Norges teknisk-naturvitenskapelige universitet - NTNU
Fakultet for naturvitenskap og teknologi
Institutt for fysikk

HOVEDOPPGAVE FOR

STUD. TECHN. LENE MYKLEBUST OG MARIT LIMSTRAND

Oppgaven gitt: 21.01.02
Besvarelsen levert: 14.06.02

FAGOMRÅDE: DATABASE

Norsk tittel: *"Temporal XML database "*
Engelsk tittel: *"Temporal XML database"*

Hovedoppgaven er utført ved institutt for datateknikk og informasjonsvitenskap, under veiledning av Kjetil Nørvåg

---

Trondheim, 14.06.02,
Kjetil Nørvåg
Ansvarlig faglærer
Førsteamanuensis ved Institutt for fysikk

Forord

Denne rapporten er skrevet av to sivilingeniør-studenter Lene Myklebust og Marit Limstrand som studerer datateknikk på femteåret ved IME-fakultetet på NTNU. Oppgaven ble utført våren 2002.

Målet med prosjektet var å sette oss inn i prinsippene bak temporale XML-databasesystemer, samt implementert en temporal utvidelse over et av databasesystemene.

Vi vil takke vår veileder Kjetil Nørvåg og databaseadministratoren Jon Olav Hauglid for veiledning.

Trondheim, 14.06.02

Marit Limstrand   Lene Myklebust

________________   ________________

Sammendrag

Denne rapporten beskriver arbeidet vi har gjort rundt temporale XML-databasesystemer. Gjennom prosjekttiden har vi gjort bakgrunnsundersøkelser, sammenligning av ulike databasesystemer og implemetering av et overbygg til et databasesystem. I oppgaven har vi fokusert på lagring av XML-dokumenter og testing for å finne den beste måten å modellere de temporale tabellene på.

Vi har også sett på en del andre databasesystemer med XML-støtte som er på markedet i dag. Ingen av disse har noe særlig støtte for temporale aspekter. Det er gjort noe forskning innen temporale XML-databasesystemer, men det er ingen databasesystemer som per i dag har en god løsning på denne problematikken.

Når man skal legge temporale aspekter til et DBMS, i vårt tilfelle Oracle9i, må man lagre tidsstempler for hver dokumentversjon. Vi har sett på tre ulike måter man kan modellere dette på. Vi kjørte en del tester for å finne den beste lagringsmetoden, og kom fram til at det var ved å lagre gjeldende og historiske versjoner i to forskjellige tabeller. Tidsstemplene vil tilsvarende bli lagt i to forskjellige tabeller.

Videre har vi implementere et overbygg som skal legge temporale aspekter til et databasesystem. Her brukte vi som sagt databasesystemet Oracle9i, på grunn av at den allerede er installert ved universitetet. De temporale aspektene overbygget skulle stå for har vært å legge tidsstempler på XML-dokumenter, og legge til en utvidelse av spørrespråket. Dermed skal brukeren selv kunne lage temporale tabeller for versjonering av XML-dokumenter og spørre i disse tabellene.

Innholdsfortegnelse

1.0 Innledning ... 1
- 1.1 Motiverende eksempler ... 1
  - 1.1.1 Case 1: Ansatte i en bedrift ... 1
  - 1.1.2 Case 2: Reklame i en avis ... 2
  - 1.1.3 Case 3: Medlemsavgift ... 2

DEL I Databasesystemer

2.0 Databasesystemer ... 4
- 2.1 Første generasjons databasesystemer ... 4
- 2.2 Andre generasjons databasesystemer ... 5
- 2.3 Tredje generasjons databasesystemer ... 5

3.0 XML og databasesystemer ... 6
- 3.1 XML ... 6
  - 3.1.1 XPath ... 7
- 3.2 XML-dokumenter ... 8
  - 3.2.1 Dataorienterte dokumenter ... 8
  - 3.2.2 Lagring av dataorienterte dokumenter ... 8
  - 3.2.3 Menneskelesbare dokumenter ... 9
  - 3.2.4 Lagring av menneskelesbare dokumenter ... 9
  - 3.2.5 Hybrid av menneskelesbare og dataorienterte dokumenter ... 10
  - 3.2.6 Lagring av hybriden ... 10
- 3.3 Lagring av XML-dokumenter ... 11
  - 3.3.1 XML-enabled databasesystemer ... 11
  - 3.3.2 Native XML-databasesystemer ... 13
  - 3.3.3 Forskjellen mellom XML-enabled og native XML-databasesystemer ... 14

4.0 Temporaldatabasesystemer ... 16
- 4.1 Tidsaspekter: gyldighetstid og transaksjonstid ... 16
- 4.2 Versjonering ... 17
  - 4.2.1 Extension versioning ... 17
  - 4.2.2 Schema versioning ... 17
- 4.3 Implementere tid i relasjonsdatabasesystemer ... 17
  - 4.3.1 Gyldighetstidsrelasjoner ... 18
  - 4.3.2 Transaksjonstidsrelasjoner ... 18
  - 4.3.3 Bitemporale relasjoner ... 19
  - 4.3.4 Mulige måter å implementere tid i relasjonsdatabasesystemer ... 19
- 4.4 Implementere tid i objektorienterte databasesystemer ... 19

5.0 Temporale XML-databasesystemer ... 21
- 5.1 Tradisjonelle dokument-versjoneringssystemer ... 21
- 5.2 Usefulness-Based Copy Control ... 22
- 5.3 SPaR versjons skjema ... 24

6.0 Spørrespråk ... 25
- 6.1 Spørrespråk for tradisjonelle databasesystemer ... 25
  - 6.1.1 SQL3 ... 25
- 6.2 XML-spørrespråk ... 26
  - 6.2.1 XQuery ... 28
  - 6.2.2 SQL/XML ... 30
- 6.3 Temporale spørrespråk ... 30
  - 6.3.1 TSQL2 ... 31

DEL II Programmer/Produkter

7.0 Native XML-databasesystem produkter ... 35
7.1 Tamino XML Server ... 35
7.1.1 Arkitektur ... 36
7.2 Extensible Information Server ... 37
7.2.1 Arkitektur ... 38
7.3 X-hive/DB ... 38
7.3.1 Arkitektur ... 39
8.0 XML-enabled databasesystemer ... 40
8.1 IBM: DB2 Universal Database V7.2 ... 40
8.1.1 XML-støtte ... 40
8.2 Microsoft SQL Server 2000 ... 41
8.2.1 XML-støtte ... 41
9.0 Oracle9i ... 42
9.1 Utviklingen til Oracles XML-teknologi ... 42
9.2 Lagring av XML-dokumenter ... 43
9.2.1 Datatypen XMLType ... 44
9.2.2 Dekomponert XML ... 45
9.3 Indeksering ... 45
9.3.1 Functional Index med XMLType ... 45
9.3.2 Text Index på XMLType-kolonner ... 46
9.4 Søking i XML-data ... 46
9.5 Datatyper for tidsstempel ... 47
9.6 Aritmetikk ... 47
9.7 XML SQL Utility (XSU) ... 47
9.7.1 SQL-til-XML mapping ... 48
9.7.2 XML-til-SQL mapping ... 49
9.8 Oracle9i database Release 2 ... 50
9.8.1 XMLType ... 50
9.8.2 Repository ... 50
9.8.3 Indeksering ... 50

DEL III Vår løsning

10.0 Funksjonalitet ... 53
10.1 Grensesnittet ... 53
10.2 API ... 55
10.3 Spørrespråket ... 55
11.0 Design ... 59
11.1 Arkitektur ... 59
11.2 Lagring ... 60
11.2.1 Lagring av XML-dokumentene ... 60
11.2.2 Legge til tid på elementer ... 60
11.2.3 Hvordan skille versjonene ... 60
11.3 Evaluering av uttrykkene ... 61
11.3.1 Algoritmer til version ... 62
11.4 Transaksjonshåndtering ... 69
11.5 Kommunikasjon ... 69
11.6 Ytelsesvurderinger ... 71
11.7 Edit script ... 72
12.0 Lagringsalternativer ... 73

12.1 Løsningsforslag 73
12.1.1 TX1: En tabell med gjeldende og historiske 74
12.1.2 TX2: To tabeller; en for gjeldende og en for historiske 74
12.1.3 TX3: To tabeller; en for gjeldende og en for historiske og gjeldende 75
12.1.4 Sammenligning 75
12.2 Kostnadsmodell 76
12.2.1 Antakelser 76
12.2.2 Innflytelser på resultatet 76
12.2.3 Kostnadsmodellen 76
12.2.4 Sammenligning av løsningene 78
12.3 Beskrivelse av testmateriale 78
12.4 Gjennomføring av testene 78
12.4.1 Tidsbruk for innlegging i oracle database 79
12.5 Resultater 79
12.6 Konklusjon 81

13.0 Implementasjon 83
13.1 Verktøy og databasesystem 83
13.2 SQL parser 83
13.3 Klasser 84
13.3.1 Interface 84
13.3.2 Parser 85
13.3.3 Message 85
13.3.4 Help 86
13.3.5 ParseStatement 86

14.0 Evaluering av programmet 87
14.1 Styrker 87
14.1.1 Grensesnitt 87
14.1.2 Spørrespråket 87
14.2 Svakheter 88
14.3 Utvidelser av programmet 88
14.3.1 Utvidelser i forhold til transaksjoner 88
14.3.2 Utvidelser av tidsaspektet 89
14.3.3 Utvidelser av spørrespråket 89

DEL IV Avslutning

15.0 Konklusjon 91

16.0 Referanser 92

DEL V Appendiks

A Ordliste .... 3 sider

B Tabeller over produkter vi har sett på .... 3 sider

C Native XML Database produkter .... 3 sider

D Testing .... 5 sider

E Test av XML .... 4 sider

F Kode .... 59 sider

Figurliste

FIGUR 1. Versjon 1....22
FIGUR 2. Versjon 2....23
FIGUR 3. Versjon 2....23
FIGUR 4. Viser predikater ved ulike temporale par av perioder [2]....32
FIGUR 5. Taminos arkitektur[48]....36
FIGUR 6. Arkitektur til eXtensible Information Server[49]....38
FIGUR 7. Utviklingen til Oracles XML-teknologi (Oracle8i, Oracle9i Release 1 og Oracle9i Release 2)[55]....42
FIGUR 8. Viser forskjell på bruk av XMLType og ekstrahering av XML-dataene til flere kolonner....43
FIGUR 9. TX grensesnittet....54
FIGUR 10. Viser versjoner som blir returnert ved bruk av ulike version-parametre....56
FIGUR 11. Arkitektur over TXSQL....59
FIGUR 12. Oversikt over tabellen som lages av TX når brukeren lager Usertable.62
FIGUR 13. Viser hvilke versjoner som returneres ved bruk av version(all) og version(history)....69
FIGUR 14. Flyt skjema for preprosessor....70
FIGUR 15. Oversikt over antall tabeller og hva de lagrer for alternativene. H - historiske versjoner og C - gjeldende versjoner....73
FIGUR 16. Tabeller som lages for TX1....74
FIGUR 17. Tabellene som lages for TX2 og TX3....75
FIGUR 18. Viser tidsbruken for å legge inn dokumenter i Oracle-databasen....79
FIGUR 19. Sammenligning av kostnader for de ulike uttrykk....80
FIGUR 20. Totalkostnader ved ulik vekting av uttrykkene....80
FIGUR 21. Klassediagram over vår preprosessor....84

Tabell-liste

TABELL 1. Ulikheter mellom XML-data og relasjons databaser............................12
TABELL 2. Relasjonen Konto_GT...18
TABELL 3. Funksjonalitet som et XML-spørrespråk bør ha..................................27
TABELL 4. Kostnadsfunksjoner til TX1, TX2 og TX3 for insert, delete, update og select...77
TABELL 5. Oversikt over faktorer for de ulike uttrykkene i prosent......................78
TABELL 6. Oversikt over uttrykk som testes...112
TABELL 7. Resultatet etter spørringene...113
TABELL 8. Antall dokumenter som inneholder bestemt element.........................113

1.0 Innledning

XML standarden begynner å bli mer og mer brukt ved datatransportering. Etter hvert som mengden av XML-dokumenter øker, øker også behovet for å få gode lagringsalternativer til disse XML-dokumentene. Dette har gjort at det har blitt en utvikling mot å lagre XML-dokumentene i databasesystemer. De tradisjonelle databasesystemene, som relasjonsdatabasesystemer, har begynt å legge til egne XML-støtter. Disse har ofte en del begrensninger for enkelte typer av XML-dokumenter. I tillegg til disse databasesystemene, har det begynt å komme egne XML databasesystemer. Disse er laget for å lagre XML-dokumenter.

Med tanke på at XML-dokumenter blant annet blir brukt som datatransportering, vil det også komme mange versjoner av disse dokumentene. Her kommer vår oppgave inn. Vi skal se på temporale XML-databaser, som versjonerer disse XML-dokumentene. Det er fram til i dag forsket på dette, men ikke laget et databasesystem som faktisk implementerer dette på en god måte.

Vår oppgave går ut på å lage en temporal utvidelse av et databasesystem med XML-støtte. Denne utvidelsen skal kunne versjonere XML-dokumenter. For å gjøre dette har vi valgt databasesystemet Oracle9i. Hovedgrunnen er at det er installert på skolen, og har en støtte for XML-dokumenter. For å legge til tidsstempel på XML-dokumentene har vi valgt å bruke tidsdimensjonen transaksjonstid. Dette vil være tiden når XML-dokumentet blir lagt i databasen. Vi vil komme med noen motivernde eksempler, som viser når ulike tidsdimesjoner vil være mest relevante.

Vi vil forøvring gjøre oppmerksom på at vi har lagt til en ordbok i appendiks A.

1.1 Motiverende eksempler

For at leseren skal bli motivert til å lese videre og få en forståelse av hvor man kan bruke temporale XML-databasesystemer har vi tatt med noen eksempler. Vi vil gi en kort beskrivelse av eksemplene.

1.1.1 Case 1: Ansatte i en bedrift

Det første caset tar for seg frima/ansatt-forholdet. Et firma registrerer ansatte og informasjon om dem i XML-dokumenter. Etterhvert som de forandrer på informasjonen om de ansatte, vil XML-dokumentene oppdateres.

```
<firma>data as
   <ansatt>
      <navn>Ole Karlsen</navn>
      <stilling>sekretaer</stilling>
      <lonn>220000</lonn>
      <mail-adresse>Ole.Karlsen@data.no</mail-adresse>
      <telefon>23232323</telefon>
   </ansatt>
</firma>
```

Om Ole Karlsen får en ny stilling vil en ny versjon av dokumentet bli laget. Lønnen vil også mest sannsynlig gå opp. Ved en senere anledning kan det være at en annen person blir ansatt i sekretær-stillingen som Ole Karlsen hadde. Det vil da for eksempel være

ønskelig å vite hvor mye Ole Karlsen fikk i lønn da han hadde stillingen. Ved bruk av et temporalt XML-databasesystem kan man sende en spørring til databasesystemet der man spør etter hvor mye Ole Karlsen hadde i lønn når han var ansatt som sekretær.

1.1.2 Case 2: Reklame i en avis

En avis e.l. med reklame, vil kunne ha stor nytte av å benytte en temporal database. Når noen vil ha en reklame i avisen kan denne legges i databasen sammen med tidspunkter for når reklamen skal være med i avisen. Om man for eksempel legger inn en reklame i databasen den 7. august, kan man si at den skal være gyldig fra 10. august til og med den 13. august. Gjør man en spørring om hvilke reklamer som er gyldige den 9. august, vil ikke denne reklamen være med i svaret som returneres fra databasen. Derimot vil reklamen være med i resultatet for datoene 10., 11., 12. og 13. august.

1.1.3 Case 3: Medlemsavgift

I noen tilfeller er det interessant å finne ut hvilke XML-dokumenter som er lagt til databasen etter et visst tidspunkt. En organisasjon kan for eksempel sende ut betalingsgiro for medlemsavgift en gang i året. Denne giroen kan for eksempel bli sendt ut i januar. Oraganisasjonen vil også at medlemmer som har meldt seg inn i organisasjonen etter januar skal betale årsavgiften. Med en temporal database lar dette seg gjøre ved å sende en spørring til databasen om hvilke dokumenter som har blitt lagt til etter januar.

