Gagnasafnsfræði

Páll Melsted

4. nóv

Færslur

2 aðalatriði sem færslur gefa

1. Óskiptanleiki: verndun gegn áföllum, gagnagrunnur er alltaf í löglegu ástandi
2. Raðbinding: kemur í veg fyrir árekstra milli tveggja færslna.

Hvernig er þetta útfært?

Dæmi

Ef við setjum keyrum

```
INSERT INTO Product VALUES ('Apple', 101, 'Laptop');
INSERT INTO Laptop VALUES (101, ...);
```

og það tekst bara að keyra fyrri INSERT skipunina þá töpum við gögnum og gætum hafa skilið gagnagrunninn í hálfkláruðu ástandi, því það er engin fartölva með númerið 101.

Dæmi

Í síðustu heimadæmum átti bara að setja inn nýja vöru ef varan var ekki til. Þetta var gert í tveimur skrefum:

1. Keyra

   ```
   SELECT COUNT(model) FROM Product WHERE model=?
   ```

2. Ef niðurstaðan var 0 þá keyra

   ```
   INSERT INTO ...
   ```

Ef tveir keyra skipun 1 á sama tíma og fá niðurstöðuna 0 þá gætu þeir báðir reynt að setja inn í gagnagrunninn. En það ætti aðeins önnur tilraunin að heppnast, þ.e. það þarf að pakka þessum tveimur skipunum í eina færslu (og helst keyra með Serializable)

Óskiptanleiki

Á meðan verið er að breyta gögnum í færslu og áður en `COMMIT` er framkvæmt er haldið utan um breytingar í `log` sem heldur utan um breytingar.

Ein einföld leið til að halda utan um log er að skrá hvaða breytingar hafa verið framkvæmdar.

Ef færslan klárast ekki eða `ROLLBACK` er framkvæmt þá förum við í log skránna og tökum allt til baka, þetta kallast UNDO logging.

UNDO

Við þurfum að halda utan um allar breytingar fyrir hverja færslu í log skrá. Þar sem log skráin þarf að lifa af þá þurfum við reglur um í hvaða röð má skrifa á disk

1. Ef færsla T breytir X úr v verður að skrifa (T, X, v) í log skrá **áður** en nýja gildið er skrifað á disk.
2. Ef færsla T framkvæmir `COMMIT` þarf að klára að skrifa allar breytingar áður en COMMIT er skráð fyrir T í log skrá.

Dæmi

Ef $M = (A, B)$ er tafla með 1 gildi $(8, 8)$ og T er færslan

```
BEGIN TRANSACTION
  UPDATE M SET A=A*2;
  UPDATE M SET B=B*2;
COMMIT;
```

Ef færslan klárast ekki þá þarf að skoða log skrána og sjá hvaða breytingar hafa verið framkvæmdar og færa þær aftur í fyrra horf (UNDO).

REDO log

UNDO log aðferðin skrifar óþarflega mikið á disk. Við getum ekki skrifað COMMIT fyrr en allar eru komnar á sinn stað.

REDO log aðferðin geymir breytingar á gagnagrunninum í minni og klárar að skrifa í log skrá áður en breytingar fara á disk. Við hrun þarf því að fara í gegnum log skrá og klára breytingar sem komust ekki á disk.

Flestir gagnagrunnar nota bæði UNDO og REDO, sqlite notar UNDO og REDO með `PRAGMA journal_mode=WAL`

Raðbinding

Notum sama dæmi og áður en nú eru tvær færslur. T_1 er

```
BEGIN TRANSACTION
  UPDATE M SET A=A+2;
  UPDATE M SET B=B+2;
COMMIT;
```

og T_2 er

```
BEGIN TRANSACTION
  UPDATE M SET A=A*2;
  UPDATE M SET B=B*2;
COMMIT;
```

	Mimi		Disk		Log
Adım	A	B	A	B	
Begin Tr.			8	8	<Begin, T>
read(A)	8		8	8	
A ← A * 2	16		8	8	
Log(A, 8)	16		8	8	<T, A, 8>
write(A, 16)	16		16	8	
read(B)	16	8	16	8	
B ← B * 2	16	16	16	8	
log(B, 8)	16	16	16	8	<T, B, 8>
write(B, 16)	16	16	16	16	
Commit	16	16	16	16	<T, commit>
clear log(T)					

Færslurnar tryggja að T_1 og T_2 fléttast ekki en við getum ekki ákvarðað hvor keyrir fyrst.

