Database E-sport turneringsplatform

Af Andreas & Oliver

Afleverings Indhold:

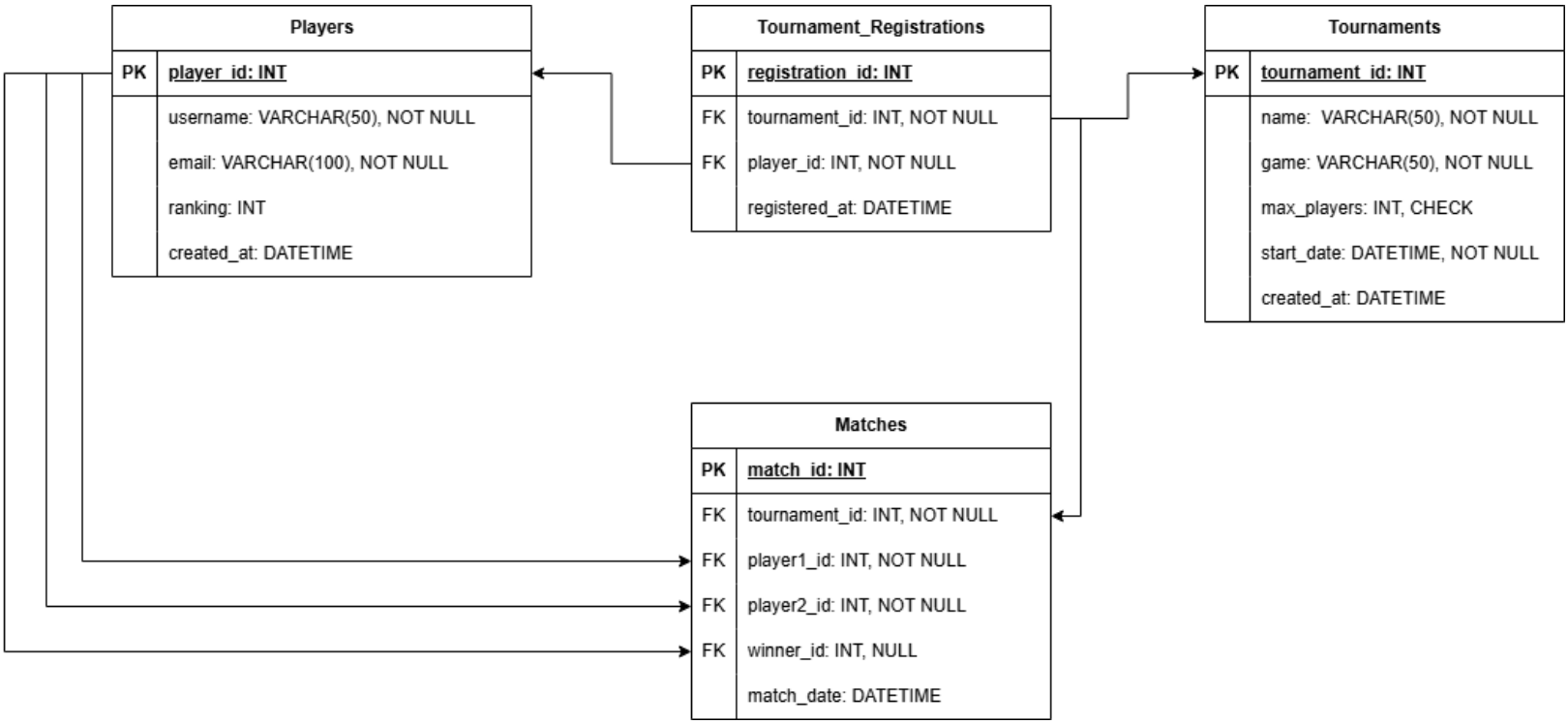
1. **SQL-script** med databaseoprettelse.
2. **SQL script** med de 15 SQL-forespøgselser.
3. **Stored Procedures**, **Functions** og **Triggers**.
4. **Applikation** som angivet i opgave 4.
5. **Dokumentation** for at løsningen fungerer.
6. **Kort redegørelse** med jeres betragtninger om brugen af SQL programmering (fordele og ulemper).

Indholdsfortegnelse:

1 Databasedesign og oprettelse af databasen

Databasedesign

Vi lavede vores databasedesign i drawio diagrams.net for at danne et bedre overblik over entiteterne, primære nøgler (PK), fremmede nøgler (FK), datatyper og constraints. I diagrammet nedenfor vises strukturen af en relationelle database til en E-sports turneringsplatform, hvor spillere kan deltage i turneringer, spille kampe og registrere deres resultater:



Billed 1*: Databasemodel/ER-Diagram design for vores E-sports turneringplatform.*

ERD viser en relationel databse til en E-sports turneringsplatform. Spillere (*Players*) kan tilmelde sig turneringer (*Tournaments*) via "*Tournament Registrations*", som fungerer som en mellem-tabel for many-to-many relationen. Turneringer består af kampe (*Matches*), hvor to spillere konkurrerer, og en vinder registreres. Fremmede nøgler sikrer relationerne mellem tabellerne, hvilket understøtter databasen i at håndtere spillerregistreringer, turneringer og kampresultater effektivt.

