Mera DB

Transaktioner

ACID-(Atomic, Consistent, Isolation, Durability)
Hur hanteras transaktioner ?
Lost update
Dirty read
Låsning kan vara en lösning.
Vad är problemet ?
deadlock
långsamt

Alternativ till låsning

Optimistik approach
TimeStamp

opptimistisk approach

En transaktion fortsätter utan restriktioner tills closeTransaction (inget väntande)

Då kollar vi om det finns konflikt mot annan transaktion.

Om konflikt → en transaktion abortas.

TimeStamp

Varje operation valideras när den utförs. Om det inte går abortas transaktionen.

Varje transaktion får en tidsstämpel

Varje objekt har en read och write stämpel med sensaste read/write.

Vid read – ok om obj write stämpel är lägre än transaktionens.

Vid write – ok om obj read och write stämpel är lägre än transaktionens.

Distribuerade databaser

Är när man sprider ut datat på flera databaser

Ser oftast ut som en utåt.

D.v.s. det är DBMSen som sköter distribueringen transparant.

Distribuering av data är transparant

Transaktioners distribuering är transparent

Distribuerade databaser

Fördelar
- Pålitlighet
- Säkerhet
- Performance (kanske)
- Förmåga att kunna växa

Nackdelar
- Komplexitet
- Säkerhet (kanske)
- Kostnad (kanske)

Replikering

Vid ändring i en databas skickas ändringen till

alla databaser

Duplicering

En databas är master och den där alla ändringar görs och de distribueras sedan ut.

Distribuera data

Antingen har man allt på alla servrar

- Replikerings svårigheter

Alternativt så delar man upp data och har olika delar på olika servrar

- Krävs ingen replikering, dock inte samma säkerhet
- Sämre prestanda
- Mer komplicerade transaktioner.

Distribuerade transaktioner

Är en transaktion som accessat objekt på flera servrar.

När den avslutas committar alla servrar eller alla servrar abortar.

En server aggerar Coordinator

Two phase commit

Fas 1

Varje deltagare röstar. Om alla röstar för commit förbereds det och den kan inte ändra. Om den krashar måste tillståndet ändå sparas.

Fas 2

Deltagarna utför det gemensamma beslutet

OODB

Relationer vs objekt, ingen 100% koppling

, görs med kopplings tabell, men ingen skillnad mellan entitet och kopplingstabell i modellen.

Även 1,* kan vara av lite olika typ (association aggregat) vilket inte syns i datamodellen

Relation-OO mappning

Hur mappar man relationsdatabas till OO ?

Enitit -> Klass

1-1 vanliga ref attribut

1-* ref attribute i många entiteten

I klassen en collection i 1 klassen.

Obs! På andra sidan i OO!!!!

- Kopplings tabellen görs till collections i de båda klasserna.

OO

Hur mappas arv ?

Tre sätt

A(attr a1,a2) <- B(attr b1)

T_A (a1, a2) T_B (fk_a, b1)

T_A(a1, a2) T_B (a1, a2, b1)

En tabell innehåller all data från alla subklasser

OO

Frågan är dock blir modellen verkligen object orienterad ?

Eller är det bara en objektifiering av eniteter där arv inte finns och där relationenerna mest finns i databasen och inte i objektmodellen.

Finns ramverk som hibernate och JPA där sådant int ebara är möjligt utan relativt enkelt.

OODBM

Är databaser som inte är relations baserade utan objectbaserade.

Du sparar och hämtar upp object.

Passar väldigt bra ihop med objectorienterade program

Hut är det med snabbhet ???

db4o

Är en OpenSource OO databas med Java stöd (och .NET)

http://www.db4o.com/

DBn öppnas med

ObjectContainer db = DB4oEmbedded.
openFile(Db4oEmbedded.newConfiguration(),
<file>);

Spara object

```
Person p1 = new Person("Reine", 42);
db.store(p1);
Person p2 = new Person(Anders, 43);
db.store(p2);
```

Hämta object

```
List<Person> persons = db.query(Person.class);

Person p3 = new Person("Reine", 0);
ObjectSet res = db.queryByExample(p3);
while(res.hasNext())
    ....
List<Person> persons = db.query(new Predicate<Person>()
{ public boolean match(Person p) { return p.getShoeSize()
== 42;
});
```

update

```
Person p3 = new Person("Anders", 0);

ObjectSet res = db.queryByExample(p3);
Person p4 = (Person) res.next();
p4.setShowSize(44);
db.store(p4);
```

delete

```
ObjectSet result = db
        .queryByExample(new Person("Anders", 0));
Pilot p42 = (Person) result.next();
db.delete(p42);
```