DEL I

Databasesystemer

Vi starter her med å beskrive ulike databasesystmer som er relevante for vår oppgave. Som en innledning gir vi en liten oversikt over de viktigste databasesystemer som er laget. Deretter kommer vi mer inn på selve oppgaven vår med å beskrive hvordan XML kan lagres i databasesystemer. I kap 4 beskrives det hvordan tid kan implementeres i ulike databasesystemer. Videre kommer vi inn på hvordan man kan løse utfordringen med å versjonere XML-dokumenter, altså legge til temporale aspekter på XML-dokumenter. Til slutt vil vi gi en kort beskrivelse av spørrespråkene til de viktigst databasesystemene vi har gått gjennom.

Kap 2: Databasesystemer

Kap 3: XML og databasesystemer

Kap 4: Temporaldatabasesystemer

Kap 5: Temporale XML-databasesystemer

Kap 6: Spørrespråk

2.0 Databasesystemer

Opp gjennom tidene er det blitt utviklet ulike typer databasesystemer. Disse kan deles inn på følgende måte[7,8]:

- **Først generasjon:** Hierarkiske og nettverks databasesystemer
- **Andre generasjon:** Relasjons databasesystemer
- **Tredje generasjon:** Objektorienterte og objektrelasjons databasesystemer

I tillegg til disse er det også blitt utviklet ulike spesialiserte databasesystemer. Av de viktigst kan det nevnes knowlegde-based, temporal, spatial, document og multimedia databasesystemer. Vi kommer ikke til å gå inn på disse spesialiserte databasesystemene, med unntak av temporaldatabasesystemer som vi beskriver i kapittel 4.0.

I dag er det relasjons databasesystemer som dominerer. Hierarkiske og objekt databasesystemer er ikke like generelle som relasjons databasesystemene, men utfyller fordypende krav.

For dagens databasesystemer er det en del egenskaper som er veldig viktige. De viktigste kan listes opp som følger[9]:

- ACID egenskapene. Atomiske, konsistente, isolerende og varige data. Disse finner man i både relasjons og objekt databasesystemer
- Samtidighetskontroll og låse mekanismer er også viktig når man har systemer med mange brukere som skal dele den samme informasjonen
- Recovery er viktig for å bevare konsistente data etter krasj eller andre ting som kan gå galt

Vi skal nå se nærmere på de tre generasjonene av databasesystemer som vi beskrev over.

2.1 Første generasjons databasesystemer

Første generasjons databasesystemer startet med hierarkiske databasesystemer på 1960-tallet. Denne modellen ble laget for å lagre data og relasjonen mellom dem. Dataene er representert som en hierarkisk trestruktur. Nodene og løvet i treet representerer filene, og de kan bare ha en foreldrenode. Dataene blir alltid aksessert sekvensielt. Modellen har ikke noe standard spørrespråk, og det er i dag en lite brukt modell.

Etter hvert ble trestrukturen byttet ut med en grafstruktur i nettverksdatamodellen. Det kom da ulike restriksjoner på retningen til relasjonene. Dermed kunne man få en-til-en og en-til-mange relasjoner. Det var verre når man vil lage mange-til-mange relasjoner.

Begge disse modellene er avhengige av den interne strukturen når man skal representere dataene for brukerne. Det trengs kunnskap om hvordan de ulike dataene er lenket sammen med hverandre.

2.2 Andre generasjons databasesystemer

På 1970-tallet kom relasjons databasesystemene, som er mindre avhengig av den interne strukturen for å vise dataene for brukerne. Basis relasjonsmodellen representerer et databasesystem som en samling av tabeller, hvor hver tabell kan lagres som en separat fil. Hver tabell inneholder et fastsatt antall kolonner og et uendelig antall med rader. En eller flere kolonner i hver tabell utgjør primærnøklen. Denne brukes til å identifisere radene i relasjonen. Tabeller har typisk også sekundærnøkler. Sekundærnøklene korresponderer til primærnøkler i andre tabeller.

Kunde { Personnr, Navn, Kundeinfo }

Konto {Kontonr, Personnr, Saldo }

I dette eksemplet er kolonnen Personnr primærnøklen til tabellen Kunde, mens den er sekundærnøklen til tabellen Konto.

De fleste relasjons databasesystemene bruker høynivå spørrespråket SQL og støtter begrensende former for brukerutsnitt. I kapittel 6.1.1 er SQL beskrevet nærmere.

2.3 Tredje generasjons databasesystemer

Den største ulempen med relasjons databasesystemer er at den har en flat struktur. Dette gjør at det er flere situasjoner hvor man må representere komplekse relasjoner ved å dele dem opp i flere relasjoner. Dermed kan man miste den opprinnelige meningen med relasjonen og dataene. For å forbedre dette ble modellen utvidet med komplekse attributter og objekter. I tillegg kom det også ulike måter å gjøre spørringer på disse dataene. Denne nye modellen ble kalt objektrelasjons databasesystem[3] og kom på 1980-tallet.

Objektrelasjonsmodellen legger et objektlag over relasjonsstrukturen, slik at det kan brukes en syntaks og en semantikk som ligner på objektorienterte programmering. Dermed får den objektorienterte egenskaper som brukerdefinerte og komplekse datatyper og arv.

Objektorienterte databasesystemer er en utvidelse av den klassiske databasesystem modellen og består av objektorienterte teknikker. Objektmodellen definerer et databasesystem ved objekter, deres egenskaper og deres operasjoner. Objekter med samme struktur og oppførsel hører til en klasse, og klasser er organisert hierarkisk. Operasjonene til hver klasse er spesifisert ved forhåndsdefinerte prosedyrer kalt metoder. Objekter kan være heterogene og de inneholder samlinger av eide data i tillegg til data som er arvet.

Forskjellen mellom objektrelasjons databasesystem og objektorienterte databasesystemer er at ved objektrelasjon databasesystem vil dataene lagres i en relasjons struktur, altså i tabeller. Brukeren vil se dataene som et objekt, og ikke som tabeller. Ved objektorienterte databasesystemer vil dataene lagres som objekter, og dermed også vises som objekter for en bruker.

Spørrespråket Object Query Language (OQL) er laget spesielt for objektmodellen. Syntaksen er veldig lik som SQL syntaksen. OQL har også støtte for objekt identiteter, komplekse objekter, operasjoner, arv, polymorfi og relasjoner.

3.0 XML og databasesystemer

XML er i dag mye brukt til flere ulike formål, og begynner derfor å bli mer og mer aktuell i forhold til databasesystemer. I dette kapitlet vil vi derfor komme inn på koblingen mellom XML og databasesystemer. Vi starter med en beskrivelse av hva XML er og hva det brukes til. Deretter kommer vi inn på ulike typer av XML-dokumenter, som kan være aktuelle å lagre i et databasesystem. Til slutt vil vi komme inn på hvordan XML-dokumenter kan lagres i ulike typer databasesystemer.

3.1 XML

XML[39,44], Extensible Markup Language, er en standard for strukturert data på menneskelesbar tekstform. Alle typer av data kan representeres i XML og man definerer egne tagger etter behov. Som HTML[41], bruker XML også start- og slutt-tagger. I HTML fins det omkring 100 forskjellige tagger, mens XML ikke har noen fastsatte taggsett.

XML ble utviklet av W3C[1][40] og er en delmengde av SGML[2], Standard Generalized Markup Language. HTML er en applikasjon av SGML, med sitt fastsatte taggsett. XML selv er ingen applikasjon, men et metaspråk, i likhet med SGML. Det kan derfor lages applikasjoner av XML også. Den første anbefalingen til XML ble gitt ut i februar 1998. Den ble utviklet på grunn av at SGML er en for tung og vanskelig standard. XML har 20% av kompleksiteten, men 80% av kapasiteten til SGML.

Et XML-dokument er en logisk enhet av data som er merket i XML. De fleste dokumenter har en naturlig, hierarkisk oppbygging, som kan representeres som en trestruktur. Det kan bestå av elementer og attributter:

```
<?xml version="1.0"?>
<personer>
  <person id=10>
    <navn>Lars Olsen</navn>
    <adresse>
      <gate>Olsensvei 1</gate>
      <postnr>7050</postnr>
      <poststed>Trondheim</poststed>
    </adresse>
    <telefon>73975126</telefon>
    <epost>larso@hotmail.com</epost>
    <epost>olsen@ntnu.no</epost>
  </person>
</personer>
```

Eksemplet viser hvordan et XML-dokument skal settes opp. Det starter med en tagg, som forklarer at det brukes XML versjon 1.0 i resten av XML-dokumentet. Videre

1. The World Wide Web Consortium ble dannet i oktober 1994 og har til hensikt å lede World Wide Web ved å utvikle protokoller som hjelper fram dens utvikling og sikrer dens samspillevne. W3C har mer enn 500 medlemsorganisasjoner fra forskjellige steder rundt om i verden.
2. Standard Generalized Markup Language er en internasjonal standard for beskrivelse av oppmerket tekst. SGML er et formateringsspråk.

kommer et rot-element, som for dette dokumentet er personer. Neste element, som er person, består av et attributt som gir hver person en id. Videre er det listet opp ulike data for personen Lars Olsen. XML kan ha flere like tagger, slik det vises i eksemplet med to epost-adresser for Lars Olsen.

Grunnfilosofien til XML er at kodingen i et dokument bør beskrive dets struktur og innhold, men ikke hvordan det skal presenteres. Kodingen følger formelle regler, slik at dataprosesseringsteknikker som brukes på databasesystemer også kan brukes på dokumenter.

XML er en plattform- og leverandøruavhengig metode å transportere strukturert data på. Man kan flytte data mellom systemer gjennom HTTP uten å være bekymret for leverandører eller plattformer til sender og mottaker. Siden XML-dokumenter er enkel tekst kan de lett transporteres via HTTP, og kombinasjonen av HTTP og XML gjør det enkelt å transportere data i hvilken som helst struktur og kompleksitet. Dette gjør at XML brukes av flere databasesystemer til utveksling og lagring av data.

En DTD, Document Type Definition, definere en grammatikk for elementene i et XML-dokument. Den beskriver hvordan elementer og entiteter kan føres opp i et dokument og hva elementets innhold og attributter er. Om dokumentet stemmer med DTDen til dokumentet, er dokumentet gyldig. Et XML-dokument trenger ikke ha noen DTD. Det eneste kravet er at dokumentet må være velformulert, altså følge reglene til XMLs syntaks.

DTDer er ikke alltid tilstrekkelig for å dekke alle behov for beskrivelse av XML-dokumenter. Det er behov for mer effektiv kontroll av innholdet i XML-dokumenter noe W3C-standarden XML Schema [42] vil gjøre bedre. XML Schema har innflytelse fra tradisjonelle databasemiljøer. Et XML-skjema er et dokument som beskriver det gyldige formatet til et XML-datasett. Det forteller om hvilke elementer som er tillatt, hvilke attributter et element kan ha, antall forekomster av et element, osv. Et XML Schema er altså ganske lik en DTD, men med noen forbedringer. Hovedsakelig går forbedringene ut på at XML Schema støtter namespace[1] og er skrevet i XML syntaks.

3.1.1 XPath

XPath[4,5] brukes for å søke i og hente ut informasjon fra XML-dokumenter. Den bruker en syntaks som relaterer seg til den logiske strukturen til XML-dokumentet. For å illustrere dette kan vi hente ut navnet til personen i XML-eksemplet over som har telefonnr lik 73975126.

```
/person[telefon = 73975126]/navn
```

Man kan skrive XPath-uttrykk som referer til bestemte elementer, attributter, tekst-strenger, kommentarer, prosessinstruksjoner eller navneområder. XML-dokumenter er egentlig et tre av noder. Dermed kan XPath brukes til å indikere ulike noder i treet ved hjelp av posisjon, relativ posisjon, type, innhold ol.

1. Namespace, eller navnerom, er en samling av navn, identifisert av en URI-referanse, som blir brukt i XML-dokumenter som elementtyper og attributtnavn.

XPath brukes av flere ulike systemer for XML-dokumenter, særlig i databasesystemer som er spesielt tilrettelagt for XML-dokumenter. Videre brukes XPath mye i XSLT-dokumenter[43], som brukes til å transformere XML-dokumenter.

I løpet av dette året skal XQuery[31] komme ut som et standard spørrespråk for XML. Dermed vil mange gå over til å bruke denne standarden, som er mye mer avansert enn XPath. I XQuery vil det fortsatt være mulig å skrive helt vanlige XPath-uttrykk. XQuery vil bli beskrevet i kapittel 6.2.1.

3.2 XML-dokumenter

Det er vanlig å dele XML-dokumentene inn i to hovedtyper[21], nemlig dataorienterte og menneskelesbare dokumenter. Det er ikke alltid like lett å skille mellom disse to dokumenttypene, men vi skal i den følgende delen forsøke å belyse noen karakteristikker som beskriver dem og se på noen av deres bruksområder. Vi kommer også til å beskrive en hybrid løsning mellom disse dokumenttypene, som i mange situasjoner kan være nyttig.

3.2.1 Dataorienterte dokumenter

I dataorienterte dokumenter blir XML brukt som et format til å transportere data. For dokumenttypen er det maskinen som er i fokus, og den har derfor en klar regulær struktur. Videre er den karakterisert med fingranulerte data og lite eller ingen blanding av innholdet. Rekkefølgen på når søsken-data-elementer og PCDATA oppstår er ofte ikke avgjørende, foruten om ved validering av dokumenter. Eksempler på dataorienterte dokumenter kan være salgsordre, flyreservering, forskningsdata og aksjemarkedsdata.

```
<?xml version=”1.0”?>
<bøker>
  <bok>
    <tittel>XML for nybegynnere</tittel>
    <forfatter>Jimmy West</forfatter>
    <år>2001</år>
    <sidetall>654</sidetall>
    <pris>599</pris>
 </bok>
  <!-- Flere bøker -->
</bøker>
```

3.2.2 Lagring av dataorienterte dokumenter

Når man bruker dataorienterte dokumenter for å transportere data, benytter man seg ofte av mappemetoder for å legge dataene inn i ulike kolonner i databasens tabeller. Her vil det være ulike ting man må tenke på, både når det gjelder transformering av data fra dokumentet til databasesystemet og motsatt. Ved transformering av data fra et XML-dokument til databasesystemet, er det ofte akseptabelt å se bort fra informasjon om dokumentet. Dette kan være dokumentets navn og DTD, om dokumentets fysiske

struktur, slik som enhets definisjoner og bruk av disse, CDATA[1] deler og kodeinformasjon.

Ved transformering av data fra databasesystemet til et XML-dokument, vil ikke dokumentet inneholde noe CDATA eller bruk av entiteter. Rekkefølgen som søskenelementer og attributter oppstår i er sannsynligvis i den rekkefølgen som dataene ble returnert fra databasesystemet.

Dette viser at det som er viktig å ta i betraktning for datatransformerings programvaren er den hierarkiske rekkefølgen. Det skal også være nevnt at ved å se bort fra den nevnte informasjonen vil det være vanskelig å round-trippe-dokumentet. Å round-trippe et dokument vil si at dokumenter lagret i databasesystemet blir rekonstruert til nøyaktig det samme dokumentet når det hentes ut fra databasesystemet igjen.

3.2.3 Menneskelesbare dokumenter

Menneskelesbare dokumenter er beregnet for at mennesker skal kunne lese dem. De blir også kalt dokumentorienterte dokumenter. De er karakterisert ved mindre regulær eller irregulær struktur, grovgranulerte data og mye blandet innhold. Rekkefølgen på når søskenelementer og PCDATA[2] oppstår er neste alltid viktig. Eksempler kan være bøker, e-post, reklame og XHTML-dokumenter.

```
<?xml version=”1.0”?>
<bøker>
  <bok>
   <tittel>XML for nybegynnere</tittel> er en bok for nybegynnere i XML,
  og er skrevet av <forfatter>Jimmy West</forfatter> i <år>2001</år>.
  Her kommer en kort beskrivelse av <tittel>XML for nybegynnere
  </tittel> :<beskrivelse><avsnitt>Denne boken vil gi en kort innføring i
  hva <standard>XML</standard> er. Det vil bli gitt en omfattende
  forklaring av de ulike deler av <standard>XML</standard> med
  eksempler for hver av dem. For dem som allerede har kunnskap om
  <standard>HTML</standard>, kan noen kapitler hoppes over.
  </avsnitt></beskrivelse>
  </bok>
  <!-- Flere bøker -->
</bøker>
```

3.2.4 Lagring av menneskelesbare dokumenter

Menneskelesbare dokumenter har i motsetning til dataorienterte dokumenter ikke opphav i databasesystemet. For enkle eller små dokumentsett finnes det mange måter å lagre dokumentene på. En måte er å lagre dokumentene på er å lagre dem som BLOB (Binary Large Object) i et relasjonsdatabasesystem.

