SQL

Stuctured Query Language, spiller roller som

- DDL, definere relationsskemaer m.v.
- DML, Forespørgsler ≈ Relationel algebra + *noget mere!*
  Opdatering af relationer
  af skemaer (overlap m. DDL)

Hvem bruger SQL til hvad?

- Den almindelige bruger af en database???
- Databaseadministrator?
- Udvikler?
  SQL gennem interface til programmeringssprog
  Specialiserede udviklingsværktøjer til grænsefladeudvikling i DB syst.
  SQL scripts

SQL som sprog betragtet:

+ fleksibilitet

+ stor udtrykskraft

+/÷ samme ting kan udtrykkes på 10^6 måder

÷ udtryk kan nemt blive *meget* komplicerede

+ udtryk kan forstås "deklarativt"

÷ beregningsmæssige konsekvenser uoverskuelige for begyndere — og øvede

+ fuldt udrustet til at klare navnesammenfald osv. (modsat semiformel notation for relationel algebra!)

SQL som DDL

Kontekst: SQL-server (f.eks. Oracle) er startet; udvikleren kører et script, som bl.a. indeholder dette her:

```
CREATE TABLE MovieStar (
   name CHAR(30),
   address VARCHAR(255),
   gender CHAR(1),
   birthdate DATE
);
```

Også: Slette tabeller

```
DROP TABLE MovieStar;
```

Ændre tabeller:

```
ALTER TABLE MovieStar ADD phone CHAR(16);
ALTER TABLE MovieStar DROP birthdate;
```

Det ser nemt ud, men potentielt voldsomme konsekvenser.

Behov for “nulværdier”:

- `NULL` særligt element i enhver datatype (domæne)
- Eksplicit default-værdi i skema:

  ```
  CREATE TABLE MovieStar (
     name CHAR(30),
     address VARCHAR(255),
     gender CHAR(1) DEFAULT '?',
     birthdate DATE DEFAULT DATE '0000-00-00'
  );
  ```

I praksis (ofte): Generere helt ny database

SQL som DML: Opdatering

Indsætte:

```
INSERT INTO R(A_1, ..., A_n) VALUES (v_1, ..., v_n);
```

Slette, f.eks.:

```
DELETE FROM StarsIn
WHERE   movieTitle 'The Maltese Falcon' AND
        movieYear 1942 AND
        starName 'Sydney Greenstreet';
```

Ændre felter i (mange) tupler, f.eks.:

```
UPDATE MovieExec
SET name = 'Pres. ' ||  name
WHERE cert# IN (SELECT presC# FROM Studio);
```

I praksis (ofte): Opdatering er *transaktion* = gruppe af opdateringsoperationer, som hører sammen.
Mere generelt: En (programmeret) dialog, som indebærer også opslag i databasen.

OBS: Database til tid k: bestemt ved sekvens af DDL + opdateringer fra tid 0 til tid k
eller database til tid i + sekvens af DDL + opdateringer fra tid i til tid k.

Views

Databasernes svar på programmeringssprogenes funktionsdefinitioner!

`CREATE VIEW` $\langle$*navn*$\rangle$ `AS` $\langle$*SQL-udtryk som denoterer en relation*$\rangle$`;`

Eksempel:

```
CREATE VIEW ParamountMovie AS
  SELECT title, year FROM Movie WHERE studioName = 'Paramount';
```

Semantik *som om*: Hver gang `ParamountMovie` optræder i et udtryk evaluér indmaden til R og lad som om der findes en basisrelation ved navn `ParamountMovie` og med attributter `title`, `year` og indhold R.

OBS: Bogens beskrivelse er meget forvirrende — er ikke en "definition", men beskrivelse af *optimering* af forespørgsel. F.eks.:

Evaluere `SELECT title FROM ParamountMovie WHERE year=1979;`

$\Rightarrow$ Evaluere `SELECT title FROM (SELECT title, year FROM Movie WHERE studioName = 'Paramount') WHERE year=1979;`

$\Rightarrow$ Evaluere `SELECT title FROM Movie WHERE studioName = 'Paramount' AND year = 1979;`

Særligt ved views: Opdateres indirekte, når basisrelationer i deres definition opdateres.

Opdatering *gennem* **views, f.eks.**
`INSERT INTO` *View-rel* (...) `VALUES (...);`

Kun mulig når det kan afgøres på forhånd, at afledt opdatering af basisrelation er *entydig*. Modeksempel:

```
CREATE VIEW V AS
  R UNION S;
```

Fører `INSERT INTO V` ... til opdatering af R, af S eller af begge to???

NB: "View update" generelt beslægtet med "abduktion" (a la Peirce) og studeret i litteraturen som eksempel på sådant.

SQL som DML: Forespørgsler

Udtryk, hvis værdi er relationer; baseret på relationel algebra ... og meget mere. F.eks.

$$\begin{array}{rl}\langle\texttt{relations-udtryk}\rangle ::= & \langle\texttt{relations-udtryk}\rangle\ \texttt{UNION}\ \langle\texttt{relations-udtryk}\rangle\\ & \mid \langle\texttt{relations-udtryk}\rangle\ \texttt{INTERSECT}\ \langle\texttt{relations-udtryk}\rangle\\ & \mid \langle\texttt{relations-udtryk}\rangle\ \texttt{EXCEPT}\ \langle\texttt{relations-udtryk}\rangle\end{array}$$

Generel form i SQL (forenklet):

`SELECT` *attributter* `FROM` *relation*, *relation*, ... `WHERE` *betingelse*;

Semantik (i de fleste tilfælde): Udregn relationerne, udregn deres $\times$-produkt og anvend $\sigma_{betingelse}$ og derefter $\pi_{attributter}$.

