**Grundlæggende Database**

**Kursist: Jens Erik Pedersen**

**Dato: 08-10-2017**

**Afleveringsopgave 3**

# 1. Hvilke egenskaber har en transaktion

En transaktion består af et antal operationer udført af en bruger eller et program, som læser eller manipulerer data i en database. Operationerne er enkelte SQL statements af typen DML (Data Manipulation Language), og kan være alle fire CRUD operationer fordelt på en eller flere tabeller. Antallet og typen af operationer afhænger af den enkelte transaktion og dens formål.

Fælles for alle transaktioner er, at de skal overholde **ACID** egenskaberne:

1) Atomar / **A**tomicity

En transaktion er atomar i den forstand, at enten gennemføres alle operationer i transaktionen eller også gennemføres ingen operationer. DBMS’s genopretnings subsystem sørger for, at denne egenskab overholdes ved enten at en transaktion bliver *committet* eller helt annulleret via en *rollback*.

2) Konsistens / **C**onsistency

En transaktion transformerer altid databasen fra en konsistent tilstand til en ny konsistent tilstand. Ansvaret for konsistens hviler på både DBMS og programmør. DBMS skal sikre, at alle integritets *constraints\*)* er overholdt efter databasen schema, og programmøren skal sørge for, at data operationerne efterlader databasen data i en konsistens tilstand.

3) **I**solation

Transaktioner udføres uafhængigt af hinanden, og hver enkelt transaktion kender intet til de øvrige transaktioners forløb og de tilstande som data er i på et givent tidspunkt. DBMS’en samtidigheds kontrol subsystem sikrer denne egenskab, når flere samtidige transaktioner udføres parallelt.

4) Durability / Holdbarhed

En succesfuld gennemført transaktion medfører altid, at effekten af operationerne bliver persisteret, dvs. gemt i database filen. Skulle DBMS’en efterfølgende gå ned eller databasen blive ødelagt, kan effekten af transaktionen genetableres.

# 2. Hvordan fungerer two-phase locking (2PL)

Når flere samtidige transaktioner udføres på en database, skal DBMS sikre at de enkelte transaktioner ikke kommer i konflikt med hinanden, hvis de manipulerer de samme data items.

Et eksempel på en metode er anvendelsen af låse (*locks*) som forhindrer, at mere end én transaktion kan manipulere et givent data item af gangen. Der findes to typer låse: læse-låse (*shared locks*) og skrive-låse (*eksklusive locks*), som DBMS’en bruger afhængig af den operation, som udføres på et givent tidspunkt i løbet af en transaktion. Hvis et data item kun skal læses, anvendes en *shared-lock* som andre transaktioner kan dele. Derved låses data item ikke for andre læse operationer, men kun for skrive operationer. Hvis et data item derimod skal manipuleres (opdateres eller slettes), anvendes en *exclusive lock* som forhindrer alle andre transaktioner i at få en lås på samme data item. Når en operation er gennemført, frigiver DBMS data item igen ved at fjerne låsen. Herefter kan andre transaktioner anmode om at få låsen til det pågældende data item.

Two-phase locking er en låse-teknik, hvor alle operationer i en transaktion som kræver en lås, får en tildelt lås, før afgivelse af låsene igen påbegyndes. En transaktion inddeles i to faser: *growing phase* og *shrinking phase*.

I *growing phase* tildeler DBMS låse til de data items, som operationerne foretages på. Det sker inden operationen udføres. Nøgler tildeles ikke nødvendigvis på én gang, men bliver tildelt efterhånden som operationerne kræver det. I denne fase frigives ingen tildelte låse, før alle nødvendige låse er tildelt.

I *shrinking phase* afgiver DBMS alle låsene igen, når sidste operation i transaktionen som gør brug af en lås, er færdig. I denne fase tildeler DBMS ikke længere nye låse.

# 3. Hvordan fungerer optimistisk concurrency control?

Optimistisk *concurrency control* kan anvendes som alternativ til *Locking* og *Timestamping* forhåndtering af samtidigheds konflikter ved transaktioner. Denne metodik har sin styrke, hvis sandsynligheden for konflikter er lille, og har ikke samme overhead som *Locking* og *Timestamping* ved henholdsvis håndtering af låse og ventetider forbundet hermed, og timestamp processering. Det betyder at performance generelt er bedre for optimistisk concurrency control, men hvis der opstår konflikter, skal hele transaktionen til gengæld rulles tilbage og startes forfra, hvilket er en tung og tidskrævende proces.

Optimistisk concurrency control består typisk af tre fase , og bruger timestamps til at tjekke eventuelle konflikter mellem de samtidige transaktioner. I hver transaktion noteres et timestamp for start af hver fase, undtagen i fase tre hvor timestamp noteres efter et commit.

De tre faser er:

1) *Read phase*

*Read phase* forløber fra start til umiddelbart før transaktionen *committes*. Værdier læses ud af alle relevante data items for transaktionen, og lagres i lokale variable i memory. Alle DML operationer udføres på de lokale variable, og selve data-filen røres således ikke.

2) *Validation phase*

Data lagret i lokale variable tjekkes i mod data i databasen. Der tjekkes for om de forestående operationer på databasen kan foretages, uden af *serializability\*\*)* princippet brydes.

Ved transaktioner, som kun består af læse-operationer tjekkes, om data items i databasen har ændret sig siden data blev udlæst. Hvis ikke det er tilfældet, committes transaktionen.

Hvis transaktionen indeholder opdaterings operationer, tjekkes om transaktionen vil efterlade databasen i en konsistent tilstand hvor Serializability honoreres.

En af følgende må være sand, før en transaktion består denne fase:

1. Alle transaktioner som er startet tidligere end indeværende, skal være afsluttet inden indeværende påbegyndes.
2. Hvis indeværende transaktion påbegyndes inden en tidligere transaktion er afsluttet, skal en af følgende betingelser være opfyldt:  
   - data items som opdateres af den tidligere transaktion må ikke være samme data items som indeværende transaktion arbejder på  
   - den tidligere transaktion afslutter sin w*rite phase* inden den indeværende transaktion påbegynder sin *validation phase*.

3) *Write phase*

Hvis valideringsfasen gennemføres med succes, committes transaktionen til den fysiske database fil i denne fase.

\*) *Integritets constraints* inkluderer *entitets integritet*, primær nøgler er unikke og kan identificere records. *Semantisk integritet* som sikrer at de manipulerede data værdier er i overensstemmelse med databasens schema. *Referentiel integritet*, som angiver om der til en fremmednøgle i en record i én tabel findes en record i en anden tabel hvor den primære nøgle har samme værdi.

\*\*) *Serializability* betyder, at flere samtidige transaktioner udføres således, at deres operationer ikke er i konflikt med hinanden. Databasen tilstand vil derfor efter alle transaktioner er udført være i samme tilstand, som hvis transaktioner var udført serielt.