Om man trenger flere egenskaper enn de enkle systemene vi har nevnt over har, er native XML-databasesystemer et godt alternativ. De er laget spesielt for å lagre XML-dokumenter.

1. CDATA, Character Data, er tekst som ikke skal bli parset av XML-parseren. Dette kan for eksempel være skript eller programkode.
2. PCDATA, Parsed Character Data, er selve innholdet mellom start- og slutt-tagger.

3.2.5 Hybrid av menneskelesbare og dataorienterte dokumenter

I denne blandingen av menneskelesbare og dataorienterte dokumenter vil det både finnes regulære og irregulære elementer. Andre egenskaper vil være som nevnt over i kapittel 3.2.1 og kapittel 3.2.3. Eksempler på denne typen dokumenter er publikasjoner og nettbokhandler.

```
<?xml version="1.0"?>
<bøker>
  <bok>
    <tittel>XML for nybegynnere</tittel>
    <forfatter>Jimmy West</forfatter>
    <år>2001</år>
    <beskrivelse>
      <overskrift>Kort omtale av <tittel>XML for nybegynnere</tittel>:
      </overskrift>
      <avsnitt>Denne boken vil gi en kort innføring i hva <standard>XML
      </standard> er. Det vil bli gitt en omfattende forklaring av de ulike
        deler av <standard>XML</standard> med eksempler for hver av
      dem. For dem som allerede har kunnskap om <standard>HTML
      </standard>, kan noen kapitler hoppes over.</avsnitt>
      <!-- Flere avsnitt -->
    </beskrivelse>
  </bok>
  <!-- Flere bøker -->
</bøker>
```

3.2.6 Lagring av hybriden

Som en generell regel kan man si at dataorienterte dokumenter blir lagret i et tradisjonelt databasesystem, slik som et relasjon, objektorientert eller hierarkisk databasesystem. Mens menneskelesbare blir lagret i et native XML-databasesystem eller et content management system. Dette er som sagt en generelle regler, og det finnes flere unntak.

Hvis man vil lagre et hybrid-dokument i en RDBMS, kan en løsning være å dele opp dokumentet. Da kan alt som er dataorientert i dokumentet normaliseres i ulike kolonner, mens det som er av blandet innhold lagres samlet i en kolonne som en BLOB.

3.3 Lagring av XML-dokumenter

XML begynner å bli mer og mer vanlig til blant annet å transportere data. Dette gjør at mengden av XML-dokumenter øker, noe som igjen fører til at det trengs gode måter for å lagre XML-dokumentene på. Det har kommet flere ulike løsninger på denne utfordringen. Man kan dele disse løsningene inn i fire kategorier[10];

- *XML-enabled databasesystemer:* er databasesystemer med utvidelser for transformering av data mellom XML-dokumenter og databasesystemer, og motsatt.
- *Native XML-databasesystemer:* er databasesystemer som er laget med XML som fundamental lagringsenhet.
- *Middleware:* er programvare som blir kalt opp fra databasesystemet for å transformere data mellom XML-dokumenter og databasesystemet.
- *Content Management Systems:* er systemer for å lagre, hente ut og å sette sammen dokumenter av ulike dokumentfragmenter.

Av disse fire kategoriene er XML-enabled databasesystemer og native XML-databasesystemer de som blir mest brukt og vil bli nærmere beskrevet her.

3.3.1 XML-enabled databasesystemer

XML-enabled databasesystemer[21] er databasesystemer som inneholder utvidelser slik at det er mulig å transportere data mellom XML-dokumenter og databasesystemet selv. Disse utvidelsene kan gjelde for både relasjons, objektorienterte og hierarkiske databasesystemer. For disse databasesystemene vil det vanligste være å lagre og motta dataorienterte dokumenter. Grunnen til dette er at XML som regel blir brukt som et utvekslingsformat, hvor dokumentene stort sett er dataorienterte.

Det er en del ulikheter mellom XML-data og relasjons databasesystemer, som gjør at det er en del operasjoner som må utføres for å kunne legge inn XML i relasjons databasesystemer. Disse ulikhetene sammenfattet i Tabell 1. For å kunne lagre XML-dokumenter i XML-enabled databasesystemer, må det utføres en mapping fra dokumentskjemaet til databaseskjemaet. For relasjons databasesystemer er det også et alternativ å lagre hele XML-dokumentet i en kolonne på en rad i relasjonen. Det første alternativet krever at databasesystemet har gode mappeoperasjoner. Problemene for mappingen ligger i at XML er et merket tre, mens en relasjon er en tabell. En trestrukturen representere en irregulær struktur, mens tabeller har en regulær struktur. Det må altså lagres irregulære data i en regulær struktur.

TABELL 1. Ulikheter mellom XML-data og relasjons databaser[46].

XML	*RDBMS (Normalisert)*
Data i en hierarkisk struktur	Data i flere tabeller
Noder med elemter- og/eller attributt-verdier	Kolonneplass med en enkle verdi
Elementer kan være nøstede	Atomiske kolonneplass-verdier
Elementers rekkefølge har betydning	Rader/kolonners rekkefølge har ingen betydning
Skjema er valgfritt	Skjema er påkrevd
Direkte lagring/uthenting av enkle dokumenter	Join er påkrevd for uthenting av enkle dokumenter
Spørrespråk benytter XML-standarder	SQL-spørringer med utvidelser for XML

Det er flere metoder for mapping som er foreslått. To av de mest sentrale er:

- *Template-driven mapping:* Her er ikke selve mappingen mellom dokumentet og databasesystemet definert på forhånd. Den består i stedet for av regler med en rekke kommandoer som kjøres. Disse kommandoene består blant annet av SQL-uttrykk. Denne mappe-metoden blir mest brukt til å transformere relasjonsdata til XML-dokumenter. Fordelene med metoden er at den er veldig fleksibel og man kan sette resultatsett fra et SQL-uttrykk hvor man vil i XML-dokumentet. Videre kan man benytte både løkker, if-uttykk, variabler og funksjoner. Her vises et eksempel på å et select-uttrykk som er flettet inn i en regel:

```
<?xml version="1.0"?>
<student_liste>
  <overskrift>Følgende studenter skal være med på klasseturen:
  </overskrift>
  <select_uttrykk>SELECT navn, adresse, telefon, epost FROM
     student</select_uttrykk>
  <avslutning>De andre som fortsatt vil være med må betale
     depositum.</avslutning>
</student_liste>
```

- *Model-driven mapping:* Her blir dataene modellert i forhold til en forhåndsdefinert modell, og denne er mappet til databasesystemet. Metoden er ikke så fleksibel som den template-driven mappingen, men den er desto mer enkel. Man kan bruke to ulike modeller, enten en som er tabellbasert eller en som er objektbasert. Dette passer for henholdsvis relasjonsdatabasesystemer og objektrelasjonsdatabasesystemer.

Tabellbasert modell:

```
<database>
  <tabell>
    <rad>
      <kolonne1>...</kolonne1>
      <kolonne2>...</kolonne2>
    </rad>
  </tabell>
</database>
```

Objektbasert modell:

```
                 Bok
          /       |       \
Innholds-     Kapittel   Kapittel
fortegnelse
```

Som sagt over kan XML-dokumentet også lagres som et helt dokument i en BLOB. Dette vil gjøre at man får problemer med å bruke SQL. Med vanlig SQL kan man ikke spørre i slike spesielle datatyper, men må lage utvidelser som forstår XML-strukturen.

For dataorienterte dokumenter er som sagt XML-enablede databasesystemer en god løsning. Når det gjelder menneskelesbare dokumenter blir det vanskelig å transformere dokumentet til databasesystemet. Da vil man enten oppnå et stort antall kolonner med null verdier eller et stort antall tabeller. Alternativet er en blob, men da må man benytte utvidelser av spørrespråket. Det vil ofte ta lang tid og er veldig ineffektivt.

3.3.2 Native XML-databasesystemer

Native XML-databasesystemer (NXD)[21] har XML-dokumenter som fundamentale lagringsenheter. Dette gjør at den kan tilby mer XML-spesifikke egenskaper enn XML-enablede databasesystemer. Dermed vil dette databasesystemet i flere tilfeller være en del raskere enn XML-enablede databasesystemer.

Native XML-databasesystemer kan deles inn i to hovedkategorier:

- *Tekstbasert lagring:* her blir hele dokumentet lagret i tekstform, og en form for databasefunksjonalitet tilbys ved å få tilgang til dokumentet.
- *Modellbasert lagring:* her vil en binær modell av dokumentet bli lagret i et eksisterende eller vanlig datalager. Modellen kan være DOM eller en variant av den. En DOM kan eksempelvis kobles til relasjonstabeller som elementer, attributter og enheter, eller en DOM kan lagres i en forhåndsparset form i et datalager skrevet spesielt for denne oppgaven.

Det har ikke blitt fastsatt en formell teknisk definisjon på hva native XML-databasesystemer er, men medlemmer av XML:DB mailing list[45] har laget en mulig definisjon. Den lyder som følger: (Vi gjør oppmerksom på at vi har tatt oss friheten til å oversette til norsk.)

Definerer en (logisk) modell for et XML-dokument -- i motsetning til dataene i det dokumentet -- og lagrer og gjenfinner dokumenter i henhold til den modellen. Som et minimum må modellen inneholde elementer, attributter, PCDATA og dokumentrekkefølge. Eksempler på en slik modell er XPath datamodellen, XML Infoset og modellen implisert av DOM og hendelsene i SAX 1.0.

Har et XML-dokument som dens fundamentale enhet av (logisk) lagring, likt som relasjonsdatabasesystemer har en rad i en tabell som dens fundamentale enhet av (logisk) lager.

Det kreves ikke å ha en spesiell underliggende fysisk lagringsmodell. For eksempel kan det bli bygget på en relasjon, hierarkisk eller et objektorientert databasesystem, eller man kan bruke et passende lagringsformat som indekserte, komprimerte filer.

Persistent DOM, PDOM, er en spesialisert type av native XML-databasesystemer som implementerer DOMen på et stabilt lager. DOM-treet som blir returnert fra PDOM er ”i live”, i motsetning til de fleste andre native XML-databasesystemer som returnerer DOM-trær. Forskjellen er at når man gjør forandringer på DOM-treet fra PDOM vil det reflekteres direkte i databasesystemet. I de fleste native XML-databasesystemer er

DOM-treet som blir returnert en kopi, og forandringer som gjøres blir gjort gjennom et XML-oppdaterings språk eller ved å bytte ut hele dokumentet. PDOM tilbyr persistent lagring for applikasjonsdataene og brukes som et virtuelt minne for applikasjonsprogrammet.

De fleste native XML-databasesystemene benytter seg i dag av XPath som spørrespråk[6]. XPath var egentlig ikke ment som et database spørrespråk, og har derfor store mangler når det blir brukt alene som et spørrespråk. Det trengs blant annet en utvidelse når det gjelder spørringer mellom samlinger av XML-dokumenter. Andre begrensninger XPath har går på gruppering, sortering, join mellom ulike dokumenter og støtte av datatyper. Her ventes det å bli en forbedring når XQuery kommer. XQuery kommer til å bli den nye standarden innen XML spørrespråk og blir nærmere beskrevet i kapittel 6.2.1.

Fordelen ved å benytte native XML-databasesystemer er at man kan lagre alle typer XML-dokumenter i databasesystemet på en enkel måte. Man kan på en grei måte hente ut dokumenter i deres originale form. Den strukturelle informasjonen blir opprettholdt. Man kan lagre, spørre og hente ut struktur og innhold fra databasesystemet.

En ulempe med NXD er at de fleste NXD kan bare returnere dataene som XML. Hvis man vil ha dataene på et annet format, må først XML-dokumentet parses før den kan brukes. Dette kan bli en stor ulempe for enkelte systemer i forhold til det å bruke XML-enabled databasesystemer.

En annen ulempene ved å bruke NXD er at databasesystem-administratorene kan få en mer komplisert jobb. Han må håndtere forskjellige databasesystemer. Det vil også bli ekstra vedlikehold for it-personalet. Teknologien er fortsatt ganske ny og uprøvd. Dette fører også til at produktene som fins er dyre.

NXD holder heller ikke mål når beslektede dokumenter er lagret. I forhold til relasjons databasesystemer er ennå NXD ikke optimalisert for store mengder data. NXD tilbyr også bare et enkelt hierarkisk view[1] av dataene. I tillegg er det vanskelig å sy sammen data fra deler av forskjellige dokumenter på en fleksibel måte.

3.3.3 Forskjellen mellom XML-enabled og native XML-databasesystemer

Både XML-enablede og native XML databasesystemer har sine fordeler og ulemper. De viktigste av dem sto under beskrivelsen av disse databasesystemene, men vi vil her komme med en del sammenligninger av de to databasesystemene.

I teorien kan XML-enabled databasesystemer bevare den fysiske strukturen til dokumentet, men i praksis har dette aldri blitt gjort. Native XML-databasesystemer gjør i midlertidig dette, både enhets anvendelse, CDATA deler, kommentarer, DTDer ol.

1. View er måten dataene presenteres for en bruker. Dette virker inn på hvordan dataene blir lagret i databasen.

Native XML-databasesystemer kan lagre XML-dokumenter uten å kjenne til DTDen . XML-enabled databasesystemer kunne generert skjema under kjøring, men dette er svært upraktisk i praksis, spesielt ved håndtering av skjemaløse dokumenter.

Det eneste grensesnittet til data i native XML-databasesystemer er XML og relaterte teknologier, som XPath, DOM eller en XML-basert API. XML-enabled databasesystemer på den andre siden vil sannsynligvis tilby direkte tilgang til dataene, som gjennom en ODBC.

Videre er de tradisjonelle relasjonsdatabasesystemene med XML-utvidelser, en grei løsning når XML-dokumentet er nokså lite eller når det enkelt kan bli delt opp gjennom XSLT. Men da bør ikke raske spørringer over store dokumentsett bør da ikke være et stort krav. Hvis dette er et krav vil det være en bedre løsning med native XML-databasesystemer.

Begge databasesystemene får problemer når det gjelder oppdatering av XML-dokumentene. Når det gjelder XML-enabled databasesystemer vil det kunne oppstå problemer hvis XML-dokumentet er lagret i en BLOB. Da må hele dokumentet hentes ut. Deretter må det lages et nytt dokument med endringen og dette nye dokumentet må legges tilbake i databasen. Dette gjelder også native XML-databasesystemer hvor man må hente ut hele dokumentet for å endre det.

Andre fordeler med å lagre data i NXD er at det er relativt raskt å hente ut XML-dokumenter. I flere tilfeller vil det være en del raskere enn ved XML-enabled databasesystemer, men dette er avhengig av hvordan XML-dokumentene er fysisk lagret. Grunnen til dette er at flere NXD lagrer fysisk hele dokumentet på en plass eller de har fysiske pekere mellom delene av dokumentet. Dermed kan dokumenter bli hentet ut enten uten join eller med fysiske join. Begge disse alternativene er raskere enn de logiske join som relasjons databasesystemene bruker. Ulempen for NXD her er hvis dokumentet skal hentes ut i en annen rekkefølge enn de ligger på disk. Dokumentene blir lagret etter hvilket view man vil ha av dem. Vil man ha et annet view, vil ytelsen antakeligvis bli lavere enn ved relasjons databasesystemer.

4.0 Temporaldatabasesystemer

Begrepet temporaldatabasesystemer[25,26] dekker alle databasesystemer som krever en form for tidsaspekt for å organisere dataene. Eksempler på dette kan være finansiellsystemer, reservasjonssystemer, helsevesenet og forskningsdatabasesystemer.

Formelt kan man si at tid er en ordnet sekvens av punkter med en bestemt granularitet. Denne granulariteten kan f.eks. være minutt, dag eller uke. Dermed kan man si at det som skjer innenfor samme punkt, eksempelvis minutt, skjer samtidig.

Begrepet hendelser kan deles i to deler, punkthendelser og varighetshendelser. Ved punkthendelser er det noe som skjer på et bestemt tidspunkt, mens varighetshendelser er en hendelse som er gjeldende over lengre tid. Varighetshendelsen går altså over en tidsperiode med en start- og en slutt-tid.

