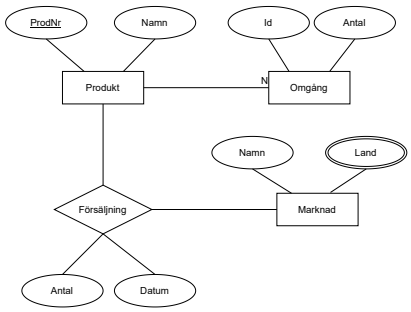
Facit och kommentarer till tentamen 2021-01-08 i DA4001

1. En lösning kan se ut så här:



   De viktiga begreppen i frågan är *produkt*, *marknad*, och *omgång*, så jag skapar entitetstyper för dessa. Texten nämner data som vi vill lagra för de olika entiteterna, så tex produktnummer, produktnamn, etc blir naturliga attribut till entiteterna. Marknader består av flera länder, så jag väljer att använda ett flervärt attribut där. Man skulle kunna lyfta ut land som entitetstyp också, men vi har ju inte några attribut förutom landsnamn att knyta till länderna, så det är enklare med ett flervärt attribut.

   Varje produkt tillverkas i flera omgångar, så det får bli en 1-N-relation.

   Försäljningsdata kan man tänka sig hantera på flera sätt, men genom att låta försäljning bli en namngiven relation med försäljningssiffror och datum (för månadsvisa rapporter) som attribut, så får vi en enkel representation.

2. Man kan identifiera följande relationer:

   - **Chemical**
     ChemId ProdID Name ProviderName
     ProdId borde vara unikt och skulle kunna vara ensam primärnyckel, men eftersom olika producenter antagligen inte kommunierar kan vi inte lita på det, så ChemId och ProdId får bilda primärnyckel tillsammans med producentnamnet.
   - **ChemDependency**
     ChemId Dependency
     Relationen NeededFor implementerar jag med relationen ChemDependency. Ett ChemId kan bero av flera andra ChemId (kallade Dependency här).
   - **Provider**
     ProviderName ContactName Phone
     Vi listar kontakter för det olika producenterna. En producent/provider kan ha flera kontakter, så ProviderName tillsammans med ContactName blir unikt tupel och därmed primärnyckel. Det kanske är lite svagt att anta att kontaktnamnet är unikt, men det är väl ett rimligt förenklande antagande?

- **InventoryInfo**

LabRoom	ManagerName

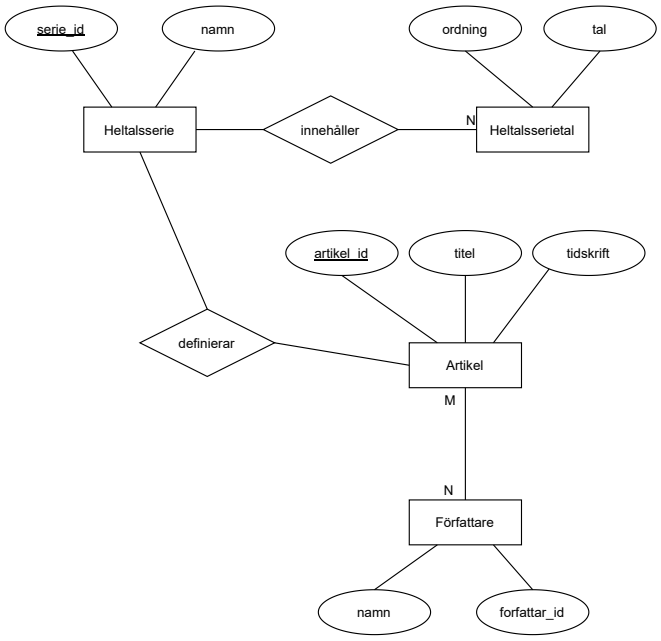
  Varje *Inventory* tillhör ett rum och har en *Manager*. Jag låter labbrummet identifiera inventariet, så entitetstypen Inventory ger denna tabell.
- **ChemicalInventory**

LabRoom	ChemId	ProdId	Quantity

  Relationen "has" knyter ihop ett inventarium med en kemisk produkt, så primärnycklarna från inventariet (LabRoom) och ChemId+ProdId blir primärnyckel här. Och så har vi relationenns attribut Quantity som attribut i relationen. Det går nu alltså att en kemikalie i flera rum, och varje rum kan ha flera kemikalier.

Svagheter: Entiteten **Chemical** representerar både generiska produkter och kommersiella produkter. Så för laboratoriets hemgjorda produkter är det ju inte rimligt att ha ett ProdId.