Lásar

Til að ákvarða hvor færslan á að keyra þarf að nota lása. Færsla má ekki breyta gildi nema að hún læsi stakinu fyrst og aflæsi þegar hún er búin.

Þetta þýðir að við lestur þarf líka að biðja um lás.

Ef að lásinn er tekinn, þ.e. önnur færsla hefur læst staki (og ekki enn aflæst), þá fáum við neitun og verðum að reyna aftur.

Tveggja fasa lás

Færsla eru tveggja fasa ef beðið er um alla lása áður en þeim er sleppt.

Ef allar færslur eru tveggja fasa þá er tryggt að raðbinding virkar, þ.e. við getum látið sem að aðeins ein færsla sé keyrð í einu.

. . .

Tveggja fasa lás tryggir aðeins að engin mistök eigi sér stað ef færslurnar klára.

Sjálfhelda

Í dæminu að ofan ef T_1 biður um lás á A og T_2 biður um lás á B þá fá báðar færslurnar lásinn.

Þær festast þarna því hvorug getur náð í lás á hinn dálkinn.

Gagnagrunnurinn verður að höndla þessi tilfelli og skera á hnútinn.

SQL yfirlit

Við sleppum nokkrum köflum en það er gott að líta yfir þá upp á framtíðina

- Kafli 9.2, SQL Environment
- Kafli 9.4, Stored Procedures
- Kafli 10.1, Security and User Authorization in SQL
- Kafli 10.2, Recursion in SQL
- Kafli 10.6, OLAP
- Kafli 10.7, Data Cubes

Hálfformuð gögn

Hingað til höfum við eingöngu fengist við gagnagrunna sem töflur, þ.e. venslalíkanið.

Sum gögn passa ekki nógu vel inn í vensl en hafa samt einhvern strúktúr.

Í stað þess að troða slíkum gögnum í töflur er betra að láta gögnin lýsa sér sjálf.

Hálfformuð gögn

Grunnlíkanið fyrir hálfformuð gögn (semistructured data) er safn af hnútum þar sem hver hnútur er annað hvort lauf eða innri hnútur.

- Laufhnútar eru með gögn, t.d. heiltölur strengi
- Innri hnútar eru með leggi yfir í aðra hnúta, leggirnir eru merktir
- Einn innri hnútur, rótin, hefur engar tengingar inn og það er hægt að komast frá rót yfir í alla aðra hnúta.

Dæmi

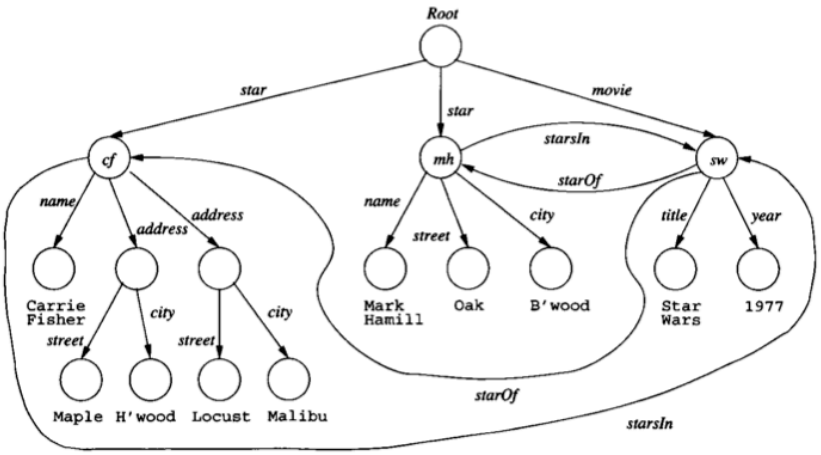

Figure 1:

Tengingar

Við túlkum tengingar annað hvort sem tengsl milli tveggja staka (t.d. StarsIn) eða sem hluta af stærri heild (city sem hluti af address).

Netið þarf ekki að vera tré, en er það yfirleitt ef engin tengsl eru á milli hnúta.

XML

XML er staðall fyrir hálfformuð gögn sem táknar netið á línulegu formi.