2 SQL-Forespørgsler

Nr.	Forespørgsel	SQL-forespørgsler

1	Hent alle turneringer, der starter inden for de næste 30 dage.	SELECT *
		FROM Tournaments
		WHERE start_date BETWEEN GETDATE() AND DATEADD(DAY, 30, GETDATE());
		`SELECT *`
		`FROM Tournaments`
		`WHERE start_date BETWEEN GETDATE() AND DATEADD(DAY, 30, GETDATE());`
		100 %
		Results Messages
		tournament_id: 1, name: Winter Championship, game: Counter-Strike, max_players: 16, start_date: 2025-03-01 12:00:00.000, created_at: 2025-02-28 14:22:38.283

2	Find det antal turneringer, en spiller har deltaget i.	SELECT player_id, COUNT(DISTINCT tournament_id) AS num_tournaments
		FROM Tournament_Registrations
		GROUP BY player_id;
		`SELECT player_id, COUNT(DISTINCT tournament_id) AS num_tournaments`
		`FROM Tournament_Registrations`
		`GROUP BY player_id;`
		100 %
		Results Messages
		player_id / num_tournaments:
		1: 1, 2
		2: 2, 2
		3: 3, 2
		4: 4, 2

3	Vis en liste over spillere registreret i en bestemt turnering.	SELECT p.player_id, p.username, p.email
		FROM Players p
		JOIN Tournament_Registrations tr ON p.player_id = tr.player_id
		WHERE tr.tournament_id = 1;
		`SELECT p.player_id, p.username, p.email`
		`FROM Players p`
		`JOIN Tournament_Registrations tr ON p.player_id = tr.player_id`
		`WHERE tr.tournament_id = 1;`
		100 %
		Results Messages
		player_id / username / email:
		1: 1, PlayerOne, player1@mail.com
		2: 2, PlayerTwo, player2@mail.com
		3: 3, PlayerThree, player3@mail.com
		4: 4, PlayerFour, player4@mail.com

4	Find spillere med flest sejre i en bestemt turnering.	SELECT p.player_id, p.username, COUNT(m.match_id) AS wins FROM Players p JOIN Matches m ON (p.player_id = m.winner_id) WHERE m.tournament_id = 1 GROUP BY p.player_id, p.username ORDER BY wins DESC; 1 SELECT p.player_id, p.username, COUNT(m.match_id) AS wins 2 FROM Players p 3 JOIN Matches m ON (p.player_id = m.winner_id) 4 WHERE m.tournament_id = 1 5 GROUP BY p.player_id, p.username 6 ORDER BY wins DESC; 00 % Results Messages player_id username wins 1 4 PlayerFour 1 2 2 PlayerTwo 1
5	Hent alle kampe, hvor en bestemt spiller har deltaget.	SELECT m.match_id, m.tournament_id, m.player1_id, m.player2_id, m.winner_id, m.match_date FROM Matches m WHERE m.player1_id = 2 1 SELECT m.match_id, m.tournament_id, m.player1_id, m.player2_id, m.winner_id, m.match_date 2 FROM Matches m 3 WHERE m.player1_id = 2 100 % Results Messages match_id tournament_id player1_id player2_id winner_id match_date 1 3 2 2 3 3 2025-04-16 16:00:00.000
6	Hent en spillers tilmeldte turneringer.	SELECT t.tournament_id, t.name, t.game, t.start_date FROM Tournaments t JOIN Tournament_Registrations tr ON t.tournament_id = tr.tournament_id WHERE tr.player_id = 1;

```sql
SELECT t.tournament_id, t.name, t.game, t.start_date
FROM Tournaments t
JOIN Tournament_Registrations tr ON t.tournament_id = tr.tournament_id
WHERE tr.player_id = 1;
```

100 %

Results | Messages

	tournament_id	name	game	start_date
1	1	Winter Championship	Counter-Strike	2025-03-01 12:00:00.000
2	2	Spring Championship	League of Legends	2025-04-15 15:00:00.000

Lavet en playerFive for at teste:

- I screenshottet under kan man set playerFive som har *player_id = 5*, er kun registreret under en turnering.

```sql
SELECT TOP (1000) [registration_id]
      ,[tournament_id]
      ,[player_id]
      ,[registered_at]
  FROM [EsportsTournamentDB].[dbo].[Tournament_Registrations]
```

100 %

Results | Messages

	registration_id	tournament_id	player_id	registered_at
1	1	1	1	2025-02-28 14:39:05.627
2	2	1	2	2025-02-28 14:39:05.627
3	3	1	3	2025-02-28 14:39:05.627
4	4	1	4	2025-02-28 14:39:05.627
5	5	2	1	2025-02-28 14:39:05.627
6	6	2	2	2025-02-28 14:39:05.627
7	7	2	3	2025-02-28 14:39:05.627
8	8	2	4	2025-02-28 14:39:05.627
9	9	1	5	2025-02-28 15:28:49.650

```sql
SELECT t.tournament_id, t.name, t.game, t.start_date
FROM Tournaments t
JOIN Tournament_Registrations tr ON t.tournament_id = tr.tournament_id
WHERE tr.player_id = 5;
```

100 %

Results | Messages

	tournament_id	name	game	start_date
1	1	Winter Championship	Counter-Strike	2025-03-01 12:00:00.000

7	Find de 5 bedst rangerede spillere.	SELECT TOP 5 p.player_id, p.username, p.ranking FROM Players p ORDER BY p.ranking ASC;

<table>
<tr><td></td><td></td><td>

```
SELECT TOP 5 p.player_id, p.username, p.ranking
FROM Players p
ORDER BY p.ranking ASC;
```

100 %

Results | Messages

	player_id	username	ranking
1	3	PlayerThree	1100
2	1	PlayerOne	1200
3	2	PlayerTwo	1250
4	4	PlayerFour	1300
5	5	PlayerFive	1350

</td></tr>
<tr><td>8</td><td>Beregn gennemsnitlig ranking for alle spillere.</td><td>

SELECT AVG(p.ranking) AS avg_ranking
FROM Players p;

```
SELECT AVG(p.ranking) AS avg_ranking
FROM Players p;
```

00 %

Results | Messages

	avg_ranking
1	1240

- NOTE: Hvis playerFive er eksluderet er avg_ranking: **1212**

</td></tr>
<tr><td>9</td><td>Vis turneringer med mindst 5 deltagere.</td><td>

SELECT t.tournament_id, t.name, t.game, COUNT(tr.player_id) AS num_players
FROM Tournaments t
JOIN Tournament_Registrations tr ON t.tournament_id = tr.tournament_id
GROUP BY t.tournament_id, t.name, t.game
HAVING COUNT(tr.player_id) >= 5;

```
SELECT t.tournament_id, t.name, t.game, COUNT(tr.player_id) AS num_players
FROM Tournaments t
JOIN Tournament_Registrations tr ON t.tournament_id = tr.tournament_id
GROUP BY t.tournament_id, t.name, t.game
HAVING COUNT(tr.player_id) >= 5;
```

