#### ****Opsætning af database til håndtering af maskiner og omkostninger****

#### ****Databaseopdeling og 3. normalform****

Databasen følger principperne for **3. normalform** (3NF) for at sikre dataintegritet og minimere redundans. Hver tabel repræsenterer en klart defineret enhed. For eksempel:

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, linje/række

Automatisk genereret beskrivelse

**CREATE TABLE Machine (**

* Denne linje starter oprettelsen af en ny tabel i databasen med navnet Machine.
* Tabellen skal bruges til at organisere og gemme data, der relaterer sig til maskiner.

**MachineID INTEGER PRIMARY KEY AUTOINCREMENT,**

* ***MachineID***: Det er navnet på en kolonne i tabellen, som bruges til at identificere hver enkelt maskine entydigt.
* ***INTEGER***: Denne kolonne skal indeholde heltal.
* ***PRIMARY* *KEY***: Marker denne kolonne som primærnøglen for tabellen. En primærnøgle betyder, at hver værdi i denne kolonne skal være unik, og den må ikke være tom.
* ***AUTOINCREMENT***: Når en ny række tilføjes til tabellen, bliver værdien i denne kolonne automatisk øget med 1, så hver række får en unik identifikator.

**MachineName TEXT NOT NULL,**

* ***MachineName***: Det er navnet på en kolonne, der skal gemme navnet på maskinen.
* ***TEXT***: Denne kolonne skal indeholde tekstdata (string).
* ***NOT NULL***: Dette angiver, at denne kolonne ikke må være tom. Hver række i tabellen skal have en værdi for maskinens navn.

**ProcessType TEXT NOT NULL**

* ***ProcessType***: Det er navnet på en kolonne, der beskriver hvilken type procesmaskinen kan udfører.
* ***TEXT***: Denne kolonne skal også indeholde tekstdata.
* ***NOT* *NULL***: Som for MachineName, skal denne kolonne have en værdi for hver række i tabellen.

Tabellen Machine vil altså:

* Opbevare oplysninger om maskiner, såsom deres unikke ID (MachineID), navn (MachineName), og hvilken type proces de udfører (ProcessType).
* Sikre dataintegritet ved hjælp af en primærnøgle og NOT NULL-begrænsninger.

For nogle af tabellerne har det ikke været tilstrækkeligt med én primærnøgle, da flere maskiner, for eksempel kan bruge flere forskellige materialer (mange-til-mange relation). Derfor anvender vi en kombination af to primærnøgler, som sammenkobles gennem en relationstabel. Et eksempel på dette er:

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Automatisk genereret beskrivelse

***CREATE TABLE MachineMaterialCost (***

* Opretter en ny tabel med navnet MachineMaterialCost.
* Denne tabel fungerer som en **relationstabel**, der forbinder de to tabeller: Machine og Materials.

***MachineID INTEGER,***

* **MachineID**: En kolonne, der refererer til maskinens unikke ID fra tabellen Machine.
* **INTEGER**: Angiver, at værdien skal være et heltal.

***MaterialID INTEGER,***

* **MaterialID**: En kolonne, der refererer til materialets unikke ID fra tabellen Materials.
* **INTEGER**: Angiver, at værdien skal være et heltal.

***Cost REAL***

* **Cost**: En kolonne, der angiver prisen eller omkostningen for at bruge et specifikt materiale med en specifik maskine.
* **REAL**: Bruges til at gemme numeriske værdier med decimaler, som f.eks. en pris eller en kostpris.

***Unit TEXT,***

* **Unit**: En kolonne, der specificerer, i hvilken enhed omkostningerne beregnes (f.eks. "kg")
* **TEXT**: Denne kolonne gemmer tekst (string).

***PRIMARY KEY (MachineID, MaterialID),***

* Angiver en sammensat primærnøgle, der består af både MachineID og MaterialID.
* Dette sikrer, at hver kombination af maskine og materiale er unik i tabellen.
* Den sammensatte primærnøgle er nødvendig, fordi vi har en mange-til-mange-relation:
  + En maskine kan bruge mange forskellige materialer.
  + Et materiale kan bruges af mange forskellige maskiner.
* Tabellen opretter et unikt par for hver relation mellem en maskine og et materiale.

***FOREIGN KEY (MachineID) REFERENCES Machine (MachineID),***

* Opretter en fremmednøgle, der forbinder MachineID i denne tabel til MachineID i tabellen Machine.
* Dette betyder, at MachineID i MachineMaterialCost skal matche en eksisterende MachineID i Machine.

***FOREIGN KEY (MaterialID) REFERENCES Materials (MaterialID)***

* Opretter en fremmednøgle, der forbinder MaterialID i denne tabel til MaterialID i tabellen Materials.
* Dette betyder, at MaterialID i MachineMaterialCost skal matche en eksisterende MaterialID i Materials.