- Tög í XML eru afmörkuð með `<...>` eins og í HTML (HTML er ekki afbrigði af XML).
- Tög eru opnuð með `<Foo>` og lokað með `</Foo>` allt sem er á milli er kallað stak (element)
- Sum tög hafa engin stök og því hægt að skrifa `<Foo/>`
- Tög geta haft eiginleika (attribute) skrifaðir sem `<Foo attribute="value" ... >`

Löglegt XML

XML getur verið löglegt eitt og sér (well-formed XML) eða verið löglegt miðað við skema (valid XML). Skema fyrir XML skilgreinir hvaða tög eru leyfð og hvernig þau mega hreiðrast.

Löglegt XML

```
<?xml version="1.0" encoding="utf-8" ?>
<Foo>
 ...
</Foo>
```

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<Foo>
 ...
</Foo>
```

- `<?xml ... ?>` gefur upplýsingar um XML skjalið að
  - það er XML skv. útgáfu 1.0
  - textakóðinn sem er notaður er utf-8.
- Aðeins eitt rótartag
- Öll tög sem eru opnuð þurfa að lokast
- Áður en tagi er lokað þurfa öll tög hreiðruð innni að vera lokuð (eins og lögleg svigasetning).

```
<?xml version="1.0" encoding="utf-8" ?>
<StarMovieData>
    <Star>
        <Name>Carrie Fisher</Name>
        <Address>
            <Street>123 Maple St.</Street>
            <City>Hollywood</City>
        </Address>
        <Address> ... </Address>
    </Star>
```

```
    <Star>
        <Name>Mark Hamill</Name>
        <Address>
            <Street>456 Oak Rd.</Street>
            <City>Brentwood</City>
        </Address>
    </Star>
    <Movie>
        <Title>Star Wars</Title>
        <Year>1977</Year>
    </Movie>
</StarMovieData>
```

Eiginleikar

Við getum sett upplýsingar í laufhnúta eða sem eiginleika (attribute).

Movie hefur Year og Title. Við gætum t.d. gert

```
<Movie>
    <Title>Star Wars</Title>
    <Year>1977</Year>
</Movie>
```

eða

```
<Movie year="1977">
    <Title>Star Wars</Title>
</Movie>
```

eða jafnvel

```
<Movie year="1977" title="Star Wars"/>
```

Röð eiginleika skiptir ekki máli en innbyrðis röð taga í sömu hreiðrun skiptir máli.

Eiginleikar

Helsti munurinn er að við höfum meira frelsi í laufhnútum en notum eiginleika fyrir lykla, ytri lykla og valkvæða eiginleika (optional).

Til að tengja þvert yfir tréð, eins og í starsIn, getum við notað eiginleika. Einn eiginleiki verður lykill, yfirleitt kallaður `id`, og annar vísar til þessa `id`.

```
<Star id="cf" starsIn="sw">
    <Name>Carrie Fisher</Name>
</Star>
<Star id="mh" starsIn="sw">
    <Name>Mark Hamill</Name>
</Star>
<Movie id="sw" starsOf="mh,cf">
    <Title>Star Wars</Title>
    <Year>1977</Year>
</Movie>
```

. . .

Þetta er samt frekar klunnalegt

Eiginleikar og ytri lyklar

Skárri lausn

```
<Star id="cf" starsIn="sw">
    <Name>Carrie Fisher</Name>
</Star>
<Star id="mh" starsIn="sw">
    <Name>Mark Hamill</Name>
</Star>
<Movie id="sw">
    <Title>Star Wars</Title>
    <Year>1977</Year>
    <Roles>
        <starRef>mh</starRef>
        <starRef>cf</starRef>
    </Roles>
</Movie>
```

Eins væri hægt að telja upp hlutverk fyrir hverja stjörnu.

Ytri lyklar

Hvernig er hægt að tryggja að XML skjal sé á fyrirfram ákveðnu formi?

DTD - Document Type Definition er staðall til að segja til um skema fyrir XML skjal. Gegnir sama hlutverki og `CREATE TABLE` í SQL.

DTD er með ytri lykla en er ekki nógu öflugt til að segja til um hvert þeir eiga að vísa.

XSD - XML Schema Definition leyfir sterkari tengingar.