00 %

Results | Messages

	tournament_id	name	game	num_players
1	1	Winter Championship	Counter-Strike	5

—

</td></tr>
</table>

—
—
TJEK OM DET KUNNE FREMVISE MED SPILLER MED **MINDST 4 DELTAGERE**:

```sql
SELECT t.tournament_id, t.name, t.game, COUNT(tr.player_id) AS num_players
FROM Tournaments t
JOIN Tournament_Registrations tr ON t.tournament_id = tr.tournament_id
GROUP BY t.tournament_id, t.name, t.game
HAVING COUNT(tr.player_id) >= 4;
```

00 %

Results | Messages

	tournament_id	name	game	num_players
1	1	Winter Championship	Counter-Strike	5
2	2	Spring Championship	League of Legends	4

10 Find det samlede antal spillere i systemet.

SELECT COUNT(*) AS total_players
FROM Players;

```sql
SELECT COUNT(*) AS total_players
FROM Players;
```

100 %

Results | Messages

	total_players
1	5

DELETED den tidligere data, go dernæst genoprettet med en modifcerede SQL-query, for at forberede sig til opgave 11, 12, 13, 14 og 15.

INSERT INTO Players (username, email, ranking) VALUES
('PlayerOne', 'player1@mail.com', 1200),
('PlayerTwo', 'player2@mail.com', 1250),
('PlayerThree', 'player3@mail.com', 1100),
('PlayerFour', 'player4@mail.com', 1300),
('PlayerFive', 'player5@mail.com', 1360);
GO

INSERT INTO Tournaments (name, game, max_players, start_date) VALUES
('Winter Championship', 'Counter-Strike', 16, '2025-03-01 12:00:00'),
('Spring Championship', 'League of Legends', 8, '2025-04-15 15:00:00'),
('Summer Championship', 'Counter-Strike', 16, '2025-07-02 15:00:00'),
('Autumn Championship', 'Counter-Strike', 16, '2025-10-12 15:00:00'),
('Holiday Championship', 'League of Legends', 8, '2025-12-12 15:00:00');
GO

INSERT INTO Tournament_Registrations (tournament_id, player_id) VALUES
(1, 1), (1, 2), (1, 3), (1, 4), (1, 5),
(2, 2), (2, 3), (2, 4),

		(3, 2), (3, 3), (3, 4), (3, 1), (3, 5), (4, 1), (4, 3), (4, 4), (4, 5), (5, 1), (5, 3), (5, 5); GO INSERT INTO Matches (tournament_id, player1_id, player2_id, winner_id, match_date) VALUES (1, 1, 2, 2, '2025-03-01 13:00:00'), (1, 3, 4, 4, '2025-03-01 14:00:00'), (1, 1, 5, 1, '2025-03-01 15:00:00'), (2, 2, 3, 3, '2025-04-15 16:00:00'), (2, 2, 4, null, '2025-04-15 17:00:00'), (3, 2, 3, 2, '2025-07-02 16:00:00'), (3, 4, 1, 4, '2025-07-02 17:00:00'), (3, 3, 5, 3, '2025-07-02 18:00:00'), (4, 1, 3, 1, '2025-10-12 16:00:00'), (4, 4, 5, 5, '2025-10-12 17:00:00'), (5, 1, 3, 1, '2025-12-12 16:00:00'), (5, 3, 5, 5, '2025-12-12 17:00:00'); GO

```sql
INSERT INTO Players (username, email, ranking) VALUES
('PlayerOne', 'player1@mail.com', 1200),
('PlayerTwo', 'player2@mail.com', 1250),
('PlayerThree', 'player3@mail.com', 1100),
('PlayerFour', 'player4@mail.com', 1300),
('PlayerFive', 'player5@mail.com', 1360);
GO

INSERT INTO Tournaments (name, game, max_players, start_date) VALUES
('Winter Championship', 'Counter-Strike', 16, '2025-03-01 12:00:00'),
('Spring Championship', 'League of Legends', 8, '2025-04-15 15:00:00'),
('Summer Championship', 'Counter-Strike', 16, '2025-07-02 15:00:00'),
('Autumn Championship', 'Counter-Strike', 16, '2025-10-12 15:00:00'),
('Holiday Championship', 'League of Legends', 8, '2025-12-12 15:00:00');
GO

INSERT INTO Tournament_Registrations (tournament_id, player_id) VALUES
(1, 1), (1, 2), (1, 3), (1, 4), (1, 5),
(2, 2), (2, 3), (2, 4),
(3, 2), (3, 3), (3, 4), (3, 1), (3, 5),
(4, 1), (4, 3), (4, 4), (4, 5),
(5, 1), (5, 3), (5, 5);
GO

INSERT INTO Matches (tournament_id, player1_id, player2_id, winner_id, match_date) VALUES
(1, 1, 2, 2, '2025-03-01 13:00:00'),
(1, 3, 4, 4, '2025-03-01 14:00:00'),
(1, 1, 5, 1, '2025-03-01 15:00:00'),
(2, 2, 3, 3, '2025-04-15 16:00:00'),
(2, 2, 4, null, '2025-04-15 17:00:00'),
(3, 2, 3, 2, '2025-07-02 16:00:00'),
(3, 4, 1, 4, '2025-07-02 17:00:00'),
(3, 3, 5, 3, '2025-07-02 18:00:00'),
(4, 1, 3, 1, '2025-10-12 16:00:00'),
(4, 4, 5, 5, '2025-10-12 17:00:00'),
(5, 1, 3, 1, '2025-12-12 16:00:00'),
(5, 3, 5, 5, '2025-12-12 17:00:00');
GO
```