**Hvordan fungerer relationstabellen?**

1. **Mange-til-mange-relation**:
   * Tabelstrukturen løser problemet med mange-til-mange-relationer mellem Machine og Materials:
     + En maskine kan kræve flere forskellige materialer for at fungere.
     + Et materiale kan bruges af flere maskiner.
2. **Sammensat primærnøgle**:
   * Primærnøglen består af både MachineID og MaterialID. Dette sikrer, at der ikke er dubletter for en given kombination af maskine og materiale.
3. **Fremmednøgler**:
   * Fremmednøglerne sikrer, at dataene i relationstabellen altid er valide:
     + MachineID skal eksistere i Machine.
     + MaterialID skal eksistere i Materials.

#### ****Popularisering af tabeller****

Her har vi valgt at popularisere tabellerne ved hjælp af **parametre** frem for direkte INSERT INTO-kommandoer, da denne tilgang giver flere fordele i forhold til sikkerhed og fleksibilitet. Dette betyder, at vi indsætter data ved at bruge placeholders (’?’) og derefter leverer værdierne som parametre. For eksempel:

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype

Automatisk genereret beskrivelse

Ved at bruge placeholders til popularisering af tabeller har vi opnået en mere sikker, fleksibel og effektiv måde at indsætte data på. Dette gør det muligt at tilpasse databasen til ændringer uden behov for omfattende kodeændringer. Samtidig er metoden skalerbar, hvor dataindsættelser sker, hvilket gør den til et fremtidssikret valg.

Til at beregne omkostningerne pr. cm3 har vi været nødsaget til at omregne nogle af de enheder og priser, som var givet fra Excel arket. Dette er gjort i selve databasen som følger:

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Automatisk genereret beskrivelse

**INSERT INTO MaterialCostPerCM3 (MachineID, MaterialID, CostPerCM3)**

* Indsætter data i tabellen MaterialCostPerCM3.
* Der specificeres tre kolonner:
  + MachineID: Refererer til den maskine, der bruger materialet.
  + MaterialID: Refererer til det specifikke materiale.
  + CostPerCM3: Beregnet omkostning pr. kubikcentimeter af materialet.

**SELECT**

* Vælger data, der skal indsættes i MaterialCostPerCM3. Hver valgt række svarer til en række, der bliver indsat.

**mmc.MachineID,**

* Vælger maskinens ID fra MachineMaterialCost (mmc) og bruger det til MachineID i den nye tabel.

**mmc.MaterialID,**

* Vælger materialets ID fra MachineMaterialCost og bruger det til MaterialID i den nye tabel.

**CASE**

* Starter en betinget logik for at beregne CostPerCM3, afhængigt af enhedstypen (mmc.Unit).

**WHEN mmc.Unit = '$/kg' THEN mmc.Cost / (materials.Density \* 1000)**

* Hvis enheden er dollars pr. kilo ($/kg), beregnes kostprisen pr. kubikcentimeter ved at:
  + Dividere prisen (mmc.Cost) med densiteten pr. kubikcentimeter:
    - materials.Density **\*** 1000: Tæthed (i g/cm³) omregnet til kg/m³.

**WHEN mmc.Unit = 'unit' THEN NULL**

* Hvis enheden er unit (stykspris), indsættes NULL for CostPerCM3, fordi beregningen muligvis ikke er relevant eller skal håndteres særskilt.

**WHEN mmc.Unit = '$/L' THEN mmc.Cost / 1000**

* Hvis enheden er dollars pr. liter ($/L), beregnes kostprisen pr. kubikcentimeter som:
  + mmc.Cost / 1000, fordi der er 1000 cm³ i 1 liter.

**WHEN mmc.Unit = '$/10kg' THEN (mmc.Cost / 10) / (materials.Density \* 1000)**

* Hvis enheden er dollars pr. 10 kg ($/10kg), beregnes kostprisen pr. kubikcentimeter som:
  + Først omregnes prisen til 1 kg: mmc.Cost / 10.
  + Derefter divideres det med materials.Density \* 1000.

**ELSE NULL**

* Hvis ingen af de nævnte enhedstyper matcher, indsættes NULL for CostPerCM3.

**END AS CostPerCM3**

* Afslutter CASE-udtrykket og navngiver det beregnede resultat som CostPerCM3.

**FROM MachineMaterialCost AS mmc**

* Angiver tabellen MachineMaterialCost (med alias mmc) som datakilde.

**JOIN Materials ON mmc.MaterialID = Materials.MaterialID**

* Udfører en JOIN mellem tabellerne MachineMaterialCost og Materials:
  + Matcher rækker, hvor MaterialID i begge tabeller er ens.
  + Dette er nødvendigt for at hente materialets tæthed (Density), der bruges i beregningerne.