Vi vil i dette kapitlet starte med å se på de vanligste tidsaspektene som brukes. Deretter vil vi gå inn på ulike typer versjoneringsteknikker i kapittel 4.2. I de to siste delkapitlene vil vi komme inn på hvordan man implementerer tid i ulike typer databasesystemer. Vi vil ta for oss relasjonsdatabasesystemer og objektorienterte databasesystemer.

4.1 Tidsaspekter: gyldighetstid og transaksjonstid

I databaseterminologien snakker man om ulike verdener. Det er databaseverdenen, som forteller når ulike hendelser skjer i databasesystemet. Videre kan man snakke om den virkelige verdenen, som er hva som skjer i det virkelige liv uavhengig av når det blir registrert i databasesystemet. Dette gir oss ulike aspekter på tid. De mest populære aspektene som passer med disse to verdenene er gyldighetstid og transaksjonstid. Gyldighetstid forteller når en hendelse skjer i den virkelige verdenen, mens transaksjonstid sier på hvilket tidspunkt informasjon blir lagt i databasesystemet. Her er det snakk om både data som blir lagt inn i databasesystemet for første gang og data som blir oppdatert eller slettet.

En tredje type tid som bør nevnes er brukerdefinert tid. Dette er tid som brukeren selv har lagt inn i databasen, og ikke i DBMS. Denne tiden kan heller ikke tolkes av DBMS. Vi vil ikke komme noe nærmere inn på brukerdefinert tid.

Mens transaksjonstid beskriver når data blir registrert i databasesystemet, er det mer å ta hensyn til med gyldighetstid. Med gyldighetstid må man ikke bare forholde seg til når hendelsene inntreffer, men man må også se på hvilket tidspunkt en hendelse blir registrert i databasesystemet. Hvis en hendelse blir lagt i databasesystemet før den faktisk inntreffer kalles det en proaktiv oppdatering, mens det kalles en tilbakevirkende oppdatering hvis den blir registrert etter at hendelsen skjer. Blir databasesystemet oppdatert samtidig som det skjer i den virkelige verdenen, kalles det en samtidig oppdatering.

Ved hjelp av disse aspektene, gyldighetstid og transaksjonstid, har vi ulike temporaldatabasesystemer. Databasesystemer som registrer gyldighetstid kalles gyldighetstidsdatabasesystemer. Databasesystemer som trenger transaksjonstidspunktet blir det kalt transaksjonstidsdatabasesystem. Det finnes også

databasesystemer som trenger begge aspektene, disse blir kalt bitemporale databasesystemer.

Hvis man benytter gyldighetstidsdatabasesystemer vil man aldri kunne få ut den nøyaktige databasetilstanden. Grunnen er at man senere kan legge inn data som var gyldig på det tidspunktet man hentet ut databasetilstanden, og dermed vil man endre den. Dette vil ikke være tilfelle ved transaksjonstidsdatabasesystemer. Her vil man hele tiden kunne hente ut den nøyaktige databasetilstanden. Dette er viktig om man vil gjøre en rollback på tilstanden til databasesystemet til et tidligere tidspunkt.

4.2 Versjonering

Når man støtter transaksjonstid er det viktig å velge mellom om rader, objektinstanser eller attributter selv skal versjoneres (*extension versioning*), eller om definisjonen av disse objektene skal versjoneres (*schema versioning*)[47]. Om man bruker *extension versioning* kan det også være støtte for *schema versioning*. Om det ikke er støtte for extension versioning, er det ikke relevant om det er støtte for schema versioning, siden man bare trenger å bruke den siste versjonen av skjemaet.

4.2.1 Extension versioning

Det er tre generelle tilnærminger for støtte av *extension versioning*. Den første er å modellere direkte, ikke gjøre noen forandringer til datamodellen eller spørrespråket. I den andre tilnærmingen er generelle utvidelser til datamodellen og spørrespråket utnyttet for å støtte tidsvarierende informasjon. I den tredje tilnærmingen er datamodellen og språket modifisert til eksplisitt å støtte transaksjonstid. De fleste forslag er i denne gruppen.

4.2.2 Schema versioning

I skjemautvikling kan skjemaet forandres i henhold til varierende behov i applikasjonen[22]. Da kreves det at man har opprettholdt de ulike endringene man har gjort i skjemaet. Det blir også sikret at et nytt skjema ikke påvirker de allerede lagrede dataene. Dermed vil man ha flere skjemaer som er i bruk på en gang.

4.3 Implementere tid i relasjonsdatabasesystemer

Ved relasjonsdatabasesystemer benytter man en radversjonerings metode. Det vil kort si at hver relasjon får lagt til kolonner som registrer de temporale dataene til relasjonen. I dette delkapittel beskrives hvordan de ulike tidsaspektene kan legges til et relasjonsdatabasesystem. Vi tar utgangspunkt i følgende relasjon:

Kunde { <u>Personnr</u>, Navn, Kundeinfo }

Konto {<u>Kontonr</u>, Personnr, Saldo }

Denne relasjonen vil i de følgende delene bli gjort om til gyldighetstidsrelasjoner, transaksjonstidsrelasjoner og bitemporale relasjoner.

4.3.1 Gyldighetstidsrelasjoner

For å gjøre en vanlig relasjon om til en gyldighetstidsrelasjon, må man legge til to kolonner. Disse kolonnene er Gyldig_Start_Tid og Gyldig_Slutt_Tid, og beskriver start- og sluttiden en hendelse er gyldig. Dette gjør at man kan få så mange versjoner man vil av den samme informasjonen. Den gjeldende versjonen vises ved å bruke verdien now i kolonnen Gyldig_Slutt_Tid.

Relasjonen Kunde og Konto vil få følgende databaseskjema ved gyldighetstid:

Kunde_GT {Personnr, Navn, Kundeinfo, Gyldig Start Tid, Gyldig_Slutt_Tid}

Konto {Kontonr, Personnr, Saldo, Gyldig Start Tid, Gyldig_Slutt_Tid}

For å illustrere hvordan dette vil se ut med data, kan man se på tabell 2. Her vises relasjonen Konto_GT.

TABELL 2. Relasjonen Konto_GT

Kontonr	*Personnr*	Saldo	Gyldig Start Tid	Gyldig_Slutt_Tid
3232 32 32323	120656 98745	3000	13-08-01	20-11-01
3232 32 32323	120656 98745	5000	21-11-01	now
2151 54 48548	061267 65932	10000	10-01-00	10-10-00
2151 54 48548	061267 65932	2000	11-10-00	10-11-01
2151 54 48548	061267 65932	5000	11-10-01	now

Vi ser at personen med Kontonr 3232 32 32323 øker saldoen sin fra 3000 kr til 5000 kr den 21-11-01. Etter denne datoen er det ikke registrert noen endringer på ham, som man kan se ved at det står now under Gyldig_Slutt_Tid.

Hvis tabell 2 ikke var temporal, som i relasjonen Konto, ville bare radene med verdien now i Gyldig_Slutt_Tid vært inkludert. Dermed vil det ikke være nok at bare Kontonr er primærnøkkel til relasjonen Konto. Vi må også ha en primærnøkkel som går på tiden. Derfor settes også Gyldig_Start_Tid som primærnøkkel, fordi det på ett hvert tidspunkt må være minst en gyldig versjon av hver entitet. Entitet vil her være en bestemt kunde.

4.3.2 Transaksjonstidsrelasjoner

Ved denne relasjonen trengs det også to ekstra kolonner. Disse kolonnene vil bli tilsvarende som ved gyldighetstidsrelasjonen; Transaksjon_Start_Tid og Transaksjon_Slutt_Tid. Her vil den aktuelle raden få den spesielle verdien uc (until change) i kolonnen Transaksjon_Slutt_Tid. Denne relasjonen vil være tilsvarende som relasjonen under gyldighetstid. Databaseskjemaet vil se ut som følger:

Kunde_TT { Personnr, Navn, Kundeinfo, Trans Start Tid, Trans_Slutt_Tid}

Konto_TT {Kontonr, Personnr, Saldo, Trans Start Tid, Trans_Slutt_Tid}

Her har vi forkortet Transaksjon ned til Trans, for å spare plass. Vi har også gjort tilsvarende som i gyldighetstidsrelasjonen og satt Trans_Start_Tid som primærnøkkel sammen med Personnr/Kontonr.

4.3.3 Bitemporale relasjoner

Bitemporale relasjoner registrerer både gyldighetstid og transaksjonstid. For å få til dette må man ha med alle de fire kolonnene; to fra gyldighetstidsrelasjonen og to fra transaksjonstidsrelasjon. Dermed vil databaseskjemaet se slik ut:

Konto_BT
{ Personnr, Navn, Kundeinfo, Gyldig_Start_Tid, Gyldig_Slutt_Tid, Trans Start Tid, Trans_Slutt_Tid }

Avdeling_BT
{ Kontonr, Personnr, Saldo, Gyldig_Start_Tid, Gyldig_Slutt_Tid, Trans Start Tid, Trans_Slutt_Tid }

Her er primærnøklene Personnr/Kontonr og Trans_Start_Tid. Grunnen til at man bruker Trans_Start_Tid er for å indikere den gjeldende tilstanden til databasesystemet.

4.3.4 Mulige måter å implementere tid i relasjonsdatabasesystemer

Det finnes flere muligheter for hvordan rader kan lagres i en temporaldatabasesystem. En mulighet går ut på at alle rader lagres i den samme tabellen, som vist i eksemplene over. Det er en ineffektiv metode, siden tabellen vil bestå av veldig mange rader med små endringer i hver. Dette kunne man se ut av relasjonen Konto_GT i tabell 2 hvor det er lite endring for hver rad. Når saldoen endrer seg for kunden med kontonr 3232 32 32323, blir det lagt til en ny rad i relasjonen selv om det bare er to kolonner som endrer seg.

En annen mulighet er å lage to tabeller. En tabell som består av de radene som er gjeldende, og en tabell for resten av radene. Dette gjør at man kan ha ulike aksesserings stier. Et annet forslag er å legge til enda en tabell. Denne tabellen kan da bestå av de korrigerte radene hvor Transaksjon_Slutt_Tid ikke har verdien uc.

4.4 Implementere tid i objektorienterte databasesystemer

En alternativ metode til radversjonerings metoden fra forrige kapittel er attributt versjonisering. Denne metoden kan brukes i databasesystemer som støtter kompleksstrukturerte objekter, som objektorienterte databasesystemer eller objektrelasjons databasesystemer.

Metoden går ut på at et komplekst objekt blir brukt til å lagre alle temporale endringer til et objekt. Hvert attributt som endres over tid kalles et tidsvarierende attributt. Verdien til disse attributtene blir versjonisert over tid ved å legge til tidsperioder til attributtet. Dette kan vises ved et eksempel:

```
class Temporal_Saldo
{
    attribute   Date              gyldig_start_tid;
    attribute   Date              gyldig_slutt_tid;
    attribute   float             saldo;
};
class Temporal_Kontonr
{
    attribute   Date              gyldig_start_tid;
    attribute   Date              gyldig_slutt_tid;
    attribute   Kontonr_GT        saldo;
};
class Temporal_Levetid
{
    attribute   Date              gyldig_start_tid;
    attribute   Date              gyldig_slutt_tid;
};
class Kunde_ooGT
{
    attribute   list<Temporal_Levetid>    levetid;
    attribute   string                    navn;
    attribute   string                    personnr;
    attribute   list<Temporal_Saldo>      saldo_historie;
    attribute   list<Temporal_Kontonr>    kontonr_historie;
};
```

Her vil klassen Kunde_ooGT tilsvare relasjonen Kunde_GT. Den har to tidsvarierende attributter; saldo_historie og kontonr_historie. Hvis man ser på saldo_historie, finnes det en egen klasse for dette attributtet; Temporal_Saldo. Denne klassen inneholder start- og sluttid, samt verdien i det gitte tidsrom. Dette vil være likt for alle tidsvarierende attributter. Klassen Kunde_ooGT inneholder da en liste-type for hver av disse attributtene, med alle endringene til attributtet listet opp. Dette gjør at selv om man endrer et attributt mange ganger, trenger man ikke føre opp de andre attributtene like mange ganger. Dermed kan endringer av de ulike tidsvarierende attributtene skje asynkront.

Ved denne metoden må man også legge til et ekstra attributt med en egen klasse. Dette er Temporal_Levetid klassen. Denne gjelder for hele objektet av klassen Kunde_ooGT. Den har en start- og sluttid som viser det tidsrommet som hele objektet eksisterer. I eksemplet over vil dette altså være hvor lenge hver av personene er kunde i banken som databasesystemet gjelder for.

Eksemplet over viser hvordan man kan trekke inn gyldighetstid i objektorienterte databasesystemer. Dette vil se tilsvarende ut ved transaksjonstid og ved blanding av begge disse aspektene.

5.0 Temporale XML-databasesystemer

Det er flere grunner til å se på problemet med versjonshåndtering for XML-dokumenter[18,18,19,20]. En av dem er at tradisjonelle applikasjoner betror seg til versjonshåndtering. Når da disse applikasjonene blir mer og mer webbaserte, økes bruken av XML som et representasjons- og utvekslingsformat. Også nye applikasjoner trenger mer versjonshåndtering for XML. Spesielt er det viktig å holde orden på lenkene mellom webdokumenter. Det er et økende problem at URL-er blir ugyldige når andre dokumenter lenker til dem. Spesielt gjelder dette for søkermotor som gir ugyldige lenker direkte til brukerne.

Å bytte ut den gamle versjonen med en ny versjon på den samme URL, løser ikke hele dette problemet. Det er fordi det ikke er sikkert at den nye versjonen inneholder de samme nøkkelord som den gamle versjonen. Den ideelle løsningen vil være et versjonshåndteringssystem som støtter flere versjoner av det samme dokumentet, og samtidig unngår duplikat lagring av de delte segmenter. For å håndtere lenkeholdbarhet vil profesjonelle sites[1] støtte seg til dokument-versjonering, og samtidig støtte søking og spørring i multiversjonerte dokumenter.

I dette kapitlet vil vi gå gjennom noen av de versjonssystemer som finnes på markedet i dag. Vi starter med en beskrivelse av to tradisjonelle dokument-versjoneringssystemer. Disse har en del ytelsesproblemer, og det har derfor kommet andre metoder som har tatt over for dem. En av disse er UBCC som vil være den neste metoden vi presenterer. Denne metoden er effektiv ved enkle spørringer, men den takler ikke komplekse spørringer. Dermed ble metoden SPaR laget, som behandler komplekse spørringer på en god måte. Denne metoden vil være den siste vi beskriver i dette kapitlet.

5.1 Tradisjonelle dokument-versjoneringssystemer

Vi vil her presentere to tradisjonelle dokument-versjoneringssystemer, RCS og SCCS[18,20]. Disse metodene gjør bruk av edit script[2] (redigeringsskript) til å representere endringer mellom versjonene til et dokument. RCS lagrer den nyeste versjonen intakt, mens de forrige versjonene blir representer som reverse edit script. SCCS setter redigeringsoperasjoner inn i det originale dokumentet, og assosierer et par av tidsstempler til hvert dokumentssegment. Disse tidsstemplene representerer hele livssyklusen til dokumentsegmentet. For å hente ut en versjon må hele det originale dokumentet skannes gjennom og alle de gyldige segmentene hentes ut.

Begge disse to løsningene på versjonering gjør at det av og til må leses segmenter som ikke er gyldige i versjonen som skal hentes ut. Dette gjør at det blir en del ekstra kostnader for å hente ut versjonen. For RCS vil den totale I/O-kostnaden være proporsjonal med størrelsen på den nyeste versjonen, pluss størrelsen på endringene fra den uthentede versjonen i forhold til den nyeste. Dette gjør at RCS er en god løsning når endringene mellom versjonene er minimale, mens den gjør det verre og verre etter

1. Sites er en enkel eller en samling av web-sider.
2. Edit script brukes til å lagre endringene fra en versjon til en annen. Dermed slipper man å lagre hele versjonen for alle versjonene.

hvert som endringene øker. For SCCS vil kostnadene bli enda større, da må hele dokumentet med redigeringsoperasjonene leses. RCS har forsøkt å gjøre kostnadene som kommer ved uthenting av en versjon lavere. De har lagt en indeks på de gyldige segmentene for hver versjon.