11	Find alle kampe, der mangler en vinder.	SELECT match_id, tournament_id, player1_id, player2_id, match_date FROM Matches WHERE winner_id IS NULL;

```sql
SELECT match_id, tournament_id, player1_id, player2_id, match_date
FROM Matches
WHERE winner_id IS NULL;
```

100 %

Results | Messages

	match_id	tournament_id	player1_id	player2_id	match_date
1	5	2	2	4	2025-04-15 17:00:00.000

12	Vis de mest populære spil baseret på turneringsantal.	SELECT t.game, COUNT(t.tournament_id) AS tournament_count FROM Tournaments t GROUP BY t.game ORDER BY tournament_count DESC;

```sql
SELECT t.game, COUNT(t.tournament_id) AS tournament_count
FROM Tournaments t
GROUP BY t.game
ORDER BY tournament_count DESC;
```

100 %

Results | Messages

	game	tournament_count
1	Counter-Strike	3
2	League of Legends	2

13 Find de 5 nyeste oprettede turneringer.

```sql
SELECT TOP 5 t.tournament_id, t.name, t.game, t.start_date,
t.created_at
FROM Tournaments t
ORDER BY t.created_at DESC;
```

```sql
SELECT TOP 5 t.tournament_id, t.name, t.game, t.start_date, t.created_at
FROM Tournaments t
ORDER BY t.created_at DESC;
```

100 %

Results | Messages

	tournament_id	name	game	start_date	created_at
1	5	Holiday Championship	League of Legends	2025-12-12 15:00:00.000	2025-02-28 16:28:19.187
2	4	Autumn Championship	Counter-Strike	2025-10-12 15:00:00.000	2025-02-28 16:28:19.187
3	3	Summer Championship	Counter-Strike	2025-07-02 15:00:00.000	2025-02-28 16:28:19.187
4	2	Spring Championship	League of Legends	2025-04-15 15:00:00.000	2025-02-28 16:28:19.187
5	1	Winter Championship	Counter-Strike	2025-03-01 12:00:00.000	2025-02-28 16:28:19.187

14 Find spillere, der har registreret sig i flere end 3 turneringer.

SELECT tr.player_id, COUNT(DISTINCT tr.tournament_id) AS num_tournaments
FROM Tournament_Registrations tr
GROUP BY tr.player_id
HAVING COUNT(DISTINCT tr.tournament_id) > 3;

```sql
SELECT tr.player_id, COUNT(DISTINCT tr.tournament_id) AS num_tournaments
FROM Tournament_Registrations tr
GROUP BY tr.player_id
HAVING COUNT(DISTINCT tr.tournament_id) > 3;
```

100 %

Results | Messages

	player_id	num_tournaments
1	1	4
2	3	5
3	4	4
4	5	4

15	Hent alle kampe i en turnering sorteret efter dato.	SELECT * FROM Matches WHERE tournament_id = 1 ORDER BY match_date ASC; 1 SELECT * 2 FROM Matches 3 WHERE tournament_id = 1 4 ORDER BY match_date ASC; 5 100 % Results Messages match_id, tournament_id, player1_id, player2_id, winner_id, match_date 1: 1, 1, 1, 2, 2, 2025-03-01 13:00:00.000 2: 2, 1, 3, 4, 4, 2025-03-01 14:00:00.000 3: 3, 1, 1, 5, 1, 2025-03-01 15:00:00.000

SQL-scripts

	Oprettelse af tabeller
SQL-scripts	CREATE TABLE Players (player_id INT IDENTITY(1,1) PRIMARY KEY, username VARCHAR(50) UNIQUE NOT NULL, email VARCHAR(100) UNIQUE NOT NULL, ranking INT DEFAULT 0, created_at DATETIME DEFAULT GETDATE()); CREATE TABLE Tournaments (tournament_id INT IDENTITY(1,1) PRIMARY KEY, name VARCHAR(100) NOT NULL, game VARCHAR(50) NOT NULL, max_players INT CHECK (max_players > 1), start_date DATETIME NOT NULL, created_at DATETIME DEFAULT GETDATE()); CREATE TABLE Tournament_Registrations (registration_id INT IDENTITY(1,1) PRIMARY KEY, tournament_id INT NOT NULL, player_id INT NOT NULL, registered_at DATETIME DEFAULT GETDATE(), FOREIGN KEY (tournament_id) REFERENCES Tournaments(tournament_id) ON DELETE CASCADE, FOREIGN KEY (player_id) REFERENCES Players(player_id) ON DELETE CASCADE, UNIQUE (tournament_id, player_id)); CREATE TABLE Matches (match_id INT IDENTITY(1,1) PRIMARY KEY, tournament_id INT NOT NULL, player1_id INT NOT NULL, player2_id INT NOT NULL, winner_id INT NULL, match_date DATETIME NOT NULL, FOREIGN KEY (tournament_id) REFERENCES Tournaments(tournament_id) ON DELETE CASCADE, FOREIGN KEY (player1_id) REFERENCES Players(player_id) ON DELETE NO ACTION, FOREIGN KEY (player2_id) REFERENCES Players(player_id) ON DELETE NO ACTION, FOREIGN KEY (winner_id) REFERENCES Players(player_id) ON DELETE SET NULL);