Efter kørsel af dette stykke kode, vil tabellen MaterialCostPerCM3 indeholde:

* Maskinens ID (MachineID).
* Materialets ID (MaterialID).
* Kostprisen pr. kubikcentimeter (CostPerCM3), beregnet baseret på enheden og materialet densitet

Dette gør det muligt at udføre beregninger af materialeomkostninger på tværs af maskiner og materialer.

Et billede, der indeholder tekst, nummer/tal, skærmbillede, Font/skrifttype

Automatisk genereret beskrivelse

**login database**

**Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Automatisk genereret beskrivelse**

Koden opretter og initialiserer databasen **"manufacturing.db"**, som bruges til at gemme brugere og deres typer (roller). Hvis databasen eller tabellerne ikke eksisterer, oprettes de automatisk. Der indsættes også standarddata (brugere og typer).

**Databaseoprettelse**:

* sqlite3.connect("manufacturing.db") opretter databasen, hvis den ikke allerede findes, og gemmer den lokalt som "manufacturing.db".
* Denne database kan bruges konsistent på tværs af programmet ved blot at referere til dens navn(db\_name).

**Tabeloprettelse**:

* Tabellerne bruger (brugernavne, adgangskoder og type-ID'er) og type (type-ID'er og navne) oprettes, hvis de ikke eksisterer.
* Hvis tabellerne allerede findes, ignoreres oprettelsen.

**Indsættelse af data**:

* Standardbrugere (Admin og Casper) og deres roller (admin og standard) indsættes i tabellerne.
* Hvis data allerede er tilføjet, undgås en fejl, og der udskrives en besked.

Koden bruger **direkte værdier** i stedet for **placeholders** (?), hvilket kan gøre den sårbar over for SQL-injektion, hvis input data i andre tilfælde skulle være dynamiske. Selvom denne kode kun indsætter statiske værdier, ville brug af placeholders gøre den mere robust og fremtidssikret.  
Dette ville:

1. Forbedre sikkerheden ved at beskytte mod potentielle SQL-injektioner.
2. Standardisere koden til at håndtere dynamiske data på en sikker måde.

**Beregner**

For at kunne benytte informationerne fra databasen i vores beregner, skal vi lave en forbindelse mellem vores database og programmet. Dette er gjort på følgende måde:

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, linje/række

Automatisk genereret beskrivelse

**class Database:**

* Definerer en ny klasse ved navn Database. Klassen indeholder metoder til database-forespørgsler.

**def \_\_init\_\_(self, db\_name):**

* Initialiserer en ny instans af Database.
* db\_name er en parameter, der angiver navnet eller stien til databasefilen.

**self.db\_name = db\_name**

* Gemmer databasefilnavnet i instansvariablen self.db\_name for at gøre det tilgængeligt i andre metoder.

Et billede, der indeholder tekst, Font/skrifttype, linje/række, skærmbillede

Automatisk genereret beskrivelse

**def fetch\_dropdown\_data(self):**

* Definerer en metode uden parametre (udover self), der henter data til en dropdown-menu.

**conn = sqlite3.connect(self.db\_name)**

* Opretter en forbindelse til databasen ved hjælp af sqlite3 og den gemte db\_name.
* conn bruges til at kommunikere med databasen.

**cursor = conn.cursor()**

* Cursoren bruges til at udføre SQL-forespørgsler.

**machines = [row[0] for row in cursor.execute("SELECT MachineName FROM Machine")]**

* Udfører en SQL-forespørgsel, der henter MachineName fra tabellen Machine.
* cursor.execute("SELECT MachineName FROM Machine") returnerer maskinnavnet fra tabellen Machine.
* [row[0] for row in ...] opbygger en liste over maskinnavne og henter det første element i hver række, dvs. selve maskinnavnet.

**conn.close()**

* Lukker forbindelsen til databasen.

**return machines**

* Returnerer listen over maskinnavne, der skal bruges i dropdown-menuen i GUI'en.

**Opdatering af drop-down menu baseret på Machine**

Formålet er her at hente de materialer, der er kompatible med den valgte maskine, samt maskinens processtype, og derefter opdatere dropdown-menuerne i GUI'en med disse oplysninger.

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Automatisk genereret beskrivelse

**def fetch\_compatible\_data(self, machine\_name):**

* Definerer en metode, der tager machine\_name som parameter.