3. Ett möjligt ER-diagram:



Jag har gjort det enkelt för mig med en entitetstyp per relation. Jag har namngett relationer mellan Heltalsserie och Heltalsserietal/Artikel för tydlighets skull. Dessa relationer "tvingar in" seriernas identifierare i relationerna Heltalsserietal och Artikel. Jag har undvikit att ta ställning till huruvida man kan ha en eller flera artiklar till en heltalsserie, men det skulle man kunna göra genom att kvantifiera relationen *definierar*. Varje författare kan vara med i flera artiklar och varje artikel kan ha flera artiklar, en viktig liten detalj.

4a. Första normalform kräver att alla kolumner innehåller enkla och atomära värden. Den enda kolumnen som skulle kunna bryta det är *namn* i relationen Författare, eftersom man skulle kunna bryta upp den i tex förnamn och efternamn, men med tanke på hur man kan vilja använda databasen är det fullt rimligt att säga att den är på 1NF.

b. För 2NF ska modellen vara på 1NF och *varje icke-nyckelattribut är fullständigt funktionellt beroende av varje kandidatnyckel.* Det är trivialt sant för relationen Heltalsserie, där `namn` beror av den enda kandidatnyckeln `serie_id`.

I tabell Heltalsserietal är paret `serie_id, ordning` enda kandidatnyckeln och `tal` är då FFB.

Relationen Författare bryter mot 2NF, eftersom `namn` är FFB av `författar_id`, som är en del `doi, författar_id`, den enda kandidatnyckeln. För att nå 2NF kan man dela upp Författare i två relationer:

i. **Författare**

namn	författar_id

ii. **DOIFörfattare**

doi	författar_id

I Författare är `namn` FFB av `författar_id`, och i DOIFörfattare har vi inga attribut förutom kandidatnyckeln.

Relationen Artikel bryter också mot 2NF, eftersom den enda kandidatnyckeln är `serie_id, doi` och artikelattributen (titel, tidskrift, datum) är fullständigt beroende av endast DOI. Relationen är visserligen oklar i modellen. Om man antar att det kan vara flera artiklar per serie så kanske påståendet är känns mer självklart, men även om vi begränsar till en artikel per serie så har vi ju grundkunskapen att det bara är DOI som bestämmer artikelattributen. Och en artikel kan ju beskriva flera serier.

För att få modellen på 2NF ska vi därför dela upp Artikel i två relationer:

i. **Artikel**

doi	titel	datum	tidskrift

ii. **SerieArtikel**

serie_id	doi

Nya versionen av Artikel får då DOI som enda kandidatnyckel och de andra attributen är FFB av DOI. Nya relationen SerieArtikel är trivialt på 2NF.

c. För tredje normalform ska relationerna vara på 2NF och *inget icke-nyckelattribut är FFB av något annat icke-nyckelattribut.* Vi kan konstatera att alla beroenden är till nyckelattribut, och därför är den justerade relationsmodellen på 3NF.

5a.
```
CREATE TABLE Heltalsserie (
       serie_id CHAR(7) PRIMARY KEY,
       namn     VARCHAR(20)
       );

CREATE TABLE Heltalsserietal (
       serie_id CHAR(7),
       ordning  INT,
```

```
        tal      INT,
        PRIMARY KEY(serie_id, ordning, tal)
        );

CREATE TABLE Artikel (
        serie_id CHAR(7),
        doi      VARCHAR(50) PRIMARY KEY,
        titel    VARCHAR(100),
        datum    DATE,
        tidskrift VARCHAR(100),
        PRIMARY KEY(serie_id, doi)
        );

CREATE TABLE Forfattare (
        doi VARCHAR(50),
        namn VARCHAR(100),
        forfattar_id VARCHAR(20),
        PRIMARY KEY (doi, forfattar_id)
        );
```

Identifierarna måste vara strängar, pga deras uppbyggnad och i brist på närmare detaljer om dem (kanske kan serieidentifierarna lagras som 6 siffror?). Serieidentifieraren verkar vara strikt begränsad till 7 tecken, men DOI har tyvärr ingen begränsning så vi gör en "ingenjörslösning" där och utgår ifrån att den är som mest 50 tecken. ORCiD är på 16 siffror, men vi kan gå till 20 tecken för ökad flexibilitet.

