Gagnasafnsfræði

Páll Melsted

21. okt

Færslur

Oft eru margir notendur að vinna á sama gagnagrunni, samtímis

- Ólíkir notendur í sama fyrirtæki
- Vefir með margar tengingar
- Eitt forrit með marga þræði

Gagnagrunnurinn verður að "fela" þetta rétt eins og hver tenging væri ein í heiminum.

Færslur

Í SQL er þetta falið með færslum (transaction)

- Sérhver færsla sem breytir gagnagrunni klárar að öllu eða engu leyti
- Verður að þola áfall, t.d. rafmagnstap
- Þarf að skrifa beint á disk

Í sqlite skelinni verður hver insert/update skipun að sér færslu.

Árekstrar

Þegar nokkrar aðgerðir hafa áhrif á gagnagrunninn og eru innbyrðis háðar er hætta á að notendur skemmi hver fyrir öðrum.

`Midar(eventId, date, seat, status)`

Í vefkerfi væri framkvæmt

```
SELECT * FROM Midar
WHERE eventId=? AND date=? AND status='free';
```

til að sýna laus sæti og þegar einhver kaupir sæti þá verður keyrt

```
UPDATE Midar SET status='gone'
WHERE eventId=? AND date=? AND seat=?;
```

Raðbinding

Lausnin er að nota raðbindingu (serialization) til að tryggja að tveir notendur rekist ekki á. Við setjum aðgerðirnar í eina færslu og gagnagrunnurinn tryggir að engir tveir geti breytt (og lesið breyttar töflur) á sama tíma.

Reyndar er óhóflegt að læsa gagnagrunninum fyrir vefkerfi þar sem of langur tími getur liðið á milli select og update.

Óskiptanleiki

Fyrir bankareikninga

`Accounts(id,balance)`

Ef við millifærum 100 kr. með t.d.

```
UPDATE Accounts SET balance=balance+100
WHERE id=456;

UPDATE Accounts SET balance=balance-100
WHERE id=123;
```

Þá er hætta á tapi ef einhver villa í tölvubúnaði á sér stað. Við viljum ekki að peningar verði til eða tapist.

Færslur

Við getum hópað saman aðgerðir, INSERT, UPDATE, SELECT o.s.frv. í færslur.

Færslur

- byrja með `START TRANSACTION`
- ljúka með `COMMIT`
- hætta við með `ROLLBACK`

```
BEGIN TRANSACTION;
UPDATE Accounts SET balance=balance+100
WHERE id=456;

UPDATE Accounts SET balance=balance-100
WHERE id=123;
COMMIT;
```

Lesfærslur

Færslur sem lesa einungis gögn, þ.e. SELECT, geta látið gagnagrunninn vita með því að nota `SET TRANSACTION READ ONLY` (ekki til í sqlite) og gagnagrunnurinn getur keyrt lesfærslur samhliða.

Að öðru leyti gerir gagnagrunnurinn ráð fyrir að færslur muni líka breyta gögnum.

Einangrun

Í miðasöludæminu er enn möguleiki á árekstri ef lestur hjá B kemur rétt á undan skrif í gagnagrunn hjá A.

Allur lestur úr gagnagrunni er úr ástandinu eins og það var eftir að síðustu færslu lauk. Í sumum tilfellum væri betra að lesa úr öðrum færslum sem hafa ekki verið skrifaðar á disk. Slíkur lestur kallast skítugur (dirty read).

Til að leyfa skítugan lestur þarf að keyra

```
SET TRANSACTION READ WRITE ISOLATION LEVEL READ UNCOMMITTED;
```

í sqlite er þetta gert með

```
PRAGMA read_uncommitted = ON/OFF;
```

Einangrun

Áhættan við skítugan lestur er að þau gögn sem við lesum þurfa ekki nauðsynlega að enda í gagnagrunninum, t.d. ef ROLLBACK er keyrt.

Kostirnir eru

- Aukinn hraði þar sem gagnagrunnurinn þarf ekki að koma í veg fyrir skítugan lestur
- Meiri samhliða keyrslur þar sem ekki þarf að bíða eftir öðrum færslum.

Raðbinding

Í flestum tilfellum viljum við einungis nota `SERIALIZABLE`, þ.e. þar sem færslur læsa gagnagrunninum (eða hluta af honum) og virka eins og allir séu einir í heiminum.

Það er einungis réttlætanlegt að slaka á þessum skorðum ef það skiptir máli upp á hraða og við getum tryggt að ekki hljótist skaði af.

