Deduktive databaser og
udnyttelse af integritetsbegrænsninger

Deduktive eller logiske databaser

Baseret på 1.-ordens prædikatlogik.

- Alternativ til Rel.Alg. som teoretisk model for (relationelle) databaser. Eller alternativ data-model.
- Databaser betragtet som mængder af logisk formler; "arver" stort og veletableret teoriapparat.
- Generaliserer relationelle databaser med
  - rekursion
  - negation
  - (disjunktion ("der gælder "regn" ELLER "slud"))
- Transformationer på logiske formler nyttige til f.eks.
  - Semantisk optimering af forespørgsler: at ændre på forespørgslen ved at inddrage integritetsbegrænsninger
  - Intensionelle svar
  - Optimering af integritetsbegrænsninger ved "simplification"

I dag:

- Definition og eksempler af logiske databaser
- Opdatering og integritetsbegrænsninger
- Opdateringsrutiner
- Udledning af forenklede integritetsbegrænsninger, "simplification".
- Udledning af opdateringsrutiner fra integritetsbegrænsninger.

+ Afslutning af kurset og noget om mulige specialeemner.

Eksempel på relationel databaseskema

Forenklet skitse:

```
CREATE TABLE E(NAVN, ADRESSE, ...)  % Personer
CREATE TABLE F(NAVN-f, NAVN-b)      % Fædre
CREATE TABLE M(NAVN-f, NAVN-b)      % Mødre

CREATE VIEW P AS E UNION F.          % Forældre
CREATE VIEW S AS SELECT (Navn1, Navn2) FROM  ... % Søskende
CREATE VIEW B AS SELECT ...          % Bedsteforældre
CREATE VIEW FF AS ...                % Forfædre
CREATE VIEW O AS ...                 % Forældreløse
```

Forfædre, som er grundlæggende noget rekursivt kan ikke beskrives i SQL2. SQL3 har konstruktioner til rekursion (ret uelegant; se U&W s. 313ff) En deduktiv database har

- *Prædikater* som afbilder tupler over i sand/falsk; modsvarer relationsnavne
  - *ekstensionelle* givet ved fakta; modsvarer basisrelationer
  - *intensionelle* givet ved regler; modsvarer views
- Fakta, f.eks. $f(john, mary)$
- Regler el. *klausuler*, f.eks. $p(x, y) \leftarrow f(x, y)$
  Implicit alkvantorisering:
  $\forall x, y\ (p(x, y) \leftarrow f(x, y))$
  (alternativt skrevet $\forall x, y\ (p(x, y) \vee \neg f(x, y))$)
  Begrænset negation kan forekomme.
- Semantik givet ved "Lærebog om matematisk logik" med "closed world assumption".
  kan angives som mgd. af alle logisk sande fakta i databasen.

Observation: Alt hvad man kan i relationel algebra kan man også i logiske databaser

Eksempler

$R(A,B) \cap S(A,B)$:

$$\textit{r-fælles-s}(x,y) \leftarrow r(x,y) \wedge s(x,y)$$

$R(A,B) \cup S(A,B)$:

$$\textit{r-fælles-s}(x,y) \leftarrow r(x,y) \vee s(x,y)$$

eller

$$\textit{r-fælles-s}(x,y) \leftarrow r(x,y)$$
$$\textit{r-fælles-s}(x,y) \leftarrow s(x,y)$$

$\pi_A \sigma_{A>B}((R(A,B) \cup S(A,B)))$:

$$\textit{bla-bla}(x) \leftarrow (r(x,y) \vee s(x,y)) \wedge x > y$$

eller

$$\textit{r-fælles-s}(x,y) \leftarrow r(x,y) \wedge x > y$$
$$\textit{r-fælles-s}(x,y) \leftarrow s(x,y) \wedge x > y$$

$R(A,B,C,D) \bowtie S(C,D,E,F)$:

$$\textit{r-join-s}(x_1,x_2,x_3,x_4,x_5,x_6) \leftarrow r(x_1,x_2,x_3,x_4) \wedge s(x_3,x_4,x_5,x_6)$$

Eksempel på logisk database

$$
\begin{aligned}
&e(john)\\
&e(mary)\\
&e(jane)\\
&e(peter)\\
&e(paul)\\
&f(john, mary)\\
&m(jane, mary)\\
&p(x, y) \leftarrow f(x, y)\\
&p(x, y) \leftarrow m(x, y)\\
&s(x, y) \leftarrow x \neq y \wedge p(z, x) \wedge p(z, y)\\
&b(x, y) \leftarrow p(x, z) \wedge p(z, y)\\
&\mathit{ff}(x, y) \leftarrow p(x, y)\\
&\mathit{ff}(x, y) \leftarrow p(x, z) \wedge \mathit{ff}(z, y)\\
&o(x) \leftarrow e(x) \wedge \neg(\exists y\, f(y, x)) \wedge \neg(\exists y\, m(y, x))
\end{aligned}
$$

