXISTS TrigAuditNotification;
GO
-- Trigger for audit notification
CREATE TRIGGER TrigAuditNotification
ON UniversityAlma.Audits
AFTER INSERT
AS
BEGIN
        -- Insert notification for userId referenced by the audit
        INSERT INTO UniversityAlma.Notification(UserId, Title, Info, Icon)
        SELECT m.UserId, at.AuditTypeName, '', 'alert.png'
        FROM inserted i
        JOIN UniversityAlma.AuditTypes at ON i.AuditTypeId = at.AuditTypeId
        JOIN UniversityAlma.Course c ON i.CourseId = c.CourseId
        JOIN UniversityAlma.Mentor m ON c.MentorId = m.MentorId
        WHERE m.UserId IS NOT NULL;
END;
GO
```

Fig. 8- Código do Trigger TrigAuditNotification

Criar/Apagar utilizador ao Criar/Apagar perfil

Estes dois triggers têm a simples funcionalidade de associar um utilizador a um perfil quando um perfil é criado, e apagar o utilizador quando o perfil é apagado respetivamente

```
-- Trigger to insert/delete user on profile create/delete
CREATE TRIGGER trigInsertUser
ON UniversityAlma.Profile
AFTER INSERT
AS
BEGIN
        INSERT INTO UniversityAlma.[User] (UserId, ProfileId)
        SELECT ProfileId, ProfileId
        FROM inserted;
END;
GO

CREATE TRIGGER trigDeleteUser
ON UniversityAlma.Profile
AFTER DELETE
AS
BEGIN
        DELETE FROM UniversityAlma.[User]
        WHERE ProfileId IN (SELECT ProfileId FROM deleted);
END;
GO

--
```

Fig. 9- Código dos Triggers trigInsertUser e trigDeleteUser

Notificação de favorito de mentor

O propósito deste trigger é mandar uma notificação ao mentor quando um curso criado por ele é adicionado aos favoritos de um dos utilizadores. Este trigger vai buscar a informação do curso, do mentor, do número atual de favoritos que o curso tem, e verifica se o mentor tem alguma notificação associada a favoritos no curso por ver, e muda a mensagem que manda de acordo.

```
-- Favorite Notification for the Mentor trigger
CREATE TRIGGER trigFavoriteNotification
ON UniversityAlma.Favorites
AFTER INSERT
AS
BEGIN
        -- Variables to hold notification details
        DECLARE @CourseId INT;
        DECLARE @MentorId INT;
        DECLARE @UserId INT;
        DECLARE @CourseTitle VARCHAR(100);
        DECLARE @NotificationId INT;

        -- Loop trough the inserted favorites for a specific course
        DECLARE cur CURSOR FOR
        SELECT CourseId
        FROM inserted;

        OPEN cur;
        FETCH NEXT FROM cur INTO @CourseId;
        WHILE @@FETCH_STATUS = 0
        BEGIN
                -- Get the mentorId and courseId for the favorited course
                SELECT @MentorId = MentorId, @CourseTitle = Title
                FROM UniversityAlma.Course
                WHERE CourseId = @CourseId;
                -- Get the userId associated to the mentor
                SELECT @UserId = UserId
                FROM UniversityAlma.Mentor
                WHERE MentorId = @MentorId;
                -- Get total number of favorites for the course
                DECLARE @TotalFavorites INT;
                SELECT @TotalFavorites = COUNT(*)
                FROM UniversityAlma.Favorites
                WHERE CourseId = @CourseId;
```

```
                -- Check if there is already a favorites notification for this mentor and course
                SELECT @NotificationId = NotificationId
                FROM UniversityAlma.Notification
                WHERE UserId = @UserId
                        AND Title = 'New favorites for your course'
                        AND Info LIKE '%' + @CourseTitle + '%'
                        AND Checked = 0;

                IF @NotificationId IS NOT NULL
                BEGIN
                        -- Update the existing notification
                        UPDATE UniversityAlma.Notification
                        SET Info = 'Your course ' + @CourseTitle + ' has ' + CAST(@TotalFavorites AS VARCHAR) + ' favorites'
                        WHERE NotificationId = @NotificationId;
                END
                ELSE
                BEGIN
                        -- Insert new notification if it doesn't exist
                        INSERT INTO UniversityAlma.Notification(UserId, Title, Info, Icon)
                        VALUES (@UserId, 'New favorites for your course', 'Your course ' + @CourseTitle + ' has ' + CAST(@TotalFavorites AS VARCHAR) + ' favorites', 'fav.png'
                END

                FETCH NEXT FROM cur INTO @CourseId;
        END

        CLOSE cur;
        DEALLOCATE cur;
END;
GO
```

Img. 10 e 11- As duas metades do código do trigger trigFavoriteNotification

Apagar notificações vistas

Este trigger verifica e apaga qualquer notificação cujo tenha o valor “Checked” igual a 1, ou seja, foi vista e identificada como tal pelo utilizador.