Tengingar við gagnagrunna

Í flestum tilfellum eru gagnagrunnar tengdir við önnur forrit, t.d. vefkerfi.

Forrit opna tengingar við gagnagrunn, keyra skipanir og meðhöndla niðurstöður.

Skoðum dæmi um hvernig þetta er gert í Python og Java yfir í sqlite.

Python

Stuðningur við sqlite er innbyggður í python með sqlite3 pakkanum.

```
import sqlite3

conn = sqlite3.connect('wc.db')
c = conn.cursor()

c.execute("SELECT Teams.name FROM Teams,Groups "
    + "WHERE groupId=Groups.id AND Groups.name='B'")

print c.fetchall()

conn.close()
```

Python

Til að keyra skipanir notum við

- cursor.execute(sql) fyrir eina skipun
- cursor.executemany(sql,parameters) til að keyra margar skipanir fyrir svipuð gögn
- cursor.executescript(sql) fyrir nokkrar skipanir, t.d. CREATE TABLE

Við notum executemany ef við ætlum að framkvæma t.d. margar INSERT skipanir þar sem executemany pakkar öllu inn í eina færslu.

Python

```
c.execute("SELECT code FROM Teams WHERE name='%s'"%(teamName,))
```

. . .

ALDREI NOKKURN TÍMANN GERA ÞETTA!!!!!

. . .

Hvað ef `teamName="'; DROP TABLE Teams; --"` ?

Þegar þið vinnið með gagnagrunna verðið þið að læra að hreinsa strengi og aldrei treysta neinu sem þið skrifuðuð ekki sjálf.

```
c.execute("SELECT code FROM Teams WHERE name=?",teamName)
```

Python sér um að setja breytuna í staðin fyrir `?` og hreinsa strengi.

Python

```
x = [(1,2),(3,4)]
c.executemany("INSERT INTO R(key,value) VALUES (?,?)",x)
```

Keyrir INSERT skipun fyrir hvert stak í x.

Python

Til að fá gögn til baka notum við

- cursor.fetchone() til að fá næstu niðurstöðu
- cursor.fetchall() til að fá restina
- cursor.___iter___() leyfir okkur að skrifa einfaldar for lykkjur

Hvert fetchone skilar `tuple` af gildum eða `None` þegar ekkert er eftir.

fetchall skilar `list` af `tuple` eða tómum lista ef ekkert fannst.

```
c.execute("SELECT Teams.name,code FROM Teams,Groups"
    + "WHERE groupId=Groups.id AND Groups.name='B'")

for name,code in c:
    print code,name
```

Java

Java notar JDBC fyrir gagnagrunnstengingar. Til að setja allt upp þarf að ná í driver fyrir sqlite.

Til að keyra java forritið þarf driverinn að vera í classpath

```
java -classpath ".:sqlite-jdbc-3.7.2.jar" Sample
```

Java

JDBC skilgreiningarnar eru í `java.sql.*` og til að nota réttan driver þurfum við að gera `Class.forName("org.sqlite.JDBC");`

Við fáum tengingu við gagnagrunninn með

```
Connection connection
  = DriverManager.getConnection("jdbc:sqlite:wc.db");

Statement statement = connection.createStatement();
statement.setQueryTimeout(30);

ResultSet rs = statement.executeQuery("SELECT Teams.name,code "
  + "FROM Teams,Groups WHERE groupId=Groups.id AND Groups.name='B'");

while (rs.next()) {
  System.out.println(rs.getString("code")
    + " " + rs.getString("name"));
}

connection.close();
```

Reyndar þarf að pakka þessu inn í try/catch.

Java

Það getur sparað tíma að nota PreparedStatement, sérstaklega fyrir INSERT skipanir

```
PreparedStatement insertStmt =
  connection.prepareStatement("INSERT INTO R(key,value) VALUES (?,?)");

insertStmt.setInt(1,3);
insertStmt.setInt(2,4);
insertStmt.executeUpdate();
```

Java

ResultSet í Java virkar eins og cursor í python.

- next() færir áfram á næstu niðurstöðu (fyrstu í fyrsta skipti) og skilar false ef ekkert meira er til
- Til að ná í gildi er hægt að nota
  - getString(i) fyrir strengi
  - getInt(i) fyrir heiltölur
  - getFloat(i) fyrir fleytitölur
- Dálkarnir byrja að telja í 1 (sic!)
- Í stað `int i` er hægt að vísa beint í nafnið á dálkinum með streng.