En database DB definerer et begreb af logisk sandhed $DB \models \phi$ for logisk formel ϕ ... som for *denne slags formler* kan beregnes ved

> M:= mængden af fakta i databasen;
> så længe M vokser, udfør
> M:= $M \cup$ {ethvert faktum H som fås ved at indsætte
> værdier i en regel $H \leftarrow B$ således at B er
> tilfredsstillet af M}

Eksempel:

$M := \{e(john), e(mary), e(jane), e(peter), e(paul), f(john, mary), m(jane, mary)\}$

$M := M \cup \{p(john, mary), p(jane, mary), o(john), o(jane), o(peter), o(paul)\}$

$M := M \cup \{\mathit{ff}(john, mary), \mathit{ff}(jane, mary)\}$

Et krav: Klausuler skal være "range restricted". Flg. er ikke gode:

$$p(x) \leftarrow \neg q(x) \qquad p(x) \leftarrow x \neq b\o rge$$

Integritetsbegrænsninger i deduktive databaser

Udsagn som altid skal gælde i database, typisk "denials"

$$DB \models \leftarrow \phi$$

"ϕ må ikke gælde"; læses som *falsk* $\leftarrow \phi$, dvs. regel med tomt hovede.

Eksempler på "generelle" integritetsbegrænsninger

$$\leftarrow f(x,z) \wedge f(y,z) \wedge x \neq y$$
$$\leftarrow m(x,z) \wedge m(y,z) \wedge x \neq y$$
$$\leftarrow f(x,y) \wedge m(x,z)$$
$$\leftarrow \mathit{ff}(x,y), \mathit{ff}(y,x)$$

Eksempler på "referential integrity constraints"

$$\leftarrow f(x,y) \wedge (\neg e(x) \vee \neg e(y))$$
$$\leftarrow m(x,y) \wedge (\neg e(x) \vee \neg e(y))$$

For given database kan integritetsbegrænsninger evalueres som var de forespørgsler. Giver de "tomt svar" er alt OK!

Påstand

Disse overordnede integritetsbegrænsninger er langt nemmere at at forstå og overvbevise sig om er "de rigtige" end en samling SQL triggers, og såkaldte constraints drysset ud over en SQL database!

OBS:

De fleste ville kunne implementeres som SQL "assertions" – men et problem med effektivitet!
Og ingen kobling til transaktioner eller "hjælp" til at vedligeholde konsistens.

Opdatering med integritetsbegrænsninger

For nemheds skyld: Kun opdatering ved tilføjelse; sletning og ændring af enkelt-attribut kan "simuleres", men bør integreres ...

Definition: *opdatering* eller *forslag til opdatering* (eng.: *"update (request)"*) er en mængde af ekstensionelle fakta;
Det antages, at en opdatering overlapper ikke med den aktuelle database.

Logisk sandhed i aktuel DB	Logisk sandhed i opdateret DB
$DB \models \phi$	$DB \cup U \models \phi$

Hvad skal vi med integritetsbegrænsninger???

- Evaluere dem inden "fysisk" opdatering finder sted — så effektivt som muligt.
- Blot afvise forslag til opdatering hvis inkonsistens?
- Eller opdateringsrutiner, som
  1) så vidt muligt tager foreslået opdatering for gode varer,
  2) hjælper bruger(program) til at genoprette konsistens

Traditionel RDBMS: Designer/programmør skal lave det hele selv (implementere en masse småp stumper) og må selv sikre sig, at det hele virker overordnet korrekt.

Forslag:

Programmør/designer angiver:

- Overordnede integritetsbegrænsinger (a la vores eksempler)
- Hvilke relationer, en bruger have dialog for at kunne opdatere (og hvordan) + hvilke relationer, systemet kan opdatere.

Eller:

- Programmør/designer udvikler dialog med interaktivt værktøj, som checker "at logikken holder" og giver mulige forslag.

Simplification

Idé: Vi antager aktuel DB-tilstand er konsistent — så hvorfor beregne en "hel" integritetsbegrænsning om næste tilstand??

Definition: Lad ϕ være en formel, DB en vilkårlig database med $DB \models \phi$ og U en opdatering i forhold til DB. En formel ψ kaldes en *simplification* af ϕ såfremt

$$DB \cup U \models \phi \text{ hvis og kun hvis } DB \models \psi$$

Eksempel:

$$\phi = (\leftarrow f(x,z) \land f(y,z) \land x \neq y)$$

$$U = \{f(børge, peter)\}$$

Intuitiv argumentation:

- Vi har checket, at ϕ holder for alle kendte individer.
- Formlen $f(børge, peter)$ giver ny far til *peter* ...og den eneste måde ϕ kan ødelægges
- og den eneste måde ϕ kan ødelægges er, hvis *peter* havde en far i forvejen (som ikke er *børge*
- Ergo er følgende en simplification: $\leftarrow f(x, peter) \land x \neq børge$
- Og da pr. antagelse har $DM \not\models f(børge, peter)$ kan vi droppe testet: $\leftarrow f(x, peter)$ — kan bruges som simplification.