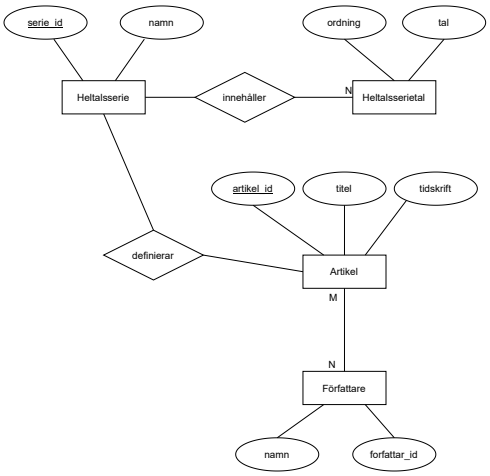
Vi gissar lite på strängarnas längd, men det är inte orimligt att säga att namn är begränsade till 100 tecken, till exempel. Självklart ska ordning och tal vara INT.

För publikationsdatum bör man använda typen DATE, för bättre generalitet i sökningar och berökningar. Om man bara vill göra jämförelser på datum kan man komma undan med att lagra datum som strängar i ISO-format ("2020-01-08"), men det är inte optimalt.

För primärnycklar tänker vi på vad som gör tupler unika. I relationen Heltal så är serieidentifieraren självklart unik, och för Heltalsserietal kombinerar vi ihop med `ordning` och `tal` för att få tupler unika.

För tabellen Artikel beror det på: om vi begränsar oss till en artikel per heltalsserie så räcker det med DOI som primärnyckel. Författartabellen kombinerar DOI med författar-id.

b.



c. För integritetsvillkor lägger jag på NOT NULL på de flest fält. Jag väljer dock att tillåta att tidskrift saknas så att man kan få med manuskript, till exempel. Många fält hanteras i övrigt som del av primärnyckeln.

Det enda extra integritetsvillkor jag ser möjligt är att kräva att ordningstalet är positivt.

```
CREATE TABLE Heltalsserie (
       serie_id CHAR(7) PRIMARY KEY,
       namn     VARCHAR(20) NOT NULL
       );

CREATE TABLE Heltalsserietal (
       serie_id CHAR(7),
       ordning  INT CHECK(ordning > 0),
       tal      INT NOT NULL,
       PRIMARY KEY(serie_id, ordning)
       );

CREATE TABLE Artikel (
       serie_id CHAR(7),
       doi      VARCHAR(50) PRIMARY KEY,
       titel    VARCHAR(100) NOT NULL,
       datum    DATE NOT NULL,
       tidskrift VARCHAR(100)
       );

CREATE TABLE Forfattare (
       doi VARCHAR(50),
       namn VARCHAR(100) NOT NULL,
       forfattar_id VARCHAR(20),
       PRIMARY KEY (doi, forfattar_id)
       );
```

d. För referensvillkor så ska vi lägga till FOREIGN KEY. Vi kan påpeka att alla hänvisningar till en serieidentifieraren ska finnas i Heltalsserie, och

att DOI måste hänvisa till en artikel.

```
CREATE TABLE Heltalsserie (
       serie_id CHAR(7) PRIMARY KEY,
       namn     VARCHAR(20) NOT NULL
       );

CREATE TABLE Heltalsserietal (
       serie_id CHAR(7),
       ordning  INT CHECK(ordning > 0),
       tal      INT NOT NULL,
       PRIMARY KEY (serie_id, ordning),
       FOREIGN KEY (serie_id) REFERENCES Heltalsserie (serie_id)
       );

CREATE TABLE Artikel (
       serie_id CHAR(7),
       doi      VARCHAR(50) PRIMARY KEY,
       titel    VARCHAR(100) NOT NULL,
       datum    DATE NOT NULL,
       tidskrift VARCHAR(100),
       FOREIGN KEY (serie_id) REFERENCES Heltalsserie (serie_id)
       );

CREATE TABLE Forfattare (
       doi VARCHAR(50),
       namn VARCHAR(100) NOT NULL,
       forfattar_id VARCHAR(20),
       PRIMARY KEY (doi, forfattar_id),
       FOREIGN KEY (doi) REFFERENCES Artikel (doi)
       );
```

6a. Uttrycket räknar hur många tal som är registrerade för talserien som heter “Fibonacci”.

b. Vi skapar en vy i databasen, dvs en “virtuell tabell”, från SQL-uttrycket som listar namn på heltalsserie tillsammans med en artikel (via titel och tidskrift) som definierar talserien, för de talserier som har en artikel listad och artikeln är från detta millenium.

c. Här sammanställs, för varje listad tidskrift, hur många artiklar som finns i databasen med namngivna talserier.