```
-- If a notification is checked, delete the notification
CREATE TRIGGER trigResetNotification
ON UniversityAlma.Notification
AFTER UPDATE
AS
BEGIN
        DELETE FROM UniversityAlma.Notification
        WHERE Checked = 1;
END;
GO
```

Img. 12- Código do Trigger trigResetNotfication

Trigger para o número da sessão ao inserir sessão

Este trigger funciona de forma que, ao inserir uma sessão, independentemente onde foi inserida, o número de cada sessão seja ajustado conforme a situação, criando um cursor e verificando a cada linha quantas sessões têm o mesmo CourseId que o valor armazenado e SessionId que tenha um valor menor ou igual à da linha atual

```
-- Session number trigger (On session insert)
CREATE TRIGGER UniversityAlma.trigInsertSessionNumber
ON UniversityAlma.Session
AFTER INSERT
AS
BEGIN
        DECLARE @CourseId INT;
        DECLARE @SessionId INT;
        DECLARE @SessionNumber INT;

        -- Loop through inserted rows
        DECLARE cur CURSOR FOR
        SELECT SessionId, CourseId
        FROM inserted
        OPEN cur;
        FETCH NEXT FROM cur INTO @SessionId, @CourseId;

        WHILE @@FETCH_STATUS = 0
        BEGIN
                -- Get the next session number
                SELECT @SessionNumber = COUNT(*)
                FROM UniversityAlma.Session
                WHERE CourseId = @CourseId AND SessionId <= @SessionId;

                UPDATE UniversityAlma.Session
                SET Number = @SessionNumber
                WHERE SessionId = @SessionId;

                FETCH NEXT FROM cur INTO @SessionId, @CourseId;
        END

        CLOSE cur;
        DEALLOCATE cur;
END;
GO
```

Img. 13- Código do Trigger trigInsertSessionNumber

Resequenciar número ao apagar sessão

Executado quando uma sessão associada a um certo curso é apagada, este trigger executa um resequenciador de números de sessão, para que os números das sessões que ainda estão presentes no curso sejam reordenados de forma que tenham os números de sessão corretos

```
-- Resequence numbers on session delete
CREATE TRIGGER UniversityAlma.trigDeleteSessionNumber
ON UniversityAlma.Session
AFTER DELETE
AS
BEGIN
        DECLARE @CourseId INT;
        DECLARE cur CURSOR FOR
        SELECT DISTINCT CourseId
        FROM deleted;

        OPEN cur;
        FETCH NEXT FROM cur INTO @CourseId;
        WHILE @@FETCH_STATUS = 0
        BEGIN
                EXEC UniversityAlma.ReSequenceSessionNumbers @CourseId;
                FETCH NEXT FROM cur INTO @CourseId;
        END
        CLOSE cur;
        DEALLOCATE cur;
END;
GO
```

Img. 14- Código do Trigger trigDeleteSessionNumber

Contagem de favoritos

Estes dois triggers têm a função de contar o número de favoritos de um curso cada fez que alguém adiciona ou remove um curso dos seus favoritos.

```
-- FavCount trigger
DROP TRIGGER IF EXISTS trigAddFavorite;
GO
CREATE TRIGGER trigAddFavorite
ON UniversityAlma.Favorites
AFTER INSERT
AS
BEGIN
    UPDATE c
    SET c.FavCount = c.FavCount + i.FavCount
    FROM UniversityAlma.Course c
    JOIN (
        SELECT CourseId, COUNT(*) AS FavCount
        FROM inserted
        GROUP BY CourseId
    ) i ON c.CourseId = i.CourseId;
END;
GO

DROP TRIGGER IF EXISTS trigRemoveFavorite;
GO
CREATE TRIGGER trigRemoveFavorite
ON UniversityAlma.Favorites
AFTER DELETE
AS
BEGIN
    UPDATE c
    SET c.FavCount = c.FavCount - d.FavCount
    FROM UniversityAlma.Course c
    JOIN (
        SELECT CourseId, COUNT(*) AS FavCount
        FROM deleted
        GROUP BY CourseId
    ) d ON c.CourseId = d.CourseId;
END;
GO
```

Img. 15- Código dos Triggers trigAddFavorite e trigRemoveFavorite

Views

Foram criadas duas Views, ambas para expor informação de forma a haver transparência na aplicação. A primeira View mostra todos os detalhes de um curso, e a segunda mostra os detalhes de um perfil

