# Database tips

## Denne artikel vil give nogle forskellige små praktiske råd om ting man skal tænke på når man arbejder med databaser og applikationer som bruger en database.

## Den forudsætter lidt kendskab til SQL men er for mindre erfarne.

Skrevet den **14. Feb 2010** af **arne_v** I kategorien **Databaser / Generelt** | ★★☆☆☆

Historie:
V1.0 - 04/07/2006 - original
V1.1 - 10/07/2006 - ret grim stave fejl i overskrift og uddybe den med en tabel
V1.2 - 12/02/2010 - smårettelser

### Formål

De følgende afsnit vil give nogle praktiske råd om hvad der er skidt og hvad der er godt når man arbejder med database, samt forklare lidt om hvorfor.

Tilsammen skulle de gerne gøre livet lidt nemmere for udvikleren.

Bemærk at det meste af det er generelle tommelfinger regler som gælder for alle databaser. En erfaren database mand vil ofte kunne finde lidt bedre løsninger for en specifik database.

### En type kolonne fremfor flere tabeller

Det er så godt som altid en fordel at samle sine data i en enkelt tabel fremfor at splitte dem op i flere tabeller med identisk tabel struktur.

```
tabelx_2005
----------
id INTEGER PK
val INTEGER

tabelx_2006
----------
id INTEGER PK
val INTEGER
```

laves bedre som:

```
tabelx
------
id INTEGER PK
aar INTEGER
```

val INTEGER

lande_europa
---
id INTEGER PK
navn VARCHAR(100)
hovedstad VARCHAR(100)

lande_amerika
---
id INTEGER PK
navn VARCHAR(100)
hovedstad VARCHAR(100)

laves bedre som:

lande
---
id INTEGER PK
navn VARCHAR(100)
hovedstad VARCHAR(100)
kontinent INTEGER

Det er nemmere at lave queries mod den samlede tabel og normalt vil performance også være bedre.

Bemærk at det ikke har noget med (de)normalisering at gøre. Normalisering er flere tabeller med forskellig tabel struktur som flettes sammen ved siden af hinanden med join. Det her drejer sig om flere tabeller med samme tabel struktur som stables ovenpå hinanden med union.

Der findes et begreb ved navn partitionering, hvor man bevidst splitter sine data op for at få bedre performance. Læs noget om det når din database bliver større end 1 TB !

**God performance via indexes**

Korrekt brug af indexes er kritisk for performance.

Er man god til database så kan man spørge databasen om hvilke index den vil bruge og hvor meget de betyder.

Er man ikke så god til databaser, så kan man bruge følgende tommelfinger regel:
  index på alle felter som bruges i join betingelser eller
  almindelige where betingelser

SELECT * FROM t1,t2 WHERE t1.f1=t2.f1 AND t2.f2=X

betyder index på t1.f1, t2.f1 og t2.f2

SELECT * FROM t1 JOIN t2 ON t1.f1=t2.f1 WHERE t2.f2=X

betyder index på t1.f1, t2.f1 og t2.f2

Vær opmærksom på at:

WHERE t.f='xxx'
WHERE t.f LIKE 'xxx%'

formentligt kan bruge index, mens:

WHERE t.f LIKE '%xxx'
WHERE t.f LIKE '%xxx%'

ikke kan bruge index.

Hvorfor de sidste to ofte er betydeligt langsommere end de første to.

**Standard SQL for portabilitet**

Hvis det er muligt bør man bruge ren standard SQL. Det gør det nemmere at skifte til en anden database.

Idealet er at man kun ændrer en connection string/URL og så virker ens applikation med en anden database.

Det er sjældent muligt at overholde den strategi 100% i praksis, men hvis du kan overholde den 75%, så har du kun 1/4 arbejde den dag du skal skifte database.

Og vær meget forsigtig med at tro, at du aldrig vil skifte database. Det er svært at spå om fremtiden og lige pludselig kan der være gode grunde til at skifte database.

Et godt eksempel på et totalt unødvendig portabilitets problem er brug af "invalide" navne.

MySQL:

SELECT `et felt` FROM `en tabel`;

Access og SQLServer:

SELECT [et felt] FROM [en tabel];

Fordi hvis man vælger sine navne fornuftigt kan det laves i standard SQL:

SELECT etfelt FROM entabel;

Andre ting som man skal være opmærksom på er:
- funktioner til forskellige former for data konverteringe er forskellige i forskellige SQL dialekter
- stored procedures kan være gode for performance men er sjældent portable mellem databaser

En anden pointe i samme boldgade: undgå navne som er reserverede ord i andre databaser.

**Korrekt arbejds fordeling mellem SQL og applikations kode**

SQL er fremragende til at selektere, joine, gruppere og sortere data med.

Men SQL er ikke et programmering sprog som sådan.

Skal man lave nogle meget komplekse beregninger på eller konverteringer af data, så er det ofte det bedste at hente de rå data ud med SQL og lave beregningerne/konverteringerne i sin applikation.

Prøv og undgå meget komplekse streng manipulationer eller matematiske beregninger i SQL.

Stored procedures og funktioner i SQL er glimrende til at udføre multiple SQL sætninger uden roundtrips mellem applikation og database. Men de er stadigvæk ikke velegnede til generel programmering.

