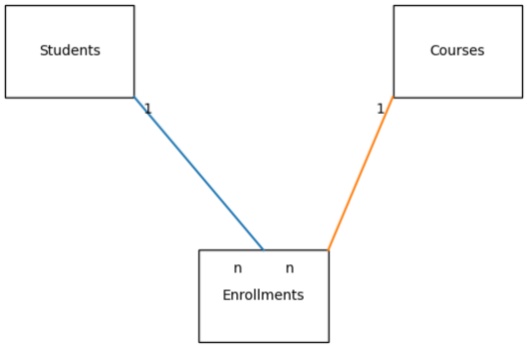
Databasuppgift – Hogwarts Kursplanering
Författare:
Kurs:
Datum:

Innehållsförteckning

1. ER-diagram
Textbaserat diagram:
[Students] 1 --- n [Enrollments] n --- 1 [Courses]
Beskrivning:
Students innehåller elever, Courses innehåller kurser och lärare.
Enrollments kopplar elever till kurser.
Detta skapar en n-till-n-relation mellan Students och Courses.



2. Tabellbeskrivningar
Students – lagrar elevinformation.
Courses – lagrar kursnamn och lärare.
Enrollments – kopplar elever till kurser.
Datatyper:
• INTEGER för ID
• TEXT för namn
• FOREIGN KEY för relationssäkerhet
• UNIQUE(StudentId, CourseId) förhindrar dubbletter

Byte datatyper till NVARCHAR (SQL Server-anpassning)
När jag började arbeta med databasen använde jag datatypen *TEXT*, vilket är vanligt i SQLite. När jag sedan körde koden i Microsoft SQL Server uppstod felmeddelandet “text and varchar are incompatible”, eftersom SQL Server inte tillåter jämförelser mellan TEXT och vanliga strängar. För att lösa detta och göra databasen kompatibel med SQL Server ändrade jag alla strängkolumner till NVARCHAR och kunde köra UPDATE-kommandon utan fel, vilket säkerställer att databasen fungerar stabilt i SQL Server-miljön. Detta gjorde också databasen mer konsekvent och framtidssäker.

3. SQL – CREATE
CREATE TABLE Students (Id INTEGER PRIMARY KEY,Name TEXT NOT NULL,House TEXT NOT NULL);
CREATE TABLE Courses (Id INTEGER PRIMARY KEY,Name TEXT NOT NULL,Teacher TEXTNOT NULL);
CREATE TABLE Enrollments (Id INTEGER PRIMARY KEY,StudentId INTEGER NOT NULL, CourseId INTEGER NOT NULL,FOREIGN KEY (StudentId) REFERENCES Students(Id), FOREIGN KEY (CourseId) REFERENCES Courses(Id),UNIQUE (StudentId, CourseId));

4. SQL – INSERT
INSERT INTO Students (Id, Name, House) VALUES
(1, 'Harry Potter', 'Gryffindor'),
(2, 'Hermione Granger', 'Gryffindor'),
(3, 'Ron Weasley', 'Gryffindor'),
(4, 'Draco Malfoy', 'Slytherin'),
(5, 'Luna Lovegood', 'Ravenclaw'),
(6, 'Cedric Diggory', 'Hufflepuff');
INSERT INTO Courses (Id, Name, Teacher) VALUES
(1, 'Defense Against the Dark Arts', 'Remus Lupin'),
(2, 'Potions', 'Severus Snape'),
(3, 'Charms', 'Filius Flitwick'),
(4, 'Herbology', 'Pomona Sprout');
INSERT INTO Enrollments (StudentId, CourseId) VALUES
(1, 1), (1, 3), (2, 1), (2, 2), (2, 3),
(3, 1), (4, 2), (5, 3), (6, 4);

5. SQL – SELECT Basic
SELECT * FROM Students;
SELECT * FROM Students WHERE House = 'Gryffindor';
SELECT * FROM Courses ORDER BY Name;
SELECT * FROM Courses WHERE Teacher LIKE 'S%';
SELECT House, COUNT*() AS Count FROM Students GROUP BY House;*
SELECT CourseId, COUNT() AS Enrolled FROM Enrollments GROUP BY CourseId;

6. SQL – JOIN
SELECT Students.Name AS Student, Courses.Name AS Course
FROM Enrollments

JOIN Students ON Enrollments.StudentId = Students.Id
JOIN Courses ON Enrollments.CourseId = Courses.Id;
SELECT Courses.Name AS Course, Courses.Teacher, Students.Name AS Student
FROM Enrollments
JOIN Courses ON Enrollments.CourseId = Courses.Id
JOIN Students ON Enrollments.StudentId = Students.Id;

7. SQL – UPDATE
UPDATE Courses
SET Teacher = 'Horace Slughorn'
WHERE Id = 2;
UPDATE Students
SET House = 'Gryffindor'
WHERE Name = 'Luna Lovegood';

8. SQL – DELETE
DELETE FROM Enrollments
WHERE StudentId = 4 AND CourseId = 2;

9. Jämförelse SQL ↔ LINQ
A. WHERE + ORDER BY
SQL:
SELECT * FROM Students WHERE House = 'Gryffindor' ORDER BY Name;
LINQ:
Students.Where(s => s.House == "Gryffindor").OrderBy(s => s.Name).ToList();

B. JOIN
SQL:
SELECT Students.Name, Courses.Name FROM Enrollments JOIN Students ... JOIN Courses ...
LINQ:
from e in Enrollments
join s in Students on e.StudentId equals s.Id
join c in Courses on e.CourseId equals c.Id
select new { Student = s.Name, Course = c.Name };

C. GROUP BY
SQL:
SELECT House, COUNT(*) FROM Students GROUP BY House;
LINQ:
Students.GroupBy(s => s.House).Select(g => new { House = g.Key, Count = g.Count() });

10. Säkerhet
Säker åtkomst skyddar databasen mot obehöriga. Authentication avgör vem användaren är, och authorization avgör vad användaren får göra. Lösenord ska hash:as istället för att lagras i klartext. Parameteriserade queries eller ORM skyddar mot SQL-injektion. Minsta privilegium bör alltid användas.

11. Versionshantering
Git används för att spåra ändringar, organisera projektet och kunna återställa tidigare versioner. Genom att dela upp SQL-koden i separata filer blir projektet tydligt och strukturerat. Meningsfulla commit-meddelanden gör det enklare att följa utvecklingen.

12. Reflektion
Under arbetets gång lärde jag mig hur tabeller i en relationsdatabas kopplas ihop genom primärnycklar och främmande nycklar. JOIN-frågor var det svåraste från början, men efter att jag testat olika exempel förstod jag logiken. Versionshanteringen blev också tydligare när filerna hamnade på rätt plats. Uppgiften gav mig bättre förståelse för hur databaser fungerar i praktiken och hur SQL och LINQ hänger ihop med backend-utveckling.