Handelsakademin, Göteborg
SQL Azure – BED20
Grupp Gul Hotelldatabas
Eliza, Eric, Erik, Hanna

1. Uppgift

Uppgiften som tilldelats oss handlar om att skapa ett bokningssystem i form av en databas för ett hotell. Detaljinformation och önskemål tillhandahålls av en fiktiv kund. Huvudmålet med uppgiften är att skapa kompletta databaser utifrån verksamhetens behov.

2. Struktur

Innan själva databasen skapades arbetade gruppen ihop ett ER-diagram med hjälp av verktyget Lucidchart. Kraven från *Appendix A. Projektdata, från fiktiv kund* användes som grund för de tabeller som utgör ER-diagrammet. Listan delades upp i mindre delar där krav som tillhörde samma ämne blev till attribut; kundnamn och kundtelefon sparades till exempel under samma tabell för kundinformation.

Under arbetets gång har flera tabeller tillkommit och kopplats samman med originalen. ”ExtraBills” är ett sådant exempel, en tabell som sköter mer specifika kostnader som tillkommit under gästens vistelse och sedan kopplas ihop med tabellerna för räkningar.

2.1 Normalformer

Tabellerna har byggts med hänsyn till den första-, andra- och tredje normalformen. Första normalformen, 1NF, kräver att tabellerna innehåller atomära värden och har varit enkel att tillgodose, då vi redan från början undvikit att skapa dubbletter bland värdena och splittrat tabeller där vi ansett att det har varit nödvändigt.

Genom att kontrollera att tuplerna i varje tabell hade ett funktionellt beroende till deras primärnycklar kunde vi även uppfylla kraven för den andra normalformen, 2NF. Varje kolumn kommer alltså beskriva det tabellens primärnyckeln har i syfte att identifiera. Tabellen ”Rooms” attribut ”RoomNumber”, ”Price” och ”NoBeds” är alla relaterade till ämnet rum. Vi valde däremot att skapa en ny tabell, ”RoomTypes”, som är länkad genom en Främmande Nyckel, FK, tillbaka till ”Rooms”, eftersom ett hotell kan ha en mängd olika typer av rum utöver egenskaperna som återfinns i ”Rooms”.

Slutligen, i enlighet med den tredje normalformen, 3NF, är attributen i tabellerna beroende av deras primärnycklar och inte varandra. I tabellen ”Employees”, till exempel, finns inga transitiva förhållanden mellan attributen; den anställdes titel påverkas inte av dennes för- eller efternamn.

2.2 Integritet och säkerhet

Den vanligaste formen av farliga intrång i databaser, framförallt i SQL-Server, som vi pratar om här, kallas för “SQL-Injection”. Tekniken används av individer som inte ska ha åtkomst men söker det iallafall (https://www.w3schools.com/sql/sql_injection.asp). Intrången kan förebyggas genom parametrar som anges i ett query eller genom att skapa en “Stored

Procedure” för just detta ändamål. Stored Procedures är namngivna grupper av statements som kan köras och återanvändas av flera program.

Stored Procedures kan modifiera data i en databas men är aldrig bundna till något specifikt objekt eller databas. Dem agerar som ett skyddande lager mellan användaren och själva databasen då användaren må ändra eller lägga till data, men saknar kontroll över dessa kommandon. De kan även reducera nätverkstrafiken mellan klienter och servrar eftersom Stored Procedures kommandon exekveras som en hel hop istället för rad efter rad av individuella kod-snuttar. Stored Procedures är också vitala för underhållet av Data integrity, eftersom tekniken utförs på ett konsekvent och förutsägbart sätt. Fördelarna är många och vi har valt att implementera följande Stored Procedures i vår hotelldatabas:

(Insert våra egna Stored Procedures)

När man skapar och administrerar en databas bör man också ha i åtanke hur man ska gå tillväga med tillgång och verifiering av personal som skall kunna använda databasens information. Andra steg som man bör vidta om man vill säkra sin databas ytterligare är att:

- Alltid uppdatera SQL-Server, samt alla plugins, Frameworks, etcetera.
- Använda sig av **“Principle of Least Privilege”** när man ska skapa användarkonton och liknande saker. PoLP betyder att man ska bara ge användaren rättigheter att göra operationer som är nödvändiga. Till exempel, att bara låta denne skriva SELECT-statements för att få fram viss personal på en avdelning och inte ge den anställde möjlighet att manipulera data genom INSERT, DELETE eller UPDATE.
- Kryptera sin data genom att kräva nycklar för lösenord.

I enlighet med GDPRs principier ska bl.a personuppgifter skyddas så obehöriga inte har åtkomst till dem och raderas om det inte längre behövs. Det är numera även ett krav att bevisa med dokumentation att alla principer följs, men då inga verkliga uppgifter lagrats i vår databas är det knappast essentiellt att visa i denna rapport.