	Indsættelse af (dummy) data
SQL-scripts	INSERT INTO Players (username, email, ranking) VALUES ('PlayerOne', 'player1@mail.com', 1200), ('PlayerTwo', 'player2@mail.com', 1250), ('PlayerThree', 'player3@mail.com', 1100), ('PlayerFour', 'player4@mail.com', 1300), ('PlayerFive', 'player5@mail.com', 1360); GO INSERT INTO Tournaments (name, game, max_players, start_date) VALUES ('Winter Championship', 'Counter-Strike', 16, '2025-03-01 12:00:00'), ('Spring Championship', 'League of Legends', 8, '2025-04-15 15:00:00'), ('Summer Championship', 'Counter-Strike', 16, '2025-07-02 15:00:00'), ('Autumn Championship', 'Counter-Strike', 16, '2025-10-12 15:00:00'), ('Holiday Championship', 'League of Legends', 8, '2025-12-12 15:00:00'); GO INSERT INTO Tournament_Registrations (tournament_id, player_id) VALUES (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 2), (2, 3), (2, 4), (3, 2), (3, 3), (3, 4), (3, 1), (3, 5), (4, 1), (4, 3), (4, 4), (4, 5), (5, 1), (5, 3), (5, 5); GO INSERT INTO Matches (tournament_id, player1_id, player2_id, winner_id, match_date) VALUES (1, 1, 2, 2, '2025-03-01 13:00:00'), (1, 3, 4, 4, '2025-03-01 14:00:00'), (1, 1, 5, 1, '2025-03-01 15:00:00'), (2, 2, 3, 3, '2025-04-15 16:00:00'), (2, 2, 4, null, '2025-04-15 17:00:00'), (3, 2, 3, 2, '2025-07-02 16:00:00'), (3, 4, 1, 4, '2025-07-02 17:00:00'), (3, 3, 5, 3, '2025-07-02 18:00:00'), (4, 1, 3, 1, '2025-10-12 16:00:00'), (4, 4, 5, 5, '2025-10-12 17:00:00'), (5, 1, 3, 1, '2025-12-12 16:00:00'), (5, 3, 5, 5, '2025-12-12 17:00:00'); GO

3 Stored procedures, functions og triggers

Stored Procedures

	Registrer en ny spiller.

registerPlayer	CREATE PROCEDURE registerPlayer
	@username NVARCHAR(50),
	@email NVARCHAR(100),
	@ranking INT
	AS
	BEGIN
	INSERT INTO Players (username, email, ranking, created_at)
	VALUES (@username, @email, @ranking, GETDATE());
	END;
	GO
	
	Programmability
	Stored Procedures
	System Stored Procedures
	dbo.registerPlayer
	
	2) EXEC
	1 EXEC registerPlayer 'NewPlayer', 'newplayer@email.com', 1400;
	Messages
	(1 row affected)
	Completion time: 2025-03-03T14:34:32.2016306+01:00
	→

	player_id	username	email	ranking	created_at
1	1	PlayerOne	player1@mail.com	1200	2025-02-28 16:28:19.180
2	2	PlayerTwo	player2@mail.com	1250	2025-02-28 16:28:19.180
3	3	PlayerThree	player3@mail.com	1100	2025-02-28 16:28:19.180
4	4	PlayerFour	player4@mail.com	1300	2025-02-28 16:28:19.180
5	5	PlayerFive	player5@mail.com	1360	2025-02-28 16:28:19.180
6	6	NewPlayer	newplayer@email.com	1400	2025-03-03 14:34:32.177

	En spiller tilmelder sig en turnering.
joinTournament	CREATE PROCEDURE joinTournament @player_id INT,

```
    @tournament_id INT
AS
BEGIN
    DECLARE @max_players INT;
    DECLARE @current_players INT;

    -- Hent maks antal spillere for turneringen
    SELECT @max_players = max_players
    FROM Tournaments
    WHERE tournament_id = @tournament_id;

    -- Tæl antallet af spillere, der allerede er tilmeldt turneringen
    SELECT @current_players = COUNT(*)
    FROM Tournament_Registrations
    WHERE tournament_id = @tournament_id;

    -- Tjek om der stadig er plads i turneringen
    IF @current_players < @max_players
    BEGIN
        -- Indsæt registreringen
        INSERT INTO Tournament_Registrations (tournament_id, player_id,
registered_at)
        VALUES (@tournament_id, @player_id, GETDATE());

        PRINT 'Spilleren er tilmeldt turneringen!';
    END
    ELSE
    BEGIN
        PRINT 'Turneringen er allerede fuld!';
    END
END;
GO
```

```sql
CREATE PROCEDURE joinTournament
    @player_id INT,
    @tournament_id INT
AS
BEGIN
    DECLARE @max_players INT;
    DECLARE @current_players INT;

    -- Hent maks antal spillere for turneringen
    SELECT @max_players = max_players
    FROM Tournaments
    WHERE tournament_id = @tournament_id;

    -- Tæl antallet af spillere, der allerede er tilmeldt turneringen
    SELECT @current_players = COUNT(*)
    FROM Tournament_Registrations
    WHERE tournament_id = @tournament_id;

    -- Tjek om der stadig er plads i turneringen
    IF @current_players < @max_players
    BEGIN
        -- Indsæt registreringen
        INSERT INTO Tournament_Registrations (tournament_id, player_id, registered_at)
        VALUES (@tournament_id, @player_id, GETDATE());

        PRINT 'Spilleren er tilmeldt turneringen!';
    END
    ELSE
    BEGIN
        PRINT 'Turneringen er allerede fuld!';
    END
END;
GO
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2025-03-03T14:38:31.3227897+01:00
```

—

```sql
EXEC joinTournament 1, 2;
```

100 %

Messages

```
(1 row affected)
Spilleren er tilmeldt turneringen!

Completion time: 2025-03-03T14:46:27.0940936+01:00
```

—

20	20	5	5	2025-02-28 16:28:19.200
21	21	2	1	2025-03-03 14:46:27.083

	Registrer en kamps resultat.
submitMatchResult	CREATE PROCEDURE submitMatchResult @match_id INT,

```
    @winner_id INT
AS
BEGIN
    -- Tjek om kampen findes
    IF EXISTS (SELECT 1 FROM Matches WHERE match_id =
@match_id)
    BEGIN
        -- Opdater kampen med vinderens ID
        UPDATE Matches
        SET winner_id = @winner_id
        WHERE match_id = @match_id;

        PRINT 'Kampens resultat er opdateret!';
    END
    ELSE
    BEGIN
        PRINT 'Fejl: Kampen findes ikke!';
    END
END;
GO

EXEC submitMatchResult 5, 2;
```

```sql
CREATE PROCEDURE submitMatchResult
    @match_id INT,
    @winner_id INT
AS
BEGIN
    -- Tjek om kampen findes
    IF EXISTS (SELECT 1 FROM Matches WHERE match_id = @match_id)
    BEGIN
        -- Opdater kampen med vinderens ID
        UPDATE Matches
        SET winner_id = @winner_id
        WHERE match_id = @match_id;

        PRINT 'Kampens resultat er opdateret!';
    END
    ELSE
    BEGIN
        PRINT 'Fejl: Kampen findes ikke!';
    END
END;
GO

EXEC submitMatchResult 5, 2;
```