En annen svakhet med RCS og SCCS er at den logiske strukturen til det originale dokumentet ikke blir opprettholdt. Dette gjør at det blir vanskelig med søk som går på dokumentstrukturen. Slik søk vil også være kostbare å støtte.

5.2 Usefulness-Based Copy Control

UBCC[18,20] er en annen metode for å lagre versjonerte dokumentobjekter. Strategien garanterer at den totale I/O-kostnaden er lineær med størrelsene til versjonen som skal hentes ut. I motsetning til RCS og SCCS hvor størrelsen på I/O-kostnadene blir det totale antall forandringer mellom den siste og den ønskede versjonen. UBCC grupperer dokumenter ved å bruke Page Usefulness.

Page Usefulness tar utgangspunkt i et forward edit script, hvor bare den første versjonen lagres i sin helhet. For hver versjon vil objektene til dokumentet ligge i en eller flere data pager. En page som er lagret for en tidligere versjon vil inneholde flere objekter som senere vil være slettet eller oppdatert og dermed ikke er gjeldende for den nyeste versjonen. Dermed kan man si at bare deler av pagen er nyttig for den nyeste versjonen. Usefulness for en page S, for en gitt versjon V, kan defineres som prosentandelen av siden som korresponderer til gyldige objekter for V. Den første versjonen er da 100% useful.

Usefulness vil synke for hver versjon. Det benyttes en minimum page usefulness, Umin, for versjonene. Når page usefulness faller under denne Umin for en page, vil alle objektene som forsatt er gyldig for denne pagen, kopieres over til en annen page.

UBCC schema lagrer edit scriptene separat fra data pagene. Metoden representerer et dokument som en ordnet liste av objekter, hvor O# viser posisjonen til objekt O i listen. En versjon Vj representert av et edit script, inneholder to typer elementer; insert og delete.

Vi skal nå vise et eksempel på hvordan UBCCs edit script fungerer. Data pagene lagrer objektene i rekkefølgen de blir laget. Dette betyr at om versjon 1 inneholder objektene A1, B1, C1, D1, E1, F1, G1, H1, I1, J1 og K1 vil disse bli lagret slik i:

Page 1	A1	B1	C1	D1
Page 2	E1	F1	G1	H1
Page 3	I1	J1	K1	L1

FIGUR 1. Versjon 1

I versjon 2 blir M2, N2, O2, P2, Q2, R2 og S2 lagt til, mens A1, B1, F1, G1 og L1 skal ikke være med. For å regne ut usefulness, må vi se hvor mange objekter som slettes i

hver page. Vi velger en Umin på 70%. Hvis vi nå ser hvordan data pagene vil se ut etter slette-operasjonene, kan vi regne ut usefulness ut i fra det:

Page 1			C1	D1
Page 2	E1			H1
Page 3	I1	J1	K1	

FIGUR 2. Versjon 2

Her ser vi at Page 1 og 2 vil få usefulness til 50 %, page 3 til 75 %. Dette betyr at page 1 og 2 faller under Umin, og derfor skal objektene fra disse pagene kopieres over til nye pager. Da vil objektene C1, D1, E1 og H1 bli kopiert til page 4. Objektene i page 1 og 2 blir så slettet.

Ut i fra dette får vi den nye versjonen:

Page 3	I1	J1	K1	M2
Page 4	C1	D1	E1	H1
Page 5	N2	O2	P2	Q2
Page 6	R2	S2		

FIGUR 3. Versjon 2

I tillegg vil det også bli laget edit script som viser operasjonene. Her vil insert(@B1, 2) bety at objektet B2 skal lagres på posisjon @B2 og har plasseringen 2 i versjon 1. For å finne posisjonen til et objekt O1 som skal settes inn i en ny versjon Vi, går man inn i den forrige versjonen Vi-1 og henter ut posisjonen til objektet som her kommer etter O1. Dette posisjonstallet brukes videre og man legger til antall innsettinger som kommer før O1, og trekker fra antall slettinger som er før O1. Svaret her vil da bli den nye posisjonen til O1.

Edit scriptene til versjon 1 og versjon 2 som ble presentert i tabellene over vil se slik ut:.

Versjon 1:

insert (@A1, 1), insert (@B1, 2), insert (@C1, 3), insert (@D1, 4),
insert (@E1, 5), insert (@F1, 6), insert (@G1, 7), insert (@H1, 8),
insert (@I1, 9), insert (@J1, 10), insert (@K1, 11), insert (@L1, 12)

Versjon 2:

delete (@A1, 1), delete (@B1, 1), delete (@C1, 1), insert (@C1, 1),
delete (@D1, 2), insert (@D1, 2), delete (@E1, 3), insert (@E1, 3),
delete (@F1, 4), delete (@G1, 4), delete (@H1, 4), insert (@H1, 4),
delete (@L1, 8), insert (@M2, 8), insert(@N2, 9), insert(@O2, 10),
insert(@P2, 11), insert (@Q2, 12), insert(@R2, 13), insert(@S2, 14)

Objektene for Vi kan være lagret i data pager som er generert i versjon V1, V2,..., Vi-1 og Vi, og disse objektene er antakeligvis ikke lagret i sin logiske rekkefølge. Dermed er det første steget å rekonstruere den logiske ordenen til objektene i Vi. Den logiske

ordenen er gjenopprettet i en *gap-filling* metode basert på edit scriptene. Dette gjøres ved å se på plasseringsnummeret de har fått og gå til tidligere versjoner hvis det mangler et nummer.

UBCC er effektiv ved lagringsnivået, men ikke på transportnivået, altså web-basert utveksling av dokumenter. Videre er UBCC gode på enkle spørringer, men takler ikke komplekse spørringer.

5.3 SPaR versjons skjema

Tradisjonelle dokumentversjonerings-teknikker som RCS og SCCS er som sagt ikke effektive for XML-dokument versjonering. Det trengs altså en ny og forbedret teknikk som er mer effektiv både på det fysiske samt det logiske nivået. UBCC klarer ikke komplekse spørringer. Dermed har metoden SPaR (Spare Preorder and Range)[17,19] kommet, som er god til nettopp dette.

SPaR versjonerings skjema har kommet med en løsning på hvordan man kan få tilgang til komplekse spørringer som involverer sti-uttrykk på versjoner. Skjemaet tildeler alle elementer i et XML-dokument en strukturkodet id og et tidsstempel. Id'en består av et Durable Node Number (DNN) og en range, og vil være uforandret for et objekt, gjennom alle versjoner av dokumentet. Elementene får tildelt DNN ut fra hvor elementet befinner seg i XML-dokumentets trestruktur. Dermed har man en stabil referanse til elementet, som kan brukes til å indeksere elementene. Range brukes for å gi effektiv støtte av path-uttrykks spørringer. Hvis man har DNN(E) og range(E), som er henholdsvis DNN og range til et element E, kan man si at en node B er etterkommer av en node A, hvis;

DNN(A) <= DNN(B) <= DNN(A) + range(A)

Elementet får også et tidsstempel som viser livssyklusen til elementet. Den består av en Vstart og en Vend som er henholdsvis start og sluttiden til et element. Et element er kalt "levende" for alle versjonene i levetiden. Om et element er satt inn i den aktuelle versjonen vil dens Vend-verdi ennå være ukjent og ha verdien now. Denne verdien sier at den aktuelle versjonen hele tiden kan øke.

Når en page går under Umin usefulness-nivå, blir alle "levende" elementer kopiert til en ny page, i rekkefølgen til SPaR-verdiene. Alle kopierte elementer fortsetter å ha samme SPaR-verdi, men blir gitt en ny levetid.

Ved rekonstruering av en gitt dokumentversjon må det først identifiseres hvilke pager som er nyttige til en gitt versjon. Deretter må den riktige rekkefølgen på elementene finnes. Dette gjøres i henhold til SPaR-numrene. Til slutt må den ordnede trestrukturen til dokumentet rekonstrueres.

For å hente ut en enkel versjon vil I/O-kostnadene bli størrelsen på versjonen delt på Umin. Dette gjør at SPaR vil være litt bedre enn UBCC når det gjelder versjons rekonstruksjon, og også når det gjelder lagringskostnader. Fordelen til SPaR er at de bare trenger å bruke range-verdiene for å rekonstruere en versjon.

En annen fordel til SPaR er at den gjør det mulig å bygge dokumentstrukturelle indekser og gir en effektiv evaluering av versjons spørringer.

6.0 Spørrespråk

I dette kapitlet vil vi komme inn på ulike spørrespråk for databasesystemene som er forklart i kapittel 2.0, kapittel 3.0 og kapittel 4.0. Vi starter med en beskrivelse av spørrespråk for de tradisjonelle databasesystemene. Spørrespråket som vil bli forklart her er SQL som er det mest brukte. Videre kommer vi inn på XML-spørrespråk. Spørrespråkene XQuery og SQL/XML vil da bli beskrevet. XQuery vil bli den kommende standarden for XML-spørrespråk. Til slutt kommer vi inn på spørrespråk for temporaldatabasesystemer. Her vil standard spørrespråket TSQL2 bli beskrevet. Denne standarden er lagt inn som en del av SQL3.

6.1 Spørrespråk for tradisjonelle databasesystemer

I dette kapitlet[26] vil vi gå gjennom spørrespråk og dets krav ved relasjonsdatabasesystemer. Når man jobber med databasesystemer er det en del grunnleggende operasjoner man må ha, for i det hele tatt å kunne bruke databasesystemet. Dette kan man kategorisere som oppdatering og gjenfinning av data. Det er tre basis oppdateringsoperasjoner; insert, delete og update. Insert brukes til å sette inn en ny rad, mens delete sletter en gitt rad. update brukes for å endre verdien til enkelt attributter i eksisterende rader.

For å spesifisere gjenfinningsoperasjoner bruker man relasjonsalgebra-operasjoner. Disse operasjonene lar brukeren lage ulike spørringer. Resultatet blir en ny relasjon, som kommer fra en eller flere andre relasjoner. De relasjonsalgebra-operasjonene kan deles opp i to grupper. Den ene gruppen går på matematiske operasjoner, som union, intersection, set difference eller cartesian product. Den andre gruppen går direkte på operasjoner rettet mot relasjonsdatabasesystemer, som select, project eller join.

select brukes til å velge en delmengde av rader fra en relasjon som tilfredsstiller en seleksjonsbetingelse. project brukes for å hente ut en delmengde av kolonner. For å kombinere relaterte rader fra to relasjoner inn til en enkel rad brukes join. Disse operasjonen vil bli illustrert når vi går gjennom SQL3 under neste delkapittel.

6.1.1 SQL3

SQL (Structured Query Language)[26,27,28] ble opprinnelig laget av IBM for å aksessere data i relasjonsdatabasesystemer. ANSI (American National Standard Institute) og ISO (the International Standards Institute) gikk sammen og lagde en standard av SQL for relasjonsdatabasesystemer. Denne standarden ble kalt SQL1 (eller SQL-86), og i 1992 kom det ut en ny versjon, kalt SQL2.

I 1999 kom SQL3 ut med mange viktige utvidelser. SQL3 blir karakterisert som objektorientert SQL. Egenskapene til SQL3 kan deles opp i relasjonsegenskaper og objektorienterte egenskaper. Relasjonsegenskapene er de egenskapene som relaterer seg til SQL sin tradisjonelle rolle og datamodell. I tillegg til de egenskaper som allerede eksisterer, har det kommet nye datatyper, sterkere semantikk, ytterligere sikkerhet og aktiv database. Egenskapene som går på den objektorienterte delen er støtte for funksjoner og prosedyrer. Videre har de lagt til en fundamental støtte ved objektorientering, nemlig strukturerte brukerdefinerte typer.

Siden SQL er definert som en standard, bruker de fleste relasjonsdatabasesystemer dette spørrespråket. Dette gjør det lettere å skifte ut et databasesystem med et annet. For brukerne av databasesystemet vil det også blir enklere, fordi de bare trenger å lære seg et spørrespråk. Videre er ikke selve språket endret særlig, bare utvidet. Dette gjør at SQL-kode som ble skrevet i SQL1 forsatt fungerer i relasjonsdatabasesystemene med støtte for SQL3.

I dag bruker de fleste DBMS stort sett SQL2. Flere av DBMS leverandørene har lagt til egne utvidelser. Ulempen med å bruke disse utvidelsene til et bestemt databasesystem er at man mister portabiliteten med andre relasjonsdatabasesystemer.

SQL sine basis byggeklosser er select-from-where-uttrykket. For å vise hvordan dette vil se ut kan vi hente ut navn på alle ansatte som jobber i en avdeling med avdnr større enn 10;

```
select Navn
from Ansatt
where AvdNr >10;
```

select-delen bestemmer hvilke kolonner man vil hente data fra, mens from-delen lister opp alle tabellene data kommer fra. Til slutt brukes where-delen for å spesifisere hvilke restriksjoner man vil ha på resultatet. Denne delen må man ikke ha med.

Det er vanlig at man vil hente ut data fra flere tabeller i det samme resultatet. For å klare dette må man benytte join. Dette kan vises ved å hente ut navn på de ansatte fra Ansatt-tabellen og hvilken avdeling de jobber i fra Avdeling-tabellen:

```
select Navn, AvdNavn
from Ansatt, Avdeling
where Ansatt.AvdNr = Avdeling.AvdNr;
```

6.2 XML-spørrespråk

Som beskrevet i kapittel 3.0 brukes XML til både lagring og utveksling av informasjon. Dokumentene som inneholder XML-dataene vil hovedsaklig være bygd opp enten som menneskelesbare dokumenter eller dataorienterte-dokumenter som beskrevet i kapittel 3.2. Dette gjør at det er viktig å ha et fleksibelt og effektivt spørrespråk for å kunne hente ut den informasjonen man vil fra XML-dokumentene. Hva man vil hente ut fra XML-dokumentene kan variere mye. Det har blant annet mye å si hvilken type dokument det dreier seg om. Eksempler på spørringer til de ulike dokumenttypene kan være:

- *Menneskelesbare dokumenter.* Her kan gjenoppretting av individuelle dokumenter, skaffe dynamiske indekser, utføre kontekstsensitive søkinger eller generere nye dokumenter, være interessant informasjon.
- *Dataorienterte dokumenter.* Spørringer her kan gå ut på å spørre i XML-representasjoner av databasesystemer, transformere data til nye XML-representasjoner eller å integrere data fra flere heterogene datakilder.
- *Hybride modell dokumenter.* For å utføre spørringer i dokumenter med ulike typer data, som kataloger, pasientjournaler, ansatt dokumenter eller firmaanalyse dokumenter.

Som man ser av punktene over, kommer det inn nye krav og behov til spørrespråket, som man ikke har ved spørrespråkene til f.eks. relasjonsdatabasesystemer[30]. For å samle disse satte W3C, i 1998, sammen en arbeidsgruppe med representanter fra både industrien, akademiske miljøer og forskningsmiljøer. Arbeidsgruppen utarbeidet et sett med krav til hvordan et spørrespråk for XML skulle se ut. Resultatet fra dette arbeidet kan man finne i [31]. XOO7 [32] har summert disse kravene i en liste som vises i tabell 3.

TABELL 3. Funksjonalitet som et XML-spørrespråk bør ha [32]

Krav nr	Beskrivelse
K1	Spørre i alle datatyper og samlinger av flere mulige XML-dokumenter
K2	Tillate dataorienterte, menneskelesbare og blandede spørringer
K3	Aksepter streaming data
K4	Støtte operasjoner på ulike datamodeller
K5	Tillate betingelser/begrensninger på tekstelementer
K6	Støtte for hierarkiske og sekvensielle spørringer
K7	Manipulere NULL-verdier
K8	Støtte kvantifikatorer (EXIST, ALL, negering) i spørringer
K9	Tillate spørringer som kombinerer ulike deler av et eller flere dokument(er)
K10	Støtte for aggregering
K11	Kunne generere sorterte resultater
K12	Støtte sammensetning av operatorer
K13	Tillate navigering (referansetraversering)
K14	Kunne bruke omgivelsesinformasjon som deler av en spørring (eks. gjeldende dag, tid)
K15	Kunne støtte XML-oppdateringer hvis datamodellen tillater det
K16	Støtte for tvang av type
K17	Ivareta strukturen til dokumentet
K18	Transformere og lage XML-strukturer
K19	Støtte for ID-generering
K20	Strukturell rekursjon

Siden man først begynte å bruke XML, har det kommet mange forslag og implementasjoner av spørrespråk for XML[33]. En del av spørrespråkene velger å bygge på SQL, som LOREL og XML-QL. Andre spørrespråk igjen lager et spørrespråk med utgangspunkt i andre modeller. Her kan XML-GL og XSL nevnes. XML-GL bygger på en grafisk representasjon, mens XSL lager ulike regler og XPath-uttrykk. Spørrespråkene har ulike formål og ingen av dem kan sies å støtte alt som kravene i tabell 3 uttrykker.