```
-- View to get all course details
DROP VIEW IF EXISTS UniversityAlma.vwCourseDetails;
GO
CREATE VIEW UniversityAlma.vwCourseDetails
WITH SCHEMABINDING
AS
SELECT
        c.CourseId,
        c.Title,
        c.Description,
        c.CategoryId,
        c.MentorId,
        m.Experience AS MentorExperience,
    COUNT_BIG(s.SessionId) AS SessionCount,
        c.FavCount AS FavCount,
        COUNT_BIG(*) AS CountBigAll,  -- Required for indexed views
        c.IsDeleted
FROM
    UniversityAlma.Course c
JOIN
    UniversityAlma.Mentor m ON c.MentorId = m.MentorId
JOIN
    UniversityAlma.Session s ON c.CourseId = s.CourseId
GROUP BY
    c.CourseId,
    c.Title,
    c.Description,
    c.CategoryId,
    c.MentorId,
    m.Experience,
    c.FavCount,
        c.IsDeleted;
GO
CREATE UNIQUE CLUSTERED INDEX IX_vwCourseDetails_CourseId
ON UniversityAlma.vwCourseDetails (CourseId);
GO
```

Img. 16- Código da View vwCourseDetails

```sql
-- View for User Details
DROP VIEW IF EXISTS UniversityAlma.vwUserDetails;
GO
CREATE VIEW UniversityAlma.vwUserDetails
AS
SELECT
        p.ProfileId,
        p.Name,
        p.Username,
        DATEDIFF(YEAR, p.Birthday, GETDATE()) AS Age,
        p.Email,
        CASE WHEN m.UserId IS NOT NULL THEN 'True' ELSE 'False' END AS Mentor
FROM UniversityAlma.Profile p
JOIN UniversityAlma.[User] u ON p.ProfileId = u.ProfileId
LEFT JOIN UniversityAlma.Mentor m ON u.UserId = m.UserId;
GO
```

Img. 17- Código da View vwUserDetails

UDF e Indexes

Devido à possibilidade da capacidade de poder haver uma grande seleção de cursos a selecionar, pode-se tornar difícil encontrar aquele que o utilizador procura, o que mostra a necessidade de implementar um sistema de pesquisa. É nessa situação que a nossa User Defined Funcion entra em ação, tornando possível tal pesquisa, através da utilização de palavras-chave, e o sistema filtrará de forma a mostrar os cursos que tenham tais palavras presentes no nome do curso ou na sua descrição.

```
-- UDF
-- UDF for search courses functionality
CREATE FUNCTION UniversityAlma.fnSearchCourses
(
    @CategoryId INT,
    @SearchTerm VARCHAR(100)
)
RETURNS TABLE
AS
RETURN
(
        SELECT CourseId, Title, Description, SessionCount, FavCount
        FROM UniversityAlma.vwCourseDetails
        WHERE CategoryId = @CategoryId
                AND (Title LIKE '%' + @SearchTerm + '%' OR Description LIKE '%' + @SearchTerm + '%')
);
GO
```

Fig. 18 - Código da User Defined function de pesquisa

Para facilitar a execução de tal função, foram criados dois indexes, um para os títulos dos cursos, e o segunda para as descrições deles. Desta forma, a funcionalidade da UDF de pesquisa é otimizada.

```
-- Apply indexes to optimize search
CREATE INDEX IX_vwCourseDetails_Title
ON UniversityAlma.vwCourseDetails (Title);
GO
CREATE INDEX IX_vwCourseDetails_Description
ON UniversityAlma.vwCourseDetails (Description);
GO
```

Fig. 19 - Código dos Indexes usados

Conclusão

Através do código desenvolvido, foi possível implementar:

- Um sistema de favoritos
- Um sistema de notificações que notifica auditorias, favoritos e remoção de cursos
- Um sistema de registo e login robusto, que certifica que a informação corresponde à existente na base de dados
- O utilizador é capaz de receber notificações e dar um favorito a um curso e procurar por um curso específico a partir de palavras-chave ou uma descrição
- O mentor é capaz de criar e eliminar cursos criados por ele próprio
- O administrador é capaz de remover cursos

Pelo que se pode concluir que as funcionalidades principais e objetivos estabelecidos foram cumpridos com sucesso