Visse databaser tillader brug af generelle programmerings sprog i stored procedures og funktioner (SQLserver - C#, VB.NET etc., Oracle, DB2, Sybase - Java). Så kan man naturligvis boltre sig med generel programmering.

**Korrekt data type**

Det er vigtigt for funktionaliteten af ens database at vælge den rigtige data type.

Det er åbenlyst at feltet skal have en data type så den kan indeholde de værdier man vil putte nd i feltet.

Men hvis det var eneste kriterie så kunne man jo vælge VARCHAR til næsten alt.

Men der er forskel på sortering. INTEGER bliver sorteret 3,22,111 mens VARCHAR bliver sorteret '111','22','3'.

Der er forskel på operationer. 11+22 er 33 mens '11'+'22' er '1122' (forudsat at SQL dialekten tillader plus på VARCHAR).

Her er nogle generelle tommelfinger regler:

id'er : stor INTEGER type
antal : INTEGER type - hellere for stor end for lille
beløb : NUMERIC(12,2) eller DECIMAL(12,2)
navne : VARCHAR(n) hvor n er en passende max. længde - hellere for stor end for lille
afstand eller vægt : FLOAT eller DOUBLE eller hvad de nu hedder i SQL dialekten
tid : DATETIME eller hvad de nu hedder i SQL dialekten
true/false : BOOLEAN eller hvad de nu hedder i SQL dialekten
billeder : BLOB eller hvad de nu hedder i SQL dialekten

Man må aldrig bruge FLOAT/DOUBLE til beløb p.g.a. regne nøjagtigheds problemer.

**Beskyt mod SQL injection**

Et kendt applikations/database problem er kendt som SQL injection.

Et lille eksempel.

vi skal lave et login check og har et brugernavn felt og et kodeords
felt som vi vil checke

så vi laver en SQL streng som:

sqlstr = "SELECT * FROM brugere WHERE brugernavn = '" + indtastetbrugernavn + "' AND kodeord = '" + indtastetkodeord + "'"

udfører den og checker på om den returnerer en række eller ej

det virker fint.

indtil en ondsindet person indtaster følgende i de to felter:

administrator
x' OR 'x' = 'x

fordi så ser SQL strengen ud som:

SELECT * FROM brugere WHERE brugernavn = 'administrator' AND kodeord = 'x' OR 'x' = 'x'

som altid vil returnere nogle rækker.

Man kan beskytte sig mod den slags på database niveau ved at bruge
parameters (VBS,C#,VB.NET) eller prepared statement (Java, PHP 5).

Se evt. mine artikler:

http://www.eksperten.dk/guide/830 Prepared statements i Java
http://www.eksperten.dk/guide/831 Parameters i C#
http://www.eksperten.dk/guide/832 Parameters i VB.NET

Eller man kan beskytte sig på applikations niveau enten ved skrap data
validering eller ved at escape input (konvertere ' til '' eller \').

Man skal altid være meget opmærksom på risikoen for SQL injection.

**Filer i eller udenfor databasen**

Dette er et meget omdiskuteret emne.

Det har været god latin blandt mange udviklere i mange mange
år at det er skidt at gemme filer (typisk billeder) i databasen, men
at man istedetfor skal gemme filen eksternt i filsystemet og kun gemme
stien til filen i databasen.

Og det var da også rigtigt at f.eks. en Windows 3.11 med Access 2.0
på en 486 PC totalt døde når man forsøgte at putte billeder i en
database.

Men computere idag er rigtigt mange gange kraftigere end dengang. Og database

software har også udviklet sig betragteligt. Jeg vil derfor kraftigt anbefale at man ikke bare afviser filer i database ideen men undersøger spørgsmålet grundigt.

Der er store administrative, sikkerheds og backup mæssige fordele ved at have filerne i databasen.

Spørgsmålet er udelukkende om filer i database vil give god nok performance.

Jeg har testet en del og læst en del. Mit postulat er:
- på en lille database server vil det ekstra overhead ved at have filerne i databasen være minimalt
- på en stor database server vil performance faktisk kunne være bedre ved at have filerne i databasen

(for filer i en normal størrelse som billeder til web sider - CD/DVD images er noget helt andet)

Læserne skal ikke tage min postulater som den endegyldige sandhed, men jeg vil anbefale læserne at lave deres egne performance målinger.

**Programmering**

For noget mere programmerings teknisk se:
http://www.eksperten.dk/guide/996

**Kommentar af foxmulder58 d. 10. Jul 2006 | 1**

Det er så godt som altid en fordel at samle sine data i en enkelt tabel fremfor at splitte dem op.  = redudant indtastning af data og kan ikke anbefales.

**Kommentar af trer d. 05. Jul 2006 | 2**

Fin artikel og gode basale anbefalinger. Læseværdig.

**Kommentar af miqe d. 05. Jul 2006 | 3**

Ganske informativ.
Gode hints til hvordan man, specielt som begynder, undgår de værste problemer.

**Kommentar af jim_marius d. 13. Jul 2006 | 4**

God og læsevenlig artikel

**Kommentar af lassemelbye d. 17. Dec 2006 | 5**

God og letforståelig

**Kommentar af phil-profil d. 28. Aug 2006 | 6**

**Kommentar af crc1 d. 20. Sep 2006 | 7**