W3C jobber med XQuery som de håper skal bli standarden for XML-spørrespråk. Den vil bli nærmere beskrevet i kapittel 6.2.1.

6.2.1 XQuery

XQuery (XML Query)[11,12,29,30,31,34] ble utviklet for å samle det positive fra de XML-spørrespråkene som allerede var kommet og for å tilfredstille de krav og behov som brukerne av XML hadde. Spørrespråket er enda ikke helt ferdig, men det blir likevel sett på som standarden innenfor XML-spørrespråk. Planleggingsgruppen for XQuery representerte to ulike grupper, de som jobbet med XML som dokument og de som jobbet med XML som data. Nå når XQuery er i sluttfasen, mener arbeidsgruppen å ha klart å få med alle krav og behov til begge gruppene. Kravene står oppsummert i tabell 3.

I følge spesifikasjonen til XQuery[29] er XQuery designet for å være et lite og enkelt språk som er lett å implementere. Videre skal det være så fleksibelt at det dekker store deler av de XML-dokumenter som blir skrevet. Det er laget krav for både menneskelesbar syntaks, som XQuery dekker, og en XML-basert syntaks. XQuery gjør spørringer på XML uansett hvordan de er lagret: i databaser, filer, meldinger osv. Spørrespråket stammer opprinnelig fra Quilt. Quilt har hentet egenskaper fra flere ulike spørrespråk som XPath 1.0, XQL, XML-QL, SQL og OQL.

De grunnleggende byggeklossene til XQuery er:

- Path-uttrykk
- FLWR-uttrykk
- Liste-uttrykk
- Betingede-uttrykk
- Element-konstruksjon
- Kvantifiserte-uttrykk
- Datatype-uttrykk

Det vil nå gis en kort beskrivelse av de ulike byggeklossene til XQuery. Den enkleste formen for et spørreuttrykk er et XPath-uttrykk. I XQuery kan man skrive et rent XPath-uttrykk. For å illustrere hvordan XPath-uttrykk kan se ut, kan vi hente ut alle bilder av bygninger i kapittel 2 i dokumentet NTNU.xml;

```
document(“NTNU.xml“)//chapter[2]//figure[caption = “building“]
```

FLWR står for for-let-where-return, og er en viktig del av XQuery-språket. Det brukes til iterasjon og for å binde variabler til mellomresultat. Følgende eksempel illustrerer hvordan et FLWR-uttrykk kan se ut;

```
for $f in document(“bib.xml”)//forlegger
let $b := document(“bib.xml“)//bok[forlegger = $f]
where count($b) > 100
return $p
```

Oppgaven til de ulike delene i dette eksemplet:

- for: brukes for å spesifisere et sett med kartesiske rader som resten av uttrykket vil bruke. I dette eksemplet genererer for-delen en ordnet liste av bindinger av forlegger til variablen $f
- let: tilegner en verdi eller en sekvens til en variabel. Her vil det for hver binding assosieres en videre binding til en liste av alle bok-elementer med den bestemte forleggere, til $b
- where: filtrerer listen til å bare inneholde de ønskede radene. I dette tilfellet vil det være alle forlegger med flere enn 100 utgitte bøker
- return: konstruerer en resulterende verdi for hver rad

XQuery-uttrykk kan manipulere lister av verdier, med et antall ulike operatorer. Operatorer kan være avg (gjennomsnitt), union, intersection, difference og subrange. Listene kan også sorteres på et ønsket attributt. Det finnes også støtte for generelle if-then-else konstruksjoner, og distinct-values-funksjoner. I tillegg kan XQuery brukes til å konstruere nye elementer.

Kvantifiserte uttrykk er også et krav brukere gjerne vil ha i et spørrespråk. I XQuery kan man benytte to ulike; some-in-satisfies og every-in-satisfies.

Det siste punktet går på ulike datatyper. I XQuery finnes det støtte for alle datatyper som finnes i XML Schema, både primitive og komplekse typer. De konstante verdiene kan skrives på ulike måter:

- *Literaler*: Dette kan være string, integer eller float
- *Konstruktør-funksjoner*: For eksempel true() eller date(“2000-08-15“)
- *Eksplisitt-kasting*: Dette kan se ut som cast as xsd:positiveInteger(23)

XQuery 1.0[11] er forventet å komme ut i løpet av 2002. Denne versjonen vil ha et stort minus; den mangler oppdateringsfunksjonalitet. I følge lederen av XQuery Working Group, Paul Cotton, vil det ikke være noen tekniske barrierer for å få oppdateringsfunksjonalitet til XQuery, men det er mye arbeid som må gjøres. Derfor ble det bestemt, etter en lang debatt med påfølgende avstemming, at XQuery 1.0 skal komme ut uten oppdateringsfunksjonalitet. Dette gjør at den vanligste måten i dag å oppdatere XML-dokumenter på er å hente ut dokumentet, lage et nytt dokument med endringene og til slutt slette det gamle. Dette vil ta lang tid, og kan være et stort problem i større systemer med mange brukere. Da vil dokumenter være utilgjengelig for lengre tid ved oppdatering. Enkelte produsenter har laget sine egne løsninger på oppdateringsproblemet. Her kan nevnes X-Hive som har laget en egen implementasjon, XUpdate.

XQuery er en god løsning for databasesystemer hvor dokumentene er indekserte, og kan derfor aksessere innholdet i dokumentet raskt[10]. Når det gjelder uindekserte dokumenter er XQuery ingen god løsning. Videre fungerer XQuery best ved hybride-dokumenter, altså dokumenter som inneholder både menneskelesbare og dataorienterte dokumenter.

6.2.2 SQL/XML

I andre halvdel av 2000 startet NCITS H2 og SC32 et prosjekt som hadde til mål å lage et vel-definert forhold mellom XML og SQL. Etter hvert har flere firmaer, blant annet Oracle og IBM, involvert seg i prosjektet. Denne gruppen som er åpen kalles nå SQLX Group.

Muligheter som kan inkluderes i SQL/XML [1]:

- Spesifikasjoner for representasjonen av SQL-data (spesielt rader og tabeller av rader, så vel som views og spørreresultater) på XML-form, og omvendt.
- Spesifikasjoner for mappingen fra SQL-skjema til og fra XML-skjema. Dette kan inkludere mapping mellom eksisterende XML og SQL-skjema.
- Spesifikasjoner for representasjoner av SQL-skjema i XML.
- Spesifikasjoner for representasjoner av SQL-handlinger (insert, update, delete).
- Spesifikasjoner for meldingsformidling for XML når det er brukt med SQL.
- Spesifikasjoner for hvordan SQL-språket kan bli brukt med XML.

En ny del av SQL-standarden (SQL/XML) er forventet i første del av 2003.

Oracle XML DB[1] støtter SQL-style spørring, hvor brukere kan bruke XPath-notasjon til å gå gjennom XML-dokumenter. Stilen som brukes er kjent som SQL/ XML og er som beskrevet tidligere standardisert av SQLX Group. I forhold til SQL, er SQL/XML utvidet med funksjoner og operasjoner for å inkludere prosessering av XML-data i relasjonslager. SQL/XML kan benyttes når dataene er semistrukturerte og strukturerte.

6.3 Temporale spørrespråk

For temporaldatabasesystemer[14] er det å kunne å spørre på tid, ett av hovedmålene. Dette kan være spørringer som vil vite hvilke informasjon som var gjeldende på et bestemt tidspunkt eller det kan også være å finne ut når en bestemt hendelse skjedde.

Det er to spesielle utfordringer ved temporale spørrespråk. Det er tidsaspektet og identiteten til elementer i versjoniserte XML-dokumenter. Tidsaspektet går på det som er beskrevet tidligere i kapittel 4.0, altså gyldighetstid og transaksjonstid. Ulike systemer krever ulike tidsaspekter, dette må altså vurderes i forhold til spørrespråket.

Når det gjelder identifisering av elementer, er det større utfordringer. Utfordringen ligger i at normalt sett har ikke elementer i et dokument egen identitet som er persistent mellom de ulike versjonene av dokumentet. Dermed kan spørringer bli vanskelig å uttrykke, for ikke å snakke om at det tar lang tid å utføre. Dette gjelder spesielt på spørringer som vil ha ut tidspunktet et element ble laget eller hva den forrige verdien på elementet var.

Et system som løser denne utfordringen er Xyleme[53]. Xyleme forsker på å lage et dynamisk WWW XML Warehouse, som skal kunne lagre alle XML-dokumenter[36].

---

1. Oracle XML DB er XMl-delen til Oracle9i Release 2. Den blir nærmere beskrevet i kapittel 9.8.

For å kunne identifisere noder i XML-dokumentet har de en persistent identifikator, XID. Denne identifikatoren er uavhengig av tid, og vil ikke bli brukt om igjen om en node blir slettet.

Generelt finnes det to måter å spørre i temporale XML-dokumenter på. Den ene måten går ut på å oppnå et snapshot[1] av XML-dokumentet. Det blir så gjort spørringer på snapshot ved å benytte et ordinært spørrespråk (ikke-temporalt). I det andre tilfellet benyttes et spørrespråk som er spesialisert for temporale XML-dokumenter. I det første tilfellet trenger man ikke utvide eksisterende teknologier, men i det andre tilfellet må man utvikle et spesialisert spørrespråk.

I de kommersielle DBMS er det lite støtte for temporale konsepter. Grunnen til dette er at temporale konsepter er komplekse og tar mye lagringsplass. Når det gjelder objektorienterte databasesystemer er de laget for å behandle komplekse systemer[37]. Likevel er det mest forskning rundt relasjons databasesystemene, når det gjelder temporale konsepter. [38]

Det er laget et generelt spørrespråk for temporaldatabasesystemer, TSQL2, som senere er blitt en del av SQL3. Dette spørrespråket vil vi beskrive nærmere i neste delkapittel.

6.3.1 TSQL2

TSQL2 er en temporal utvidelse av SQL2, og er fullt kompatibel med den. Den er også kommet inn som en del av SQL3. Den har en formell betegnende semantikk som mapper TSQL2-uttrykk til temporale relasjonalgebra uttrykk.

TSQL2 bruker konseptet baseline clock, som tilbyr semantikken til tidsstemplene. Baseline clock relaterer hvert sekund til et fysisk fenomen. Den deler tidslinjen inn i sett med kontinuerlige perioder. Hver periode kjører på forskjellige klokker. Et synkroniseringspunkt avgrenser en periodegrense. Baseline clock og dens representasjoner er uavhengige av kalenderen.

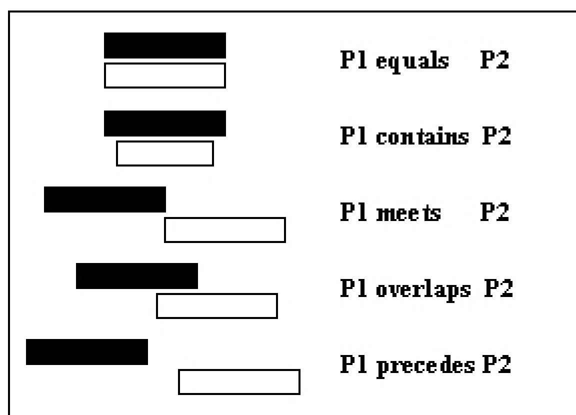
TSQL2 støtter tre tidslinjer: brukerdefinert tid, gyldighetstid og transaksjonstid. TSQL2 støtter bare en gyldighetstids dimensjon, og gyldighetstid støtten skal inkludere både fortid og framtid.

Relasjoner kan støtte gyldighetstid, transaksjonstid eller begge. Det kan også være støtte for gyldighetstid ved enten tilstands- eller hendelsesrelasjon. I tilstandsrelasjonen er hver rad tidsstemplet med et temporært element som en union av perioder. For å vise eksempel på dette kan vi ta utgangspunkt i relasjonene kunde og bankkonto som ble beskrevet i kapittel 3.0. I et temporaldatabasesystem vil et temporært tidsstemple kunne være perioder hvor en kunde har mer enn et visst beløp i saldo. I hendelsesrelasjoner er hver rad tidsstemplet med et instans sett. Eksempler på dette vil kunne være at det blir lagt til et tidsstempel på instansene når en person blir kunde i banken og når kunden tar opp lån i banken. Tidsstemplet vil da være koblet til hver rad.

1. Snapshot er informasjon som hentes ut på et bestemt tidspunkt og som raskt gir et inntrykk av hvordan situasjonen er på dette tidspunkt.

TSQL2 utvider SQL-92 syntaks med følgende konstruksjoner:

- Syntaksbegreper for å manipulere tidsstempler (ekstrahere start- og endepunkt til en periode, konstruere en periode av de to tidspunktene etc.).
- Temporale innebygde predikater som kan bli brukt i where-deler for å spesifisere temporære forhold mellom par av perioder.



FIGUR 4. Viser predikater ved ulike temporale par av perioder [2]

- En gyldig del som kan plasseres først i en spørring. Det er for å spesifisere om et spørreuttrykk skal evalueres med temporær semantikk (ingen gyldig del) eller med standard Codd semantikk (gyldig del). Den gyldige delen kommer i to former, valid snapshot hvor en snapshotrelasjon returneres og valid expr hvor en gyldighetstids relasjon er returnert med gyldighetstid definert av expr.
- Period kan følge et spørreuttrykk eller et relasjonsnavn i en from-del. Dette spesifiserer at resultatet blir samlet, altså at rader med identiske eksplisitte attributtverdier hvor gyldighetstiden overlapper eller er nærliggende, er flettet inn i enkle rader. Duplikater vil da bli fjernet.
- Andre fasiliteter som temporal ubestemthet, skjemautvikling, brukerdefinerte granulariteter og utvidbar literal-syntaks.

For å indikere at en relasjon har gyldighetstids-støtte brukes as validtime period(date). Det vil være tilsvarende for transaksjonstid, hvor nøkkelordet vil være transactiontime.

Om man legger til det reserverte ordet validtime foran et select-uttrykk spesifiserer det at spørringen skal vurderes til hvert punkt i tiden. For å modifisere kan man uttrykke validtime med et period-uttrykk. Dette kan vises med eksemplet:

```
validtime period '[2000-01-01 - 2002-01-01]' update Konto
set saldo = 150000
where personnr in (SELECT personnr from Kunde WHERE navn = 'Lars
Olsen')
```

Vi skal nå se litt nærmere på problemer med transaksjonstid. I motsetning til gyldighetstid kan ikke transaksjonstid bare bli simulert av relasjoner med eksplisitte tidsstempelkolonner. Grunnen til dette er at relasjoner med støtte for transaksjonstid er det bare mulig å gjøre tilføyninger, de vokser monotont. Spesielt siden spørre-funksjonalitet kan bli simulert på relasjoner uten temporal støtte, er det ingen måte å

begrense brukeren til modifikasjoner som sikrer at relasjonen bare vil få tilføyninger til resten av relasjonen. Man kan sette begrensninger på å benytte delete, men det er fortsatt mulig for brukeren å ødelegge transaksjonstidsstemplet via databasesystemets update og insert. Den eneste måten å sikre konsistens til dataene er å la DBMS opprettholde tidsstemplene automatisk.

For å legge til transaksjonstid til en allerede eksisterende relasjon gjør vi som følger:

```
alter table Konto add transactiontime
```

Her legger DBMS til transaksjonstid for oss.

DEL II

Programmer/produkter

For å få en oversikt over hvordan XML-støtten i ulike typer databasesystemer er implementert, har vi valgt å se på to typer databasesystemer. Vi starter med en beskrivelse av native XML databasesystem produkter. Deretter kommer vi inn på XML-enablede databasesystemer. Her vil vi først beskrive to databasesystemer i kapittel 8, før vi kommer med en lenger beskrivelse av Oracle9i i kapittel 9. Vi har valgt å ha dette i et eget kapittel, ettersom det er det databasesystemet vi har valgt å utvide.

