Min databasdesign.

Jag har valt att använda ett antal ”knut-tabeller”, med det menar jag att jag knyter ihop olika tabeller med hjälp av tabeller som jag bara har forreign keys i. Detta val gjorde jag för att undvika att ha så många slags värden i en tabell, jag har försökt hålla mig (i de flesta tabellerna till ett slags värde i varje och sedan använda id till andra tabeller för att få tag i de andra värdena som behövs. Detta gör att jag kan få lägre redundans (upprepning av samma värde som tar plats och kraft) och det kan dessutom vara bra ur säkerhetssynpunkt om man av någon anledning vill dölja vissa delar av informationen genom att inte ge ut värdena på en viss forreign key till exempel (högaktuellt i GDPR nu) och så blir det inte lika lätt fel om man uppdaterar ett värde i en tabell eftersom den finns just bara i den tabellen så då blir det uppdaterat likadant överallt.

Genom att använda sig av vyer som man har gjort på förhand så kan man ändå se alla värden i en tabell om man vill.

Flera av värdena som jag har i tabellerna, till exempel postnummer i adress tabellen skulle självfallet också kunna hamna i en egen tabell med knut-tabell till, om man har en mycket större databas än min lilla, men jag brydde mig inte om det just nu eftersom det var ont om tid och jag inte hade så mycket data.

Anledningen till att jag bara har datum i min payment tabell är medveten, man kan räkna ut kostnad genom en sp till den, och kund kan man få genom rental med hjälp av en vy, så den informationen var inte nödvändig att skriva ned (redundans) och om man skriver ned något som är uträknat och det är något som ändras någon annanstans i tabellerna, hur vet man då vilket pris som gäller? Alltså blev det en sådan sak som man kan räkna ut genom sp och vyer.

Jag har en film per rental i rental-tabellen, det är för att det var lättast att hantera utlåning och tillbakalämning på det sättet, det blir inte så mycket upprepad data i alla fall eftersom jag har id:n på allt utom datum. Eftersom man dessutom bara lånar ett exemplar av varje film och kanske inte så väldigt många på en gång heller så kändes det här som en fungerande lösning, jag inser att om man har stora ordrar på många varor åt gången osv så skulle detta inte fungera särskilt bra men så är det inte i det här fallet.

Jag har också valt att ha två booleans (tinyint) i film\_copy och rental för att firman ska kunna se och håll ordning på om en film är utlånad och om den är returnerad, annars blir det svårt att veta vem som inte lämnat tillbaka filmen.

Det blir som sagt en del invecklade vyer med flera joinar för att kunna få fram en del av datan, men det tyckte jag det var värt för att kunna få bättre koll på redundans och säkerhet.

Jag har också valt att göra rental.returned not null för att man måste kunna se om den är tillbakalämnad eller inte, medan jag har gjort film\_copy.rentedOut möjlig att nulla ifall det skulle vara så att någon filmkopia blir dålig eller av någon annan anledning måste tas ur systemet så måste man ändå kunna se vilken film det var från början om man vill titta bakåt i tiden i rental (det måste väl lagras i ett antal år, preskriptionstid osv).

Payment tabellen har jag inte med i rent\_out\_film för att jag tänker mig att det får vara en egen del när betalning kommer in på konto får den triggas till att lägga till rentalId och datum som betalningen kommer in, men det skrev du inte om någonstans i uppgiften så det lämnade jag just nu.

Om man av någon anledning sätter in en film i late\_film funktionen som inte är utlånad så får man meddelande error, jag tyckte det var nödvändigt att hantera den utkomsten med.

Storage Engine

Jag valde InnoDB för att de prioriterar dataintegritet (eftersom jag har persondata) och den fungerar bra om man har en relationsbaserad databas. Att den använder mera RAM och andra system resurser gör inte så mycket i mitt fall eftersom det inte är en särskilt stor och komplicerad databas och verksamhet ändå, så då spelar det ingen större roll om man får vänta den lilla extra tiden som en människa inte ens lägger märke till förmodligen. Just inserts och updates går snabbare med InnoDB än med vissa andra pga att deras tabeller är låsta på radnivå istället för tabellnivå vilket också är en fördel.