**materials = [row[0] for row in cursor.execute(''' ... ''', (machine\_name,))]**

* Udfører en SQL-forespørgsel, der henter navne på materialer kompatible med machine\_name.
* **SELECT Materials.MaterialName**: Vælger kolonnen MaterialName fra Materials-tabellen.
* **FROM MaterialCostPerCM3**: Starter forespørgslen fra MaterialCostPerCM3-tabellen.
* **JOIN Machine ON Machine.MachineID = MaterialCostPerCM3.MachineID**: Forbinder Machine og MaterialCostPerCM3 via MachineID.
* **JOIN Materials ON Materials.MaterialID = MaterialCostPerCM3.MaterialID**: Forbinder Materials og MaterialCostPerCM3 via MaterialID.
* **WHERE Machine.MachineName = ?**: Begrænser resultaterne til den maskine med navnet machine\_name.

**Hentning af omkostninger**

For at forstå, hvordan vores program udfører beregninger, er det vigtigt at se nærmere på, hvordan data hentes fra databasen og bruges som input. Gennem metoden fetch\_cost\_details indsamler vi de nødvendige oplysninger via SQL-forespørgsler. Disse data danner grundlaget for beregningerne.

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Automatisk genereret beskrivelse

**def fetch\_cost\_details(self, query, params):**

* Definerer en metode til at udføre en SQL-forespørgsel, som skal bruges senere.
* **query**: En SQL-forespørgsel som en streng med pladsholdere (?).
* **params**: En tuple af parametre, der skal indsættes i forespørgslen.

**cursor.execute(query, params)**

* Udfører den givne SQL-forespørgsel med de angivne parametre.
* **Parameterisering** sikrer fleksibilitet og beskyttelse mod SQL-injektion.

**result = cursor.fetchall()**

* Henter alle rækker fra resultatsættet og gemmer dem i result.

**conn.close()**

* Lukker forbindelsen til databasen.

**return result**

* Returnerer resultatsættet som en liste af tuples.

### ****Databasen i beregningerne****

Vores program benytter tabellerne i databasen; Disse oplysninger hentes via SQL-queries og bruges direkte i beregningerne. Her er hvordan det fungerer overordnet set:

1. **Henter grundlæggende data**
   * Data som materialets pris pr. cm³, maskinens byggehastighed, og faste omkostninger hentes fra databasen.
   * Disse data bruges som input til beregninger, der kombinerer flere variabler for at finde de samlede omkostninger.
2. **Relationstabeller**
   * Tabellen MaterialCostPerCM3 fungerer som en **relationstabel**, der binder maskiner (Machine) og materialer (Materials) sammen og tilføjer ekstra data (pris pr. cm³).
   * Tabellen BuildRate indeholder byggehastigheder, der er unikke for hver maskine.
   * Tabellen FixedCosts tilføjer opsætnings- og fjernelsesomkostninger.
3. **Kombinering af data**
   * **JOIN** bruges til at kombinere relevante data fra flere tabeller. Dette sikrer, at beregningerne tager højde for flere faktorer.

Denne overordnet arkitektur tillader at programmet er fleksibelt, skalerbart og dynamisk.

**Beregner**

Et billede, der indeholder tekst, Font/skrifttype, linje/række, nummer/tal

Automatisk genereret beskrivelseHer kan vi se at beregneren er opsat som en class der tager data fra databasen, herefter definere en metode til at beregne omkostningerne.

Et billede, der indeholder tekst, Font/skrifttype, nummer/tal, linje/række

Automatisk genereret beskrivelse

Dataene bliver hentet ved hjælpe af SQL-forespørgsler (queries), som vist ovenfor. For at lave selve beregningerne har det været nødvendigt at opsætte nogle regnestykker. Følgende er et eksempel på dette:



**cost\_per\_cm3 = self.database.fetch\_cost\_details(queries["cost\_per\_cm3"], (machine\_name, material\_name))**

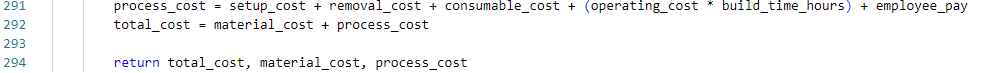
* Udfører SQL-forespørgslen queries["cost\_per\_cm3"] for at hente prisen pr. kubikcentimeter (CostPerCM3) for det angivne maskin- og materialenavn.
* Parametre: (machine\_name, material\_name), som bliver udfyldt længere oppe.

**cost\_per\_cm3 = cost\_per\_cm3[0][0] if cost\_per\_cm3 else 0**

* Hvis cost\_per\_cm3 ikke er tom, hentes værdien fra den første tuple (cost\_per\_cm3[0][0]).
* Hvis forespørgslen ikke returnerer noget, sættes cost\_per\_cm3 til 0.

**material\_cost = cost\_per\_cm3 \* volume\_cm3**

* Beregner den samlede materialeomkostning ved at gange prisen pr. kubikcentimeter med det totale volumen.



Til sidst bliver alle disse mellemregninger samlet for at kunne beregne *procesomkostninger, materialeomkostninger og totaleomkostninger.*