Tidsforbrug (uden indeksering):

- ϕ kræver tid $\mathcal{O}(n^2)$
- $\leftarrow f(x, peter)$ kræver tid $\mathcal{O}(n)$.

Mere avanceret brug af simplification- og residual formler ved opdatering

Betragt nu integritetsbegrænsning

$$\leftarrow f(x,y) \wedge (\neg e(x) \vee \neg e(y))$$

og opdateringen

$$U = \{f(børge, peter)\}$$

Som simplification kan bruges:

$$\leftarrow \neg e(børge) \vee \neg e(peter)$$

kan omskrives til to formler.

$$\leftarrow \neg e(børge)$$

$$\leftarrow \neg e(peter)$$

Begge skal være opfyldt, dvs. som forespørgsler fejle.

Antag aktuel database DB med

$$e(børge) \notin DB, \qquad e(peter) \in DB$$

(Simplificeret) integritetsbegrænsning er overtrådt.
Afvise opdateringen helt og totalt?

(Foreløbig) Definition: For given DB, opdatering U og simplificeret int.begr. ϕ (mængde af klausuler), definerer vi *fejl-residualen* under updateringen til at være den delmængde af ϕ, som ikke er tilfredsstillet.

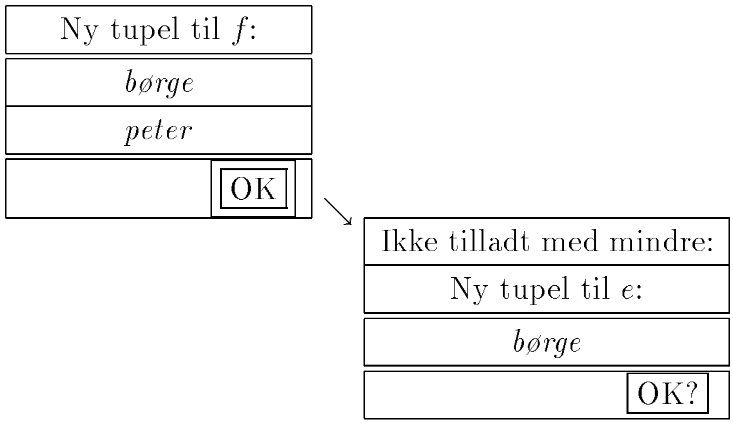
I eksemplet: $\leftarrow \neg e(børge)$.

OBS: Kan vi vi udvide opdatering, så residualen bliver opfyldt (uden at ødelægge noget andet), så er opdatering+konsistens mulig.

I eksemplet: tilføj $e(børge)$ og $\leftarrow \neg e(børge)$ bliver glad!

Eksemplet fortsat

Udnyttelse af fejl-residualer i opdatering gennem brugerdialog:



Eller ved "kaskadeagtig" automatik?

Det kan databasedesigneren beslutte i dialog med sig (endnu tænkte) udviklingssystem.

NB: Bemærk en generalisering er nødvendig for at foretage dette *statisk*:

Betragt "*børge*" og "*peter*" som *formelle parametre*,
dvs. som en slags variable, som instantieres på udførelsestidspunktet

Skitse til automatisk konstruktion af simplifications

Af tekniske årsager (og ikke helt *wlog*): Opdatering er her tilføjelse af ét nyt faktum $q(a, b)$.

Betragt updatering U og int.begr. ϕ, og definér

- $\text{after}^U(\phi)$: Erstat i ϕ enhver $q(t_1, t_2)$ med $q(t_1, t_2) \vee \langle t_1, t_2 \rangle = \langle a, b \rangle$
- $\text{normalize}^{DB}(\Gamma)$: Reducer en samling formler Γ på passende vis vha. alm. logiske regler + viden om aktuel database.
- $\text{normalize}^{DB}(\Gamma)$: Reducer en samling formler Γ på passende vis vha. alm. logiske regler + viden om aktuel database.
- normalize(Γ): Reducer en samling formler Γ på passende vis vha. alm. logiske regler *uden* viden om nogen aktuel database.
- Remove-subsumed$^\phi(\Gamma)$: Fjern formler fra Γ som "umiddelbart" følger af ϕ.

Påstand: Remove-subsumed$^\phi$(normalize(after$^U(\phi)$)) er en simplification.