3. Implementation

All testdata har lagts till via INSERT INTO-queries. Den logiska ordningen att följa och för att lägga till data från grunden är:

- *RoomTypes*: Storlekar och kortfattade beskrivningar. Kardinalitet: One-to-many till Rooms.
- *Rooms*: Rumsnummer, pris och antalet sängar. Kardinalitet: One-to-zero-or-many till BookingRows.
- *Employees*: Namn, titel och lön. Kardinalitet: One-to-many till ContactPerson.
- *ContactPerson*: Namn, titel.
- *Products*: Namn, beskrivning och pris. Kardinalitet: One-to-zero-or-many till ExtraBillRows.
- *Customers*: Personliga kunduppgifter. Kardinalitet: One-to-zero-or-many till Bookings.
- *Bookings:* Information om datum, önskemål, pris och utebliven ankomst. Kardinalitet: One-to-zero-or-many till Feedback, One-to-many till BookingRows och One-to-many till Messages.
- *BookingRows:* Specifik information om bokning som rum, extrasängar, och rabatt. Kardinalitet: One-to-zero-or-many till ExtraBills

- *Messages:* Meddelande från kund, datum.
- *ExtraBills:* Betalad eller ej (booleanskt). Kardinalitet: One-to-many till ExtraBillRows och InvoiceRows.
- *ExtraBillRows:* Består endast av nycklar, PK och två FK.
- *PaymentMethods:* Lista av betalsmetoder. Kardinalitet: One-to-zero-or-many till Invoices.
- *Invoices:* Faktura, summa, betalmetod, datum. Kardinalitet: One-to-many till InvoiceRows.
- *InvoiceRows:* Specifik fakurainformation kopplad antingen till Bookings eller ExtraBills.
- *Ratings:* Betyg från kund till rum.
- *Feedback:* Kommentar från kund kopplad till bokning.

Kunder har alltså friheten att boka flera rum under samma bokning, där extrakostnader är kopplade till bokningsraderna samt fakturaraderna. På så sätt skapas en "optional-relation" mellan InvoiceRows, ExtraBills och Bookings. Om en kund till exempel inte har köpt något från minibaren som annars hamnat under tabellen ExtraBills ska den tabellen inte behöva vara med i beräkningarna, däremot måste Bookings vara med.

Anledningen till att det ligger ett pris-attribut under tabellen Bookings (och inte bara på BookingRows) är för att datorn eventuellt ska kunna generara ett pris för bokningar under, till exempel, juldagarna. Priser på hotell kan fluktuera beroende på datum och andra faktorer. Än så länge finns ingen sådan automatisering i form av en Trigger/Stored Procedure installerad, men strukturen är åtminstone utformad för framtida tillägg som detta.

Struktursordningen har sett olika ut under arbetets gång, särskilt då vi upptäckte felaktiga relationer mellan tabellerna, som en dubbelkoppling mellan Bookings och Invoices. Fel uppkom särskilt då testdatan skulle matas in, då man sakteligen såg tabellerna och nycklarna ur ett mer praktiskt perspektiv. Självklart måste data för bokningar läggas in före datan för räkningarna och självklart ska inga many-to-many kardinaliteter förekomma. Ologiska fel som kan verka uppenbara i den verkliga världen kan ligga undangömda ett bra tag när man försöker samarbeta med datorn. För att inte tala om samarbetat med många olika datorer som måste synkas.

3.1 Konsollapplikation

Konsollapplikationen har formats efter de fyra, grundläggande funktionerna för lagring - CRUD-modellen. Programmet, som är skrivet i C# med en connection till SQL, är alltså kapabelt att lägga till, välja, uppdatera och ta bort data från valda tabeller.

- En enkel meny som ska ge användaren valet till om den vill göra Insert, Select, Update eller Delete.
- Denna meny pekar då beroende på val till en metod som tillhör respektive funktion.
- Insert är gjort enkelt med en try, catch, finally statement, Insert är gjord med ADO.NET.
- Select är gjort med Dapper så att den ska välja X i test tabellen.
- Update kommer att uppdatera en vald rad med nya värden som är hårdkodade.
- Delete kommer att ta bort en vald rad eller kan man ställa in så att den tar bort alla rader i test tabellen.

3.3 Azure databas

Azure databasen är, enkelt sagt, vår databas som ligger i en molnserver istället för att köra den med våra lokala Visual Studio plugin eller SQL-Express Server. Vi valde att först öppna en tom databas där och sedan publicera vår egen databas till den platsen, istället för att låta Azure och Visual Studio kommunicera direkt, då det var rekommenderat.

4. Tester och resultat

Problem uppstod när våra Stored Procedures skulle exekveras, men det visade sig snabbt att scripten inte stämde överens hos våra olika datorer, de behövde uppdateras. Man får inte glömma att göra en “Pull” regelbundet i Visual Studio så man får den nya datan och alla nya ändringar från Github projektet när man jobbar som grupp. och att ständigt hålla programmen uppdaterade. Kardinaliteten mellan de olika tabellerna kan behöva uppdateras, då det på vissa ställen kan se ut som om det försiggår sig om “one-and-only-one-” symboler där det egentligen ska vara “one”. Normaliseringen kan säkert förbättras också, vissa tabeller hade definitivt mått bra av att brytas ned till ännu mindre beståndsdelar.

Även om försök gjordes att skriva konsoll-applikationen med ADO.NET övergavs detta snart och ersattes av Dapper, som fungerade betydligt bättre. Att se till att Connection-strängen mellan C# och SQL var också krångligt först men nu är syntaxen rätt. Att lägga till testdata, som annars går mycket snabbt, kunde ta ett bra tag då gamla nycklar och ”constraints” låg och spökade. Även om vi fick förslaget att rensa hela databasen med ett DROP-statement kunde problemet lösas genom att återvända till ER-diagrammet och koppla om tabellerna, därefter den lokala databasen samt de gemensamma scripten som synkas mellan oss.

5. Epilog

Sammanfattat har projektet varit väldigt lärorikt för alla inblandade. Det finns mycket som hade kunnat göras annorlunda, men resultatet är vi stolta över. Vi delar gärna mer bilder och information om så önskas.