100 %

Messages

```
(1 row affected)
Kampens resultat er opdateret!

Completion time: 2025-03-03T15:02:06.8861764+01:00
```

→

- Programmability
  - Stored Procedures
    - System Stored Procedures
    - dbo.joinTournament
    - dbo.registerPlayer
    - dbo.submitMatchResult

2) Matches (Opdateret resultat)

5	5	2	2	4	2	2025-04-15 17:00:00.000

3) Hvis Matchen ikke findes (Eksempel: match_id = 13, winner_id = 1)

```sql
EXEC submitMatchResult 13, 1;
```

100 %

Messages

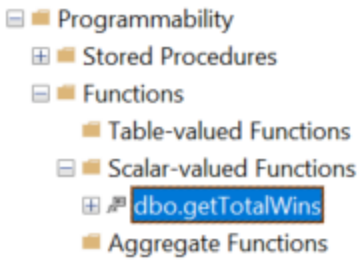
```
Fejl: Kampen findes ikke!

Completion time: 2025-03-03T15:03:47.5339210+01:00
```

Functions

	Returnerer antal sejre for en spiller.

getTotalWins(player_id)	CREATE FUNCTION getTotalWins (@player_id INT)
	RETURNS INT
	AS
	BEGIN
	DECLARE @totalWins INT;
	
	-- Tæl antal sejre for spilleren
	SELECT @totalWins = COUNT(*)
	FROM Matches
	WHERE winner_id = @player_id;
	
	RETURN @totalWins;
	END;
	GO
	
	1 CREATE FUNCTION getTotalWins (@player_id INT)
	2 RETURNS INT
	3 AS
	4 BEGIN
	5 DECLARE @totalWins INT;
	6
	7 -- Tæl antal sejre for spilleren
	8 SELECT @totalWins = COUNT(*)
	9 FROM Matches
	10 WHERE winner_id = @player_id;
	11
	12 RETURN @totalWins;
	13 END;
	14 GO
	15
	100 %
	Messages
	Commands completed successfully.
	
	Completion time: 2025-03-03T15:12:14.3279391+01:00
	→



2) Hvor mange sejre en spiller har (f.eks. player_id = 2).

```
SELECT dbo.getTotalWins(2) AS TotalWins;
```

100 %

Results | Messages

	TotalWins
1	3

- Manuelt tjek, om det er korrekt:

```
SELECT COUNT(*) AS TotalWins FROM Matches WHERE winner_id
```

100 %

Results | Messages

	TotalWins
1	3

	Returnerer turneringens status (upcoming, ongoing, completed),
	• Funktionen tager tournament_id som input. • Den sammenligner startdatoen (start_date) med dags dato (GETDATE()). • Hvis turneringen er afsluttet (dvs. ingen flere ubesvarede kampe), markeres den som "completed". • Returnerer en NVARCHAR(10) værdi (**'upcoming'**, **'ongoing'**, eller **'completed'**).

getTournamentStatus(tournament_id)	CREATE FUNCTION getTournamentStatus (@tournament_id INT)
	RETURNS NVARCHAR(10)
	AS
	BEGIN
	  DECLARE @status NVARCHAR(10);
	  DECLARE @start_date DATETIME;
	  DECLARE @ongoing_matches INT;
	
	  -- Hent startdato for turneringen
	  SELECT @start_date = start_date
	  FROM Tournaments
	  WHERE tournament_id = @tournament_id;
	
	  -- Tjek om turneringen har kampe uden vinder (dvs. den stadig er i gang)
	  SELECT @ongoing_matches = COUNT(*)
	  FROM Matches
	  WHERE tournament_id = @tournament_id AND winner_id IS NULL;
	
	  -- Bestem status
	  IF @start_date IS NULL
	  RETURN 'unknown'; -- Hvis turneringen ikke findes
	
	  IF @start_date > GETDATE()
	  SET @status = 'upcoming'; -- Hvis startdatoen er i fremtiden
	  ELSE IF @ongoing_matches > 0
	  SET @status = 'ongoing'; -- Hvis der stadig er kampe uden vinder
	  ELSE
	  SET @status = 'completed'; -- Hvis alle kampe er afsluttet
	
	  RETURN @status;
	END;
	GO

```sql
CREATE FUNCTION getTournamentStatus (@tournament_id INT)
RETURNS NVARCHAR(10)
AS
BEGIN
    DECLARE @status NVARCHAR(10);
    DECLARE @start_date DATETIME;
    DECLARE @ongoing_matches INT;

    -- Hent startdato for turneringen
    SELECT @start_date = start_date
    FROM Tournaments
    WHERE tournament_id = @tournament_id;

    -- Tjek om turneringen har kampe uden vinder (dvs. den stadig er i gang)
    SELECT @ongoing_matches = COUNT(*)
    FROM Matches
    WHERE tournament_id = @tournament_id AND winner_id IS NULL;

    -- Bestem status
    IF @start_date IS NULL
        RETURN 'unknown'; -- Hvis turneringen ikke findes

    IF @start_date > GETDATE()
        SET @status = 'upcoming'; -- Hvis startdatoen er i fremtiden
    ELSE IF @ongoing_matches > 0
        SET @status = 'ongoing'; -- Hvis der stadig er kampe uden vinder
    ELSE
        SET @status = 'completed'; -- Hvis alle kampe er afsluttet

    RETURN @status;
END;
GO
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2025-03-03T15:25:10.0083008+01:00
```