Eksempel:

$$\begin{aligned}\phi &= \{\leftarrow f(x,y) \wedge (\neg e(x) \vee \neg e(y))\}\\ &\equiv \{\leftarrow f(x,y) \wedge \neg e(x),\ \leftarrow f(x,y) \wedge \neg e(y)\}\\ U &= \{f(b\o rge, peter)\}\end{aligned}$$

$$\begin{aligned}\text{after}^U(\phi) = \{&\leftarrow (f(x,y) \vee \langle x,y\rangle = \langle b\o rge, peter\rangle) \wedge \neg e(x),\\ &\leftarrow (f(x,y) \vee \langle x,y\rangle = \langle b\o rge, peter\rangle) \wedge \neg e(y)\}\end{aligned}$$

normalize(...) = ... 1) udrul "$\vee$" til flere klausuler, og
2) udradér ligninger

$$\begin{aligned}1)\ &\leftarrow f(x,y) \wedge \neg e(x)\\ &\leftarrow \langle x,y\rangle = \langle b\o rge, peter\rangle \wedge \neg e(x)\\ &\leftarrow f(x,y) \wedge \neg e(y)\\ &\leftarrow \langle x,y\rangle = \langle b\o rge, peter\rangle \wedge \neg e(y)\end{aligned}$$

$$\begin{aligned}2)\ &\leftarrow f(x,y) \wedge \neg e(x)\\ &\leftarrow \neg e(b\o rge)\\ &\leftarrow f(x,y) \wedge \neg e(y)\\ &\leftarrow \neg e(peter)\end{aligned}$$

Remove-subsumed$^\phi$(...) sletter det som allerede er givet ved ϕ

Ergo:

$$\begin{aligned}\text{Simp}^U(\phi) = &\ \text{Remove-subsumed}^\phi(\text{normalize}(\text{after}^U(\phi))) =\\ &\{\leftarrow \neg e(b\o rge),\ \leftarrow \neg e(peter)\}\end{aligned}$$

Afslutning på det formalistiske

- after^U (og analog before^U) kan vi transformere formler frem og tilbage i tid,
- og sammen med normalize og Remove-subsumed har vi værktøj til (ca.) de transformationer vi har brug for.
- Formalisering ang. "residualformler" mangler endnu.
- Den hårde nyser: At kunne afgøre af afledte opdateringer for at hjælpe én int.begr. ikke ødelægger det for andre!!!

Mulige specialeemner

Forbedre eller udvide eksisterende databaseteknologi: Simplification, generering af opdateringsrutiner osv.

Logiske/relationelle databaser med ontologibaseret forespørgselsevaluering

$$
\begin{array}{ccccccccc}
 & & \text{ting} & & & & & & \\
 & \nearrow & & \nwarrow & & & & & \\
\text{små ting} & & & & \text{dyr} & & & & \\
 & \nwarrow & & \nearrow & & \nwarrow & & & \\
 & & \text{små dyr} & & & & \text{kæledyr} & & \\
 & & & \nwarrow & & \nearrow & & \nwarrow & \\
 & & & & \text{små kæledyr} & & & & \text{store kæledyr} \\
 & & & \nearrow & & \nwarrow & & & \\
 & & \text{hundehvalpe} & & & & \text{kattekillinger} & &
\end{array}
$$

Tilføjelse af fuzzy vægtning, opdagelse af ontologier, ontologisk indeksering af tekstdokumenter, ...
Se http://www.ontoquery.dk

... eller andre former for "fleksible" forespørgsler

Datamining: Forbedre eller udvide eksisterende teknologier

Anvendelse af databaser til videnskabelige formål, hvor det ikke bare er at hælde data ind

— men ikke nok at lave en udvidet afleveringsopgave!

Opsummering af databasekurset

Dette er kun en introduktion, men du har nok til at:

- starte i jobs med databaser; med erfaring + videreuddannelse kvalificere dig til f.eks. avanceret bruger, databaseudvikler, databaseadministrator,
- starte på special om eller med databaser; f.eks. kvalificere til forskeruddannelse eller avancerede udviklingsprojekter (datamining eller anden db-teknologi, ...)

Kursusindholdet

Forsmag på det hele med vægt på relationelle databaser:

- Teori: Mængdelære og relationel algebra. Funktionelle afhængigheder. Normalformer. Integritetsbegrænsninger
- Erfaring med praktiske systemer: SQL, ORACLE, PL/SQL.
- Kort intro til problematikken omkring transaktioner og samtidige brugere.
- Afleveringsopgaven: At få det hele slået fast!
- Forsmag på forskning indenfor databaser (i dag)

Dog ikke implementation af databasesystemer (søgetræer, hashing, strategier for effektiv diskaccess, osv., osv., osv.) — nok værd at sætte sig ind i for den som *vil* noget med databaser.

Indrømmet:

Det havde været smart med manual til Oracles SQL og PL/SQL!!!!!