Relationelle operatorer for projektion, selektion, renaming, produkt, θ-join kan udtrykkes direkte!

Eksempel på naturlig join for relationer $R(a, b, c)$ og $S(b, c, d)$, $R \bowtie S$:

```
SELECT a,b,c FROM R,S WHERE R.b=S.b AND R.c=S.c;
```

VIGTIGT: Der regnes altid med "bags" undtagen ved `UNION`, `INTERSECT`, `EXCEPT`.
At få dubletter med her: `UNION ALL` osv.
At undertrykke dubletter andre steder: Se bogen.

Variationer og finesser $\hookrightarrow$

SQL-variationer

Implicit konvertering (eng.: coercion) af relation-med-én-søjle-og-én-række til værdi.
Eksempel:

```
SELECT name
FROM MovieExec
WHERE cert# =
   (SELECT producerC#
    FROM Movie
    WHERE title = 'Star Wars'
   );
```

OBS:

- "én-søjle" kan checkes statisk (på "compile time")
- "én-række" er en dynamisk egenskab $\Rightarrow$ run time error!!!!!
  Kendskab til nøglefelter kan i nogle tilfælde udnyttes til statisk check for "højst én-række", men altså ...

SQL-variationer

Mængde-indeholdt $\in$ i betingelser, eksempler:

```
SELECT name
FROM MovieExec
WHERE cert# IN
    (SELECT producerC#
     FROM Movie
     WHERE (title, year) IN
         (SELECT movieTitle, movieYear
          FROM StarsIn
          WHERE starName = 'Harrison Ford)
        )
     );
```

OBS: Bemærk syntaktisk “coercion” af attribut `cert#` til tupel.

Andre operatorer a la `IN`:

1. `EXISTS` R
2. *tupel* `NOT IN` R
3. *værdi* $>$ `ALL` *én-søjle-relation*
   og ditto for “$=$”, “$<$” osv.
4. *værdi* $>$ `ANY` *én-søjle-relation*

   NB: forvirrende i “internationalt-engelsk”,
   havde de dog bare kaldt den `SOME`...

SQL-variation. "Aggregation"

Eksempel. Følgende udtryk evaluerer til et tal:

```
SELECT SUM(månedsløn) FROM Ansatte
```

mon det er den månedlige lønudgift?

Andre operatorer a la `SUM`:

1. `AVG` for gennemsnit
2. `MIN`, `MAX`
3. `COUNT`, `COUNT DISTINCT`

Eksempler:

```
SELECT COUNT(*) FROM MovieExec
SELECT COUNT(DISTINCT name) FROM MovieExec
```

Sej variant af aggregation

Eksempel:

```
SELECT name, SUM(length)
FROM MovieExec, Movie
WHERE producerC# = cert# GROUP BY name HAVING MIN(year) < 1930;
```

Opgave: Hvilke attributter få resultat-relationen? Hvad siger Oracle?

SQL-variationer: “Correlated subqueries”

Tilsyneladende normal brug af `SELECT-FROM-WHERE`, som er aldeles ualgebraisk.
Eksempel: Finde filmtitler som optræder mere end en gang.

```
SELECT title
FROM Movie AS Old
WHERE year < ANY
   (SELECT year
    FROM Movie
    WHERE title = Old.title
   );
```

Hvad er specielt her???

- “`Old.title`” refererer til tupel i yderste `Movie AS Old`
- “`title`” refererer til tupel i inderste `FROM Movie`
- Resultat-relation for indre `SELECT-FROM-WHERE` afhænger af `Old.title` (?!).
- ***Dvs. en “evaluer-indefra-og-ud” semantik giver ikke mening.***
- Vi *definerer* semantikken i dette tilfælde som følger:
  - Løb samtlige tupler i `Movie AS Old` igennem.
  - For hver af dem evaluer `WHERE`-betingelse som afgør om tuplen skal regnes med:
    - test `WHERE title = Old.title` for de *aktuelle* `title` (yderste løkke) og `Old.title` (inderste løkke) for at afgøre om aktuel indre-tupel giver anledning til en “indre” `year` som testes i forhold til der “ydre” `year`

Såre enkelt!

Semantiske reflektioner

"Evaluer-indefra-og-ud" semantik

$\sim$ Definitionen af en algebra

$\sim$ Call-by-value (a la metodekald i Java)

Eksempler på SQL som pseudokode i Java, hvor call-by-value tilstrækkelig:

```
class relation_ABC ...
class relation_BCD ...
class relation_ABCD ...
relation_ABCD join117(relation_ABC R, relation_BCD S) {... return ...}
rel = join117( rel1, rel2)
```

Relationel algebra kan defineres på denne måde, med én metode for hver operation, hvert sæt skemaer og hver muligt test den skal anvendes på.

Eksempel: `SELECT a,b FROM Blib, Blop WHERE c>b and d=a`
kan skrives således i Java: `project117( select35( product(Blip,Blop)))`
eller `select_from_where4012( product(Blip,Blop))`

Men hvad med eksemplet fra før???

```
SELECT title
FROM Movie AS Old
WHERE year < ANY
   (SELECT year
    FROM Movie
    WHERE title = Old.title
   );
```

Kan man skrive funktioner således at dette virker???

```
select_from_where5008( Movie, select_from_where5009( Movie))
```

Nixen! Det kræver call-by-name (a la Algol60 og Simula)!
NB: Kan eftergøres i java ved at sende objekt med udtrykket som metode over som parameter!