Kap 7: Native XML-databasesystem produkter

Kap 8: XML-enabled databasesystemer

Kap 9: Oracle9i

7.0 Native XML-databasesystem produkter

I dette kapitlet skal vi gå gjennom ulike Native XML-databasesystem (NXD) produkter som finnes på markedet i dag. Vi velger å gå grundig gjennom tre NXD produkter, men en liste over en rekke NXD produkter er å finne i appendiks C. De databasesystemene vi velger å gå gjennom er Tamino XML Server, eXcelons Extensible Information Server og X-hive/DB.

7.1 Tamino XML Server

Tamino XML Server er laget av Software AG, som er medlem av W3C. Informasjonen som står i dette kapitlet er hentet fra Tamino sine egne hjemmesider[48]. Databasesystemet er basert på XML og andre åpne Internett-standarder. I desember 2001 kom de ut med en ny versjon, Tamino 3.1.

Tamino selv mener de er den første XML-plattformen til å lagre XML-informasjon uten å konvertere den til et annet format. I tillegg til velformulerte XML-dokumenter, kan databasesystemet også lagre andre ikke-XML formater. Dette kan være kontordokumenter, lyd, video, PDF ol. Videre integrerer de relasjons og objektorienterte data inn i XML-strukturer.

Databasesystemet har støtte for en rekke av W3Cs anbefalinger;

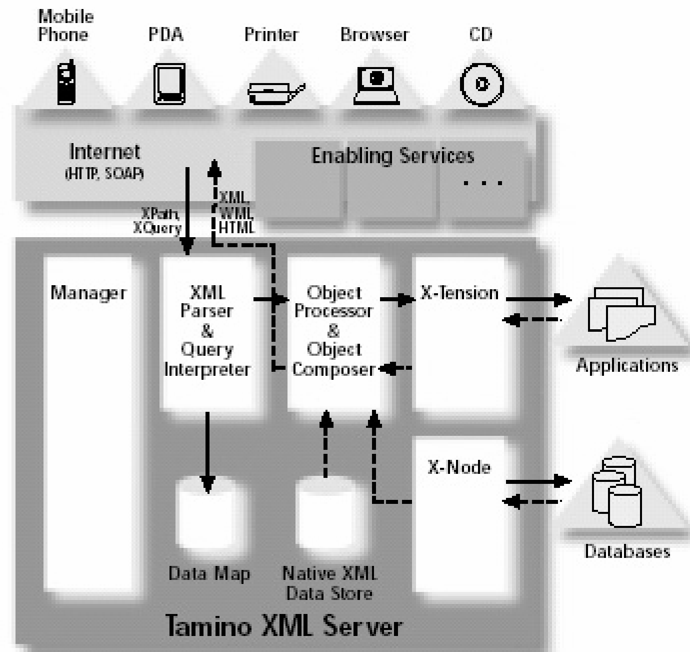
- XML
- XML Schema
- XSL og XSLT
- DOM
- XPath
- XQuery
- SOAP

For å få tilgang til databasesystemet brukes en URL ved hjelp av standard API. Tamino har støtte for de vanlige DBMS-egenskapene som recovery, transaksjonshåndtering og skalerbarhet.

Ved spørringer bruker Tamino XPath-uttrykk, men med utvidede funksjonalitet i forhold til W3C-standarden. De har også gjort det mulig å utføre søk på tekst i de lagrede dokumentene. Dette er greit å bruke ved søking i deler av XML-dokumenter eller ren tekst som ikke er indeksert.

7.1.1 Arkitektur

I figur 5 vises arkitekturen til Tamino. På den øverste grå rekken vises ulike måter brukeren kan få tilgang til dokumenter fra Tamino-databasesystemet. Dette er gjennom mobiltelefoner, Palm, Skrivere, Browsere eller på CD. Alt vil gå gjennom en HTTP-protokoll. Vi skal nå gå videre inn på hovedkomponentene til selve Tamino XML Server.



FIGUR 5. Taminos arkitektur[48]

Hovedkomponentene er:

- *XML Engine:* er den mest sentrale komponenten i arkitekturen og hovedfunksjonaliteten er lagring av XML-objekter og hente dem ut fra de ulike datakildene. For å gjøre dette har den flere underkomponenter; et integrert XML Parser, query processor, object processor, object composer og en integrert native XML-datalager. Disse delkomponentene vises i figur 5.
- *Data Map:* inneholder metadata som DTD, XML Schema, stylesheet, relationale schema osv. Den bestemmer altså hvordan XML-objektene skal se ut. Denne informasjonen brukes blant annet ved mapping av XML-objekter til andre database strukturer.
- *Tamino X-Tension*: tar seg av koblingen til eksterne applikasjoner. Den brukes også når man vil legge til utvidelser til Tamino XML Server.
- *Tamino Manager*: tar seg av administreringen til systemet, og er bygd opp som en klient-server applikasjon. Den har et grafisk brukergrensesnitt slik at systemet kan administreres over Internett.
- *Security Manager:* definerer og modifiserer tilgangsrettigheter for brukere eller brukergrupper til datalagrene.

7.2 Extensible Information Server

EXcelon Corporation er en av de største systemene innen datahåndterings programvare som er laget for å bedre ytelsen av distribuerte applikasjoner ved å benytte XML og Java (C++). De har kunder innenfor mange næringsgrener, som finans, helsevesenet og distribusjon. Nå har de kommet ut med eXtensible Information Server (XIS) 3.0 [49]. XIS er et native XML-databasesystem og en del av eXcelon's XML Platform. Plattformen består også av tre andre komponenter:

- *Business Process Manager:* en XML-basert engine for modellering og håndtering av distribuerte forretningsprosesser
- *XML development tool:* et integrert verktøy for å lage XML-baserte applikasjoner
- *Administrating Tools:* for å håndtere alle komponentene til eXcelon's XML Platform

XIS sin oppgave er å gi en distribuert og skalerbar løsning for å håndtere samlede XML data og innhold. Den er optimalisert for lagring av ethvert vilkårlig XML-dokument, uavhengig av fysisk størrelse, struktur eller skjema. Videre blir XML lagret i sin naturlige form, noe som gjør at de egenskapene som XML gir, blir bevart.

XIS har støtte for flere W3C standarder:

- XML 1.0 med namespace
- DOM Level 1 og 2
- XPath 1.0
- XSLT 1.0 med Java utvidelses funksjonalitet
- DTD
- XML Schema 1.0

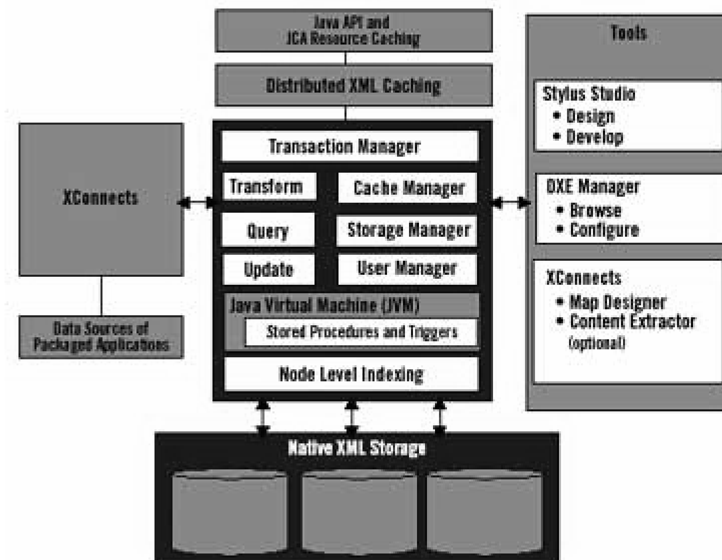
XIS er et stor databasesystem og består av den vanlige databaseteknologien som transaction consistency, online backup and restore, distributed cashing, clustering og er optimalisert for real-time data interaksjon. Den har også implementert triggere slik at Java-kode kan bli registrert og assosiert med DOM-oppdaterings operasjoner.

Det er støtte for tre ulike indekseringstyper; *value indexes for string and numeric searches*, *text indexes for word-based searches* og *structural indexes for structure-based searches*. For å få tilgang til dataene brukes XPath-uttrykk.

XIS kan mappe både mellom XML-dokumenter med ulike XML-schema, og mellom ikke-XML-schema og XML-schema. Dette gjør at data på mange ulike formater kan mappes til et XML-dokument og lagres i XIS interne datalager.

7.2.1 Arkitektur

XIS består av *design-time* og *run-time* komponenter. Design-time komponentene hjelper brukeren med designing og testing av programmer som er lagt til XIS. De XML-baserte verktøyene er Stylus Studio, Map Designer og Process Designer. Stylus Studio hjelper brukeren å lage XML-baserte programmer. Map Designer er et visuelt verktøy for å definere mapping fra back-end kilder til XML og motsatt. Mens Process Design står for lenking, gruppering og fordeling av konverteringer mellom en-til-mange og mange-til-en aggregasjoner.



FIGUR 6. Arkitektur til eXtensible Information Server[49]

Run-time komponentene står for kjøring, distribuering og skalering av programmene. Det som inngår i denne komponenten er Dynamic XML Engine (DXE), XConnect og en DXE Manager. DXE lagrer XML i sitt naturlige format og står for håndtering, transformering og levering av innholdet. Mens DXE Manager administrerer dette datalageret. XConnect står for transformering av data og innhold fra gyldige kilder til standard XML.

7.3 X-hive/DB

X-hive/DB[50] er et native XML-databasesystem som lagrer, søker i og henter ut XML-dokumenter. I motsetning til Tamino og eXcelon, er ikke X-hive/DB en del av en større plattform. Den består bare av en enkel database for XML-dokumenter. Målet til X-hive/DB er å være den mest komplette native XML-databasesysteme på markedet. Selv mener de at databasesystemet har evne til øyeblikkelig å hente ut det minste elementet i et stort XML-dokument eller en samling av dokumenter.

X-hive/DB er bygget fra bunnen av med utgangspunkt i XML. Dette gjør at det ikke er noen form fra oversetting til andre former, som tabell, lister ol. Databasesystemet er skalerbar og kan bli distribuert over flere servere.

Videre har X-hive/DB støtte for mange av W3C XML-standarder:

- XML 1.0 og XML Namespace
- DOM Level 2
- XPath
- XSLT
- XLink
- XPointer

Kjerneteknologien til X-hive/DB er som ved andre databasesystemer. Dette består av kjerne med støtte for prosessering av flere tråder, transaksjonskontroll, låsemekanismer og håndtering av hurtigbuffer. Videre har de en form for versjoneringshåndtering. Dette gjør at flere versjoner av samme dokument kan lagres. Dermed går det an å søke etter eldre versjoner, hvem som har oppdatert den og når. De benytter ikke temporale data, så det kan ikke søkes på bestemte tidspunkter.

X-hive/DB har tatt utgangspunkt i to bruksområder for XML. Disse to er at et dokument kan bli publisert på flere måter eller det kan sendes data mellom ulike applikasjoner. Begge tilfeller kan trenge en form for transformering. I det første tilfellet må XML-dokumentet transformeres til et annet format som kan publiseres. Dette kan være XHTML, PDF ol. I det andre tilfellet trenger man en transformasjon hvis applikasjonene bruker ulike XML Schema. Til denne oppgaven benyttes en integrert XSLT-engine.

For å indeksere XML-dokumentene bruker X-hive/DB flere indekseringsmetoder. Dette er *Construct Indexes*, *Content Indexes*, *Context Conditioned Indexing* og *Full Text Indexes*. Ved *Context Conditioned Indexing* kan programmereren selv bestemme hvilke noder i dokumentet som skal inkluderes eller ekskluderes i en indeks. Noder her vil være nodetypene fra XML-spesifikasjonen som element, attributt, prosessinstruksjoner, kommentarer, dokumenter og dokument fragmenter. Indeksene kan videre bli brukt av XPath.

For å legge inn data i databasesystemet har de funksjoner for å importere data fra både relasjonsdatabasesystemer og fra flate filer. For å gjøre dette brukes et sett med SQL Loader-funksjoner, som laster ikke-XML data om til et DOM-tre. Begrensningene til disse SQL Loader-funksjonene er at de bare aksepterer input fra databasesystemer med støtte for JDBC og fra filer med en sekvensiell struktur.

7.3.1 Arkitektur

X-hive/DB skriver lite om selve arkitekturen til databasesystemet. Det som er opplyst er at arkitekturen ikke er definert på forhånd, men at programmereren selv kan bestemme. Videre kan programmereren også velge om det skal brukes en klient-server eller http-basert arkitektur.

8.0 XML-enabled databasesystemer

XML-enabled databasesystemer er databasesystemer som inneholder utvidelser for overføring av data mellom XML-dokumenter og dem selv. De underliggende databasene er som regel relasjonsdatabasesystemer. XML-enabled databaser er som regel laget for å lagre og hente ut dataorienterte dokumenter. Dette er fordi data er overført til og fra brukerdefinerte tabeller, heller enn å modellere XML-dokumenter. Mange av dem kan likevel lagre XML-dokumenter i en kolonne og bruker utvidelser ved tekstprosesseringen. Vi skal i det neste ta for oss to databaser, IBMs DB2 og Microsofts SQL Server, og se hvordan de håndterer XML.

8.1 IBM: DB2 Universal Database V7.2

Den nyeste versjon av IBM sitt databasesystem er DB2 Universal Database V7.2 [23,24], og er en av de ledende databasesystemene. Det er det eneste databasesystemet som fullt ut støtter e-business, business intelligence og content management. E-business-støtten krever at databasesystemets funksjonalitet må ha en rask aksessering av nettverket og må kunne behandle store transaksjonsvolum for webbrukere.

DB2 kan kjøres på mange ulike systemer, som laptop, mobiltelefoner, store parallelle systemer osv, og minimerer derfor kostnadene ved at man kan bruke bare et databasesystem til alle applikasjoner. Databasesystemet kan også utvides til å støtte mer avanserte applikasjoner, som multimediedata. Multimediedata kan her være dokumenter, bilder, lyd, video og romlige data.

IBM har lagt til en XML Extender til databasesystemet. Denne XML Extender gir muligheter for en bruker til å kunne lagre og aksessere XML-dokumenter. Dette kommer vi nærmere inn på i kapittel 8.1.1.

8.1.1 XML-støtte

IBM har som sagt lagt til en XML Extender. Denne fungerer som et lager for XML-dokumenter og deres DTDer. Det er to muligheter for hvordan XML-dokumentene kan lagres i databasesystemet, enten som XML-kolonner eller som en XML-samling. Når XML-kolonner blir benyttet, vil det si at hele XML-dokumentet blir lagret på en kolonneplass som en XML-brukerdefinert type. XML Extender tilbyr tre ulike XML-brukerdefinerte typer. Disse er XMLCLOB, XMLVARCHAR og XMLFILE. XMLCLOB og XMLVARCHAR lagrer XML-dokumentet henholdsvis som en CLOB eller en VARCHAR, mens XMLFILE lagrer et XML-dokument som en fil på et lokalt filsystem. Brukerdefinerte funksjoner tilbyr operasjoner for å sette inn, spørre og oppdatere XML-dokumentene.

En XML-samling er et sett med relasjonstabeller som inneholder data som er koblet til XML-dokumenter. Tilgang og lagringsmetoder tilbys for å lage et XML-dokument fra eksisterende data, dekomponere et XML-dokument og å bruke XML som en utvekslingsmetode. Data Access Definition (DAD) brukes til å definere mappingen av DTD til relasjonstabeller og kolonner. XSLT og XPath syntaks benyttes til å spesifisere transformasjonen og lokasjonsstien. Det tilbys prosedyrer for lagring, uthenting, oppdatering, søking og sletting av data i en XML-samling.

Man kan generere et XML-dokument fra heterogene datakilder, som filer, DB2, Oracle og IMS Transaction Manager ved å bruke Net.Data. En annen måte å generere XML-dokumenter er å bruke prosedyrer, dxxGenXML() og dxxRetrieveXML(). Disse prosedyrene tilbys av IBM DB2 XML Extender.

8.2 Microsoft SQL Server 2000

Microsoft har laget et databasesystem som heter Microsoft SQL Server 2000[51]. Det er et rdbms og analyserings system for e-commerce, line-of-business og data warehouse løsninger. Databasesystemet kan skaleres, og også data kan skaleres over flere servere. Videre kan man enkelt aksessere data via web. Databasesystemet er begrenset til Windows-plattformen.

De har en fulltekst søke funksjon som kan brukes både på strukturerte og ustrukturerte data. Den kan blant annet søke gjennom Microsoft Office-dokumenter.

