# Databasimplementation

## Intoduktion

För denna del av uppgiften har den översta, högra delen av EER-Schemat använts, det vill säga; *Vapen, Skepp, Alien, Ras, Regalien* samt *Oregalien*. Tanken är att dokumentera vilka utomjordingar som har tillgångar som kan vara av risk eller annan meningsfullt värde att ha kunskap om, speciellt vid eventuell incident eller liknande, men ett stort fokus ligger även på att kunna gömma/hemlighetsstämpla information samt att kunna radera utomjordingar från databasen där uppgifter har utgått eller liknande.

## Datatyper: CHAR

Under implementationen har varierande datatyper använts, bland annat används datatypen CHAR för både en aliens *IDkod*, personnummer (*pnr*) av registrerade aliens samt en oregistrerad aliens *införelsedatum* i databasen. Detta känns naturligt då de bland annat alltid ska ha en fast längd men även för att det blir enkelt att kunna styra formatet med tillexempel *rlike* eller *regex*.

## Datatyper: TINYINT + UNSIGNED

Datatypen TINYINT används i fältet *id* under *farlighet*stabellen eftersom detta fält *alltid* kommer att ha en ganska liten mängd rader då dessa ska beskriva vilka farlighetsnivåer som finns, då dessa idag enbart är 8 räcker detta. Även om man en dag skulle vilja utöka antalet nivåer finns det utrymme för detta. Att använda TINYINT för detta fält är fördelaktigt eftersom datatypen är kompakt och enbart kräver 1 byte av lagring. Detta medför även en bättre query prestanda eftersom det krävs mindre minne för att hämta data från databasen. Med TINYINT används även UNSIGNED eftersom detta fält inte ska kunna vara negativ. Det är inte så intuitivt att ha ett id-fält som är negativt.

## Datatyper: DATETIME

Även datatypen DATETIME har använts och det förekommer bland annat i fälten *logg\_tid* och i *logg\_datum* under tabellen *Ras\_Logg* respektive *Alien\_Hemlighetstämplande\_Logg* som används för dokumentationsföring av raser och utomjordingar som för en eller annan anledning fått information om sig hemligstämplade. Meningen är att man ska kunna gå tillbaka och se när något ändrades, tabellen agerar alltså lite som en historik. Med detta blir det enkelt att logga saker, samt att det blir lätt att hitta, om man kommer ihåg ungefär vilken period som ändringen utfördes.

## CONSTRAINT: CHECK

I denna uppgift har CHECK använts för att sätta regler samt för att säkerställa att databasen fungerar som förväntat. CHECK används på många ställen, bland annat i *vapen*tabellen för att säkerställa att ett vapen ägs (är kopplad mot) ett skepps id eller ett aliens id, detta är på grund av kravet att ett vapen måste ha en ägare, men inte kan ägas av både en alien och ett skepp samtidigt. Det används även för att säkerställa att *IDkod*, *pnr*, och *införelsedatum* som tidigare nämnts, har ett korrekt format vid insättning av nya rader.

## CONSTRAINT: UNIQUE

UNIQUE används för att säkerställa att ett värde i en kolumn alltid kommer att vara unikt och inte finnas i flera olika rader, för den anledningen har UNIQUE har använts väldigt varsamt, enbart i *farlighets*tabellen samt i *ras*tabellen. Hela poängen med en farlighetsgrad är att de i sig ska vara unika och beskrivande för just den graden. Det finns ingen mening med att ha fler än 1 grad av ”harmlös” eftersom de representerar samma sak. Detsamma gäller namnet för en ras, det finns ingen mening att ha flera rader som beskriver samma ras, med samma namn. Det är en representation varav det enbart behövs 1 av.

## CONSTRAINT: NOT NULL

NOT NULL finns även den på många ställen, men den som är mest kritiskt är nog den i tabellen *Registrerad\_Alien\_Hemplanet* i fältet *hemplanet*. Detta är för att databasen är uppbyggd på så sätt att ett skepps tillverkningsplanet är baserat på hemplaneten en alien har. Eftersom uppgiften beskriver att ett skepp inte får sakna tillverkningsplanet, och att denna, för alla skepp, anges automatiskt har ett antagande gjorts där en aliens hemplanet också även *antagligen* är skeppets tillverkningsplanet.0