1) tjekke status på en turnering (tournament_id = 3).

```sql
SELECT dbo.getTournamentStatus(3) AS TournamentStatus;
```

100 %

Results | Messages

	TournamentStatus
1	upcoming

- SELECT dbo.getTournamentStatus(3) AS TournamentStatus;

2) Find en med “**completed**” status:

```sql
SELECT dbo.getTournamentStatus(1) AS TournamentStatus;
```

100 %

Results | Messages

	TournamentStatus
1	completed

3) Find en med “**ongoing**” status”:

```
SELECT dbo.getTournamentStatus(1) AS TournamentStatus;
```

100 %

Results | Messages

	TournamentStatus
1	ongoing

- SELECT dbo.getTournamentStatus(1) AS TournamentStatus;

3.1) Hvis der stadig er en kamp der ingen vinder har endnu, altså winner_id = NULL, og turneringen er startet før dagsdato, så burde den give status = “ongoing”)

```
SELECT * FROM Matches WHERE tournament_id = 1;
```

100 %

Results | Messages

	match_id	tournament_id	player1_id	player2_id	winner_id	match_date
1	1	1	1	2	2	2025-03-01 13:00:00.000
2	2	1	3	4	4	2025-03-01 14:00:00.000
3	3	1	1	5	NULL	2025-03-01 15:00:00.000

→

```
SELECT dbo.getTournamentStatus(1) AS TournamentStatus;
```

100 %

Results | Messages

	TournamentStatus
1	ongoing

- Så giver den “**ongoing**” status.

3.2) Hvis der er fundet en vinder for match_id = 3.

```
SELECT * FROM Matches WHERE tournament_id = 1;
```

100 %

Results | Messages

	match_id	tournament_id	player1_id	player2_id	winner_id	match_date
1	1	1	1	2	2	2025-03-01 13:00:00.000
2	2	1	3	4	4	2025-03-01 14:00:00.000
3	3	1	1	5	5	2025-03-01 15:00:00.000

→

```
SELECT dbo.getTournamentStatus(1) AS TournamentStatus;
```

100 %

Results | Messages

	TournamentStatus
1	completed

- Så giver den “**completed**”. Hvilket betyder at alle kampe har en vinder (winner_id != NULL).

Triggers

	Sikrer, at en turnering ikke overskrider max antal spillere.

beforeInsertRegistration	CREATE TRIGGER beforeInsertRegistration
	ON Tournament_Registrations
	INSTEAD OF INSERT
	AS
	BEGIN
	  DECLARE @tournament_id INT;
	  DECLARE @current_players INT;
	  DECLARE @max_players INT;
	
	  -- Hent tournament_id fra INSERTED (den række, der forsøges indsat)
	  SELECT @tournament_id = tournament_id FROM INSERTED;
	
	  -- Hent max spillere for turneringen
	  SELECT @max_players = max_players
	  FROM Tournaments
	  WHERE tournament_id = @tournament_id;
	
	  -- Tæl nuværende antal spillere i turneringen
	  SELECT @current_players = COUNT(*)
	  FROM Tournament_Registrations
	  WHERE tournament_id = @tournament_id;
	
	  -- Hvis turneringen er fuld, afvis indsættelsen
	  IF @current_players >= @max_players
	  BEGIN
	  PRINT 'Fejl: Turneringen har nået det maksimale antal spillere!';
	  ROLLBACK TRANSACTION;
	  RETURN;
	  END;
	
	  -- Hvis der stadig er plads, indsæt spilleren i turneringen
	  INSERT INTO Tournament_Registrations (tournament_id, player_id, registered_at)
	  SELECT tournament_id, player_id, registered_at FROM INSERTED;
	END;
	GO

```sql
CREATE TRIGGER beforeInsertRegistration
ON Tournament_Registrations
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @tournament_id INT;
    DECLARE @current_players INT;
    DECLARE @max_players INT;

    -- Hent tournament_id fra INSERTED (den række, der forsøges indsat)
    SELECT @tournament_id = tournament_id FROM INSERTED;

    -- Hent max spillere for turneringen
    SELECT @max_players = max_players
    FROM Tournaments
    WHERE tournament_id = @tournament_id;

    -- Tæl nuværende antal spillere i turneringen
    SELECT @current_players = COUNT(*)
    FROM Tournament_Registrations
    WHERE tournament_id = @tournament_id;

    -- Hvis turneringen er fuld, afvis indsættelsen
    IF @current_players >= @max_players
    BEGIN
        PRINT 'Fejl: Turneringen har nået det maksimale antal spillere!';
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    -- Hvis der stadig er plads, indsæt spilleren i turneringen
    INSERT INTO Tournament_Registrations (tournament_id, player_id, registered_at)
    SELECT tournament_id, player_id, registered_at FROM INSERTED;
END;
GO
```

.00 %

Messages

```
Commands completed successfully.
```

→

```sql
INSERT INTO Tournament_Registrations (tournament_id, player_id, registered_at)
VALUES (2, 7, GETDATE());
```

0 %

Messages

```
(1 row affected)

(1 row affected)
```

→

```sql
INSERT INTO Tournament_Registrations (tournament_id, player_id, registered_at)
VALUES (2, 8, GETDATE());
INSERT INTO Tournament_Registrations (tournament_id, player_id, registered_at)
VALUES (2, 9, GETDATE());
INSERT INTO Tournament_Registrations (tournament_id, player_id, registered_at)
VALUES (2, 10, GETDATE());
INSERT INTO Tournament_Registrations (tournament_id, player_id, registered_at)
VALUES (2, 11, GETDATE());
```

00 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
Fejl: Turneringen har nået det maksimale antal spillere!
Msg 3609, Level 16, State 1, Line 7
The transaction ended in the trigger. The batch has been aborted.
```

- Grundet turneringen med ‘tournament_id = 2’ kun kan max have 8 spillere, kan vi se, når vi inserter den 9’ende spiller, at den Trigger vi lavede virker.

4 Brug af databasen fra en applikation

Da jeg(Oliver) ikke har nogen erfaring med Java, besluttede vi at bruge Python til at lave applikationen.

Applikation til joinTournament **med** brug af stored procedures	<pre>import pyodbc
	print(pyodbc.drivers())
	
	print("Starting connection...")
	
	conn = pyodbc.connect(
	  'DRIVER={SQL Server};'
	  'SERVER=DESKTOP-ESAN6RR;'
	  'DATABASE=esports_tournament;'
	  'Trusted_Connection=yes;'
	)
	
	print("Connected to database!")
	
	cursor = conn.cursor()
	print("Executing stored procedure...")
	
	player_id = 6
	tournament_id = 4
	
	try:
	  cursor.execute("EXEC joinTournament ?, ?", player_id, tournament_id)
	  conn.commit()
	  print(f"Player {player_id} joined tournament {tournament_id} successfully!")
	except Exception as e:
	  print("Error:", e)
	
	print("Procedure executed!")
	cursor.close()
	conn.close()
	print("Done!")</pre>

Forklaring af kode	Forbindelse til SQL Server: 1. *pyodbc.connect()* bruges til at oprette en forbindelse til SQL Server. 2. *Trusted_Connection=yes* betyder, at Windows Authentication bruges i stedet for en SQL-login. Eksekvering af stored procedur: 1. *cursor.execute("EXEC joinTournament ?, ?", player_id, tournament_id)* sender parametre til stored proceduren. 2. *conn.comit()* sikrer, at ændringerne gemmes i databasen. Fejlhåndtering: 1. En *try-except* blok fanger fejl og skriver en besked, hvis der opstår problemer.

	Lukning af forbindelsen 1. cursor.close() og conn.close() sikre, at ressourcerne frigives korrekt efter brug.

Applikation til joinTournament **uden** brug af stored procedures	import pyodbc
	
	print("Starting connection...")
	conn = pyodbc.connect(
	'DRIVER={SQL Server};'
	'SERVER=DESKTOP-ESAN6RR;'
	'DATABASE=esports_tournament;'
	'Trusted_Connection=yes;'
	)
	print("Connected to database!")
	
	cursor = conn.cursor()
	print("Executing stored procedure...")
	
	player_id = 6
	tournament_id = 5
	
	try:
	cursor.execute("""
	SELECT COUNT(*)
	FROM Tournament_Registrations
	WHERE tournament_id = ?
	""", tournament_id)
	num_players = cursor.fetchone()[0]
	
	cursor.execute("""
	SELECT max_players
	FROM Tournaments
	WHERE tournament_id = ?
	""", tournament_id)
	max_players = cursor.fetchone()[0]
	
	if num_players >= max_players:
	print(f"Turnering {tournament_id} er allerede fuld!")
	else:
	cursor.execute("""
	INSERT INTO Tournament_Registrations (tournament_id, player_id, registered_at)
	VALUES (?, ?, GETDATE())
	""", tournament_id, player_id)
	
	conn.commit()
	print(f"Player {player_id} successfully joined tournament {tournament_id}!")
	except Exception as e:

	print("Error:", e) print("Procedure executed!") cursor.close() conn.close() print("Done!")
Forklaring af kode	Forbindelse til SQL Server: 1. *pyodbc.connect()* bruges til at oprette en forbindelse til SQL Server. 2. *Trusted_Connection=yes* betyder, at Windows Authentication bruges i stedet for en SQL-login. Tjek for tilgængelige pladser: 1. SQL-forespørgslen *SELECT COUNT(*) FROM Tournament_Registrations WHERE tournament_id = ?* tæller antallet af registrerede spillere. 2. *SELECT max_players FROM Tournaments WHERE tournament_id = ?* henter det maksimale antal tilladte spillere. Validering af turneringskapacitet: 1. Hvis turneringen alerede er fuld, udskrives en fejlmeddelelse. 2. Hvis der er plads, indsættes spilleren i *Tournamen.Registrations*-tabellen. Indsættelse af data: 1. *INSERT INTO Tournament_Registrations (tournament_id, player_id, registered_at) VALUES (?, ?, GETDATE())* registrerer spilleren. 2. *conn.commit()* gemmer ændringerne. Fejlhåndtering og lukning: 1. *try-except* fanger og udskriver eventuelle fejl. 2. *cursor.close()* og *conn.close()* sikrer korrekt lukning af forbindelsen.

Begge løsninger demonstrerer, hvordan man kan registrere spillere i en turnering via SQL Server fra en Python-applikation. Stored procedures kan give bedre performance og vedligeholdelse, men manuel håndtering i Python kan give mere fleksibilitet og lettere debugging afhængigt af behovet.

Kort redegørelse for brugen af SQL programmering (fordele og ulemper).

SQL er et udbredt værktøj til håndtering af data i relationelle databaser. En af de største fordele ved SQL er, at det giver en effektiv og struktureret måde at manipulere data på. Med SQL kan man nemt hente, indsætte, opdatere og slette store mængder data hurtigt og præcist. Derudover er SQL et standardiseret sprog, så lige meget om det er MySQL, SQL Server eller PostgreSQL, ligner syntaksen hinanden.

En anden fordel ved SQL er dens evne til at håndtere komplekse relationer mellem data ved hjælp af joins og foreign keys. Som sikrer dataintegritet og gør det muligt at skabe avancerede forespørgsler, der kan samle information fra flere tabeller. Og brugen af Stored Procedures og

Triggers giver mulighed for at automatisere mange database operationer, hvilket reducere behovet for manuel indgreb, og mindsker risikoen for fejl.

Vi har dog stødt på en del problemer med vores stored procedures. Udfordringen var, at eksekveringen ofte fejlede, men alligevel blev dataen inkrementeret uden, at der faktisk blev indsat nogen værdier. Problemet blev løst ved at implementere error handling.