Databasesystemet har støtte for XML- og Internett-standarder, som gir muligheter for å lagre og hente ut data i XML format. De har også XML updategram som brukes til å sette inn, oppdatere og slette XML-dokumenter.

8.2.1 XML-støtte

For å hente ut data i XML-format benytter SQL Server en utvidelse av SQL. Det kan returneres data på XML-format i steden for som rader. De har også XML Updatagrams, som brukes til insert, delete eller oppdatering av SQL Server data. Det er XML Bulk Loader Component som importere XML-data til SQL Server 2000. SQL Server XML View Mapper er et verktøy for å lage XML View skjemafiler som relaterer XDR-skjema til SQL Server skjema.

SQL Server bruker teknologi som XPath, URL-spørringer og XML Updategrams i steden for relasjonsdatabase-programmering. for xml-setninger returnerer XML-data fra et select-uttrykk. Formen til det returnerte XML-dokumentet kommer an på hvilken XML-modus som er spesifisert rett etter for xml-setningen. XML-modusene som er mulige er raw, auto og explicite.

OpenXML T/SQL-funksjonen anvendes for å få tilgang på XML-data fra et relasjonsdatabasesystem. OpenXML tilbyr et relasjonsview på XML-dataene som sammen med T-SQL kan brukes til å gjøre spørringer på data fra XML-dokumenter, joine XML-data med eksisterende relasjonstabeller og oppdatere databasesystemet. OpenXML lar XML-dokumenter bli adressert med relasjonssyntaks (SQL). OpenXML er en T-SQL-funksjon som tilbyr et oppdaterbart radsett over XML-dokumenter som er i minnet.

Når det gjelder lagring av XML-data, kan SQL Server 2000 tilby et XML View av relasjonsdataene så vel som å koble XML-data til relasjonstabeller. SQL Server tilbyr også utvidelser av XDR (XML-Data Reduced) schema language. Denne utvidelsen blir brukt til å spesifisere en bidireksjonal XML-til-relasjon-kobling. XML Viewene gjør det mulig å få tilgang på relasjonsdataene som om de var XML-dokumenter.

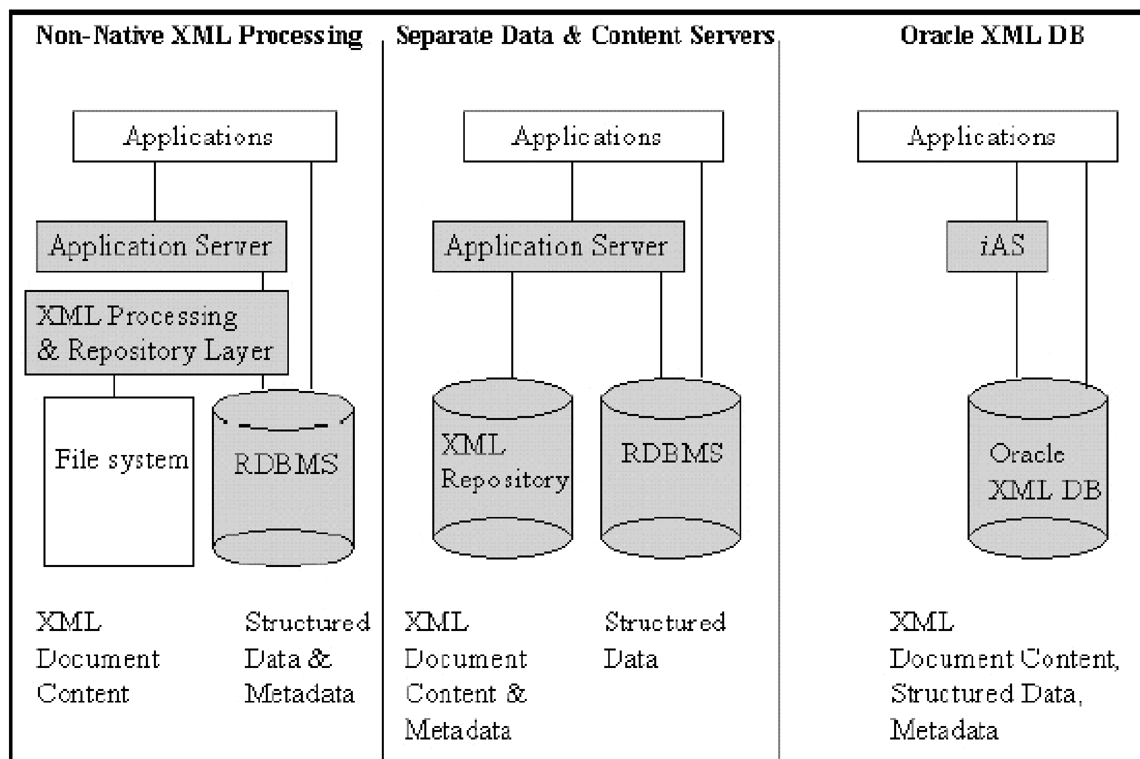
9.0 Oracle9i

Vi har valgt å bruke databasesystemet Oracle9i[56] til våre temporale utvidelser. Av den grunn vil vi gi en lenger beskrivelse av dette databasesystemet. Vil vi holde oss til de deler av Oracle9i som går på XML-støtten. Det har våren 2002 kommet ut en ny release 2 av Oracle9i. I denne versjonen er XML-støtten en god del bedre, men den kom for sent til at vi kunne ta den i bruk. Vi kommer derfor bare til å gi en kort beskrivelse av de viktigste XML-utvidelsene av den i slutten av dette kapitlet.

Først av alt vil vi starte med en oppsummering av utviklingen innen XML som Oracle har hatt. Deretter vil vi gi en dypere beskrivelse av de viktigste delene av XML-støtten for Oracle9i. Mye av det som vil bli beskrevet her vil vi få bruk for i vår løsning.

9.1 Utviklingen til Oracles XML-teknologi

I 1999 introduserte Oracle sin første versjon med støtte for XML med utgivelsen av Oracle8i. Oracle8i skulle først og fremst støtte krav til å utveksle data på XML-form og transformasjonen ble utført utenfor selve databasen. XML Developers Kit, et sett med APIer, ble tilbytt for å hjelpe utviklere å få tilgang til funksjonaliteten ved manuell koding. Oracle lagde også *inter*Media, et fulltekst indekserings produkt (nå kalt Oracle Text) som kan skille mellom XML-tagger og annen tekst.



FIGUR 7. Utviklingen til Oracles XML-teknologi (Oracle8i, Oracle9i Release 1 og Oracle9i Release 2)[55]

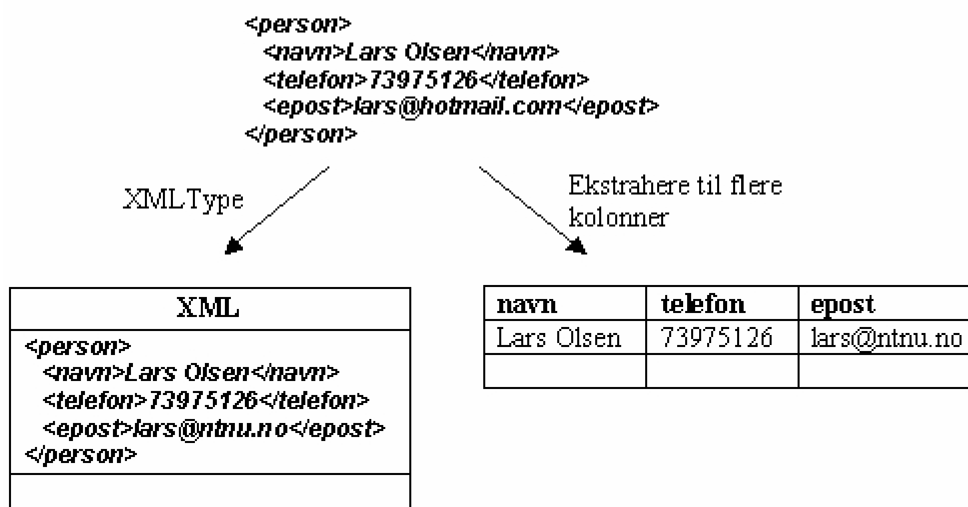
I juni 2001 ble Oracle9i utgitt. Her ble XML-støtte lagt til databasen direkte. Dette ble gjort for å forbedre utførelsen på tilgang til transaksjonelt innhold. En rekke verktøy for prosessering av XML inn i og ut av databasen ble lagt til XDKen[1]. I tillegg ble det lagt til en ny datatype for XML (XMLType) og en for logisk peking (URI-Ref), for å kunne lagre XML. Nye tabellfunksjoner ble introdusert som kunne bli brukt til å dekomponere XML-dokumenter over flere tabeller. Oracle introduserte også SQL-operatorer som gjorde det lett for SQL-programmerere å ekstrahere data som XML-dokumenter ved å bruke SQL-syntaks.

I Oracle9i Database Release 2 som ble gitt ut i første halvdel av 2002 ble XML-egenskapene kalt XML DB. Her er objektrelasjonslager med relasjonsindeks lagt til som en andre mulighet for å lagre XML. Vi kommer tilbake til hvordan Oracle9i Release 2 er utvidet i forhold til Release 1 i kapittel 9.8.

Figur 7 viser hvordan arkitekturen over lagring av XML ser ut for Oracle8i og Oracle9i, release 1 og 2. I de videre beskrivelsene av Oracle9i har vi tatt utgangspunkt i Release 1. Grunnen til dette er at det er Release 1 som er installert på skolen og vi derfor skal utvide.

9.2 Lagring av XML-dokumenter

Oracle9i sin støtte for XML består altså av generering, lagring, spørring og uthenting av XML-dokumenter. Når det gjelder lagring, kan man lagre data fra XML-dokumenter på to måter, se figur 8. Den ene måten er å lagre hele dokumentet på en kolonneplass ved å bruke XMLType eller CLOB. Det andre som kan gjøres er å bruke spesielle funksjoner for å hente ut data fra XML-dokumentet og legge det i ulike kolonner i en tabell. Disse metodene vil vi forklare i de neste to delkapitlene.



FIGUR 8. Viser forskjell på bruk av XMLType og ekstrahering av XML-dataene til flere kolonner.

1. Oracle9i Database Release 1 XDK inneholder XML Parser, XSLT Processor, XML Schema Processor, XML Class Generator, XML Transviewer Beans, XML SQL Utility og XSQL Servlet.

9.2.1 Datatypen XMLType

Datatypen XMLType kan som sagt benyttes til å lagre, spørre i og hente ut XML-dokumenter. Den lagrer selve XML-dokumentet i en CLOB i en kolonne. Dette er en god løsning ved menneskelesbare-dokumenter. For å oppdatere data som ligger i en XMLType, må hele innholdet byttes ut. Hvis innholdet er noenlunde statisk og bare oppdateres ved å skifte ut hele dokumentet, er XMLType en god løsning.

XMLType har et sett med funksjoner:

createXML(): lager XMLType-instans.

existsNode(): får inn et XPath-uttrykk og sjekker om uttrykket er gyldig. Altså om noden det spørres etter er gyldig.

extract(): får inn et XPath-uttykk og benytter det på XML-dokumentet for å hente ut bestemte fragmenter.

isFragment(): sjekker om et dokument er en fragement eller et velformulert dokument.

getClobVal(): henter ut dokumentet som en CLOB.

getStringVal(): henter ut dokumentet som en tekststreng. Funksjonen kan brukes i forbindelse med extract(), og da henter den ut resultatet fra extract() som en tekststreng.

getNumberVal(): henter ut den numeriske verdien som XMLType-instansen peker på. Kan også brukes sammen med extract(), hvor den henter ut den numeriske verdien til noden extract() peker på.

Disse funksjonen vil bli brukt i vår løsning til å behandle XML-dokumentene i databasesystemet.

For å bruke en XMLType må man først spesifisere at en bestemt kolonne i tabellen er av type XMLType.

```
create table Kunde(
  personnr    number (11),
  navn         varchar2 (20)
  kundeinfo  sys.xmltype);
```

For å legge inn et XML-dokument i kundeinfo, må man bruke funksjonen createXML:

```
insert into Kunde values (111111, ‘Lars Olsen’,
  sys.xmltype.createxml  ( ‘
    <?xml version=”1.0”?>
    <person>
      <personnr>111111</personnr>
      <navn>Lars Olsen</navn>
      <adresse>
        <gate>Olsensvei 1</gate>
        <postnr>7050</postnr>
        <poststed>Trondheim</poststed>
      </adresse>
   </person> ’ )
);
```

Om man vil søke etter dette XML-dokumentet, brukes funksjonen getClobVal().

```
select personnr, x.kundeinfo.getClobVal()
from Kunde
where navn = ‘Lars Olsen’
```

Hvis man ikke bruker funksjonen getClobVal() når man spør etter XML-dokumentet, vil man få som resultat strengen XMLTYPE().

Det kan både indekseres og søkes i XMLType-kolonner. For å gjøre dette brukes Oracle Text, som er beskrevet i kapittel 9.3 og kapittel 9.4.

En av fordelene ved å bruke XMLType, er at man kan kombinere XML og SQL. Det kan altså brukes SQL-operasjoner på XML-innhold og motsatt. En annen fordel er at det er inkludert innebygde funksjoner, indekseringsstøtte, navigering, osv.

9.2.2 Dekomponert XML

Ofte vil man lage XML-dokument av data som ligger i ulike objektrelasjons tabeller. Dette er særlig aktuelt når XML blir brukt som et utvekslingsformat, som er det vanligst bruksområde i databaser. For å generere XML-dokumenter brukes XSU (XML SQL Utility) eller SQL-funksjoner.

XSU tilbyr støtte for lagring av XML-dokumentet ved å mappe dokumentet til den underliggende lagringsplassen. Videre kan XSU også brukes til å mappe XML til objektrelasjonstabeller. Vi kommer nærmere inn på XSU i kapittel 9.7.

9.3 Indeksering

Indeksering av XMLType kan skje på to måter, nemlig funksjonelle indekser på XMLType og ved å lage tekstindeks på XMLType-kolonner. Vi skal nå se litt nærmere på disse to typene.

9.3.1 Functional Index med XMLType

Spørringer kan gjøres mye raskere ved å lage funksjonelle indekser på Extract-funksjonen av XML-dokumentet eller på ExistsNode-funksjonen. Vi skal nå se eksempler på disse mulighetene.

```
select *
from Kunde k
where k.kundeinfo.extract(‘//person/telefon/text()’).getNumberVal() = 100;
```

Vi kan her lage et functional index på Extract-funksjonen for å gjøre spørringer raskere.

```
create index telefon_indeks
on kunde (kundeinfo.extract(‘//person/telefon/text()’).getNumberVal();
```

Med denne indeksen vil SQL-spørringen benytte functional index for å evaluere predikatet, heller enn å parse hver rad i XML-dokumentet og evaluere XPath-uttrykket.

ExistsNode-funksjonen er også godt egnet til å brukes i functional index, siden den returnerer verdien 0 eller 1, avhengig av om XPath-uttrykket er tilfredsstilt i dokumentet eller ikke. Skal man for eksempel framskynde en spørring som søker etter

om XML-dokumentet inneholder et element kalt adresse (uansett på hvilket nivå), kan man bruke følgende spørring:

```
select *
from Kunde k
where k.kundeinfo.existsNode('//adresse') = 1;
```

Vi kan lage en functional index på ExistsNode-funksjonen som følger:

```
create index adresse_indeks
on Kunde (kundeinfo.existsNode('//adresse'));
```

Om vi nå gjør samme spørring som over, altså spør om det finnes et element som heter adresse, vil dette gå veldig mye raskere enn før vi laget adresse_indeks.

9.3.2 Text Index på XMLType-kolonner

Man kan lage text index på XMLType-kolonner. Oracle Text Index har i Oracle9i Database Release 1 blitt utvidet til ikke bare virke på CLOB og VARCHAR, men også på XMLType. Oracle Text Index lager automatisk XML-seksjoner, når det er definert på XMLType-kolonner.

Her viser vi et eksempel på hvordan man kan lage et indeks med Oracle Text.

```
create index kunde_text_index
on Kunde(kundeinfo)
indextype is ctxsys.context;
```

9.4 Søking i XML-data

Oracle Text kan brukes til å søke i XML-dokumenter. Mulighetene ved bruk av Oracle Text er innholdsbasert spørring, begrepssøking, tema analyse og utheving av trefford.

Brukere kan også spørre etter XML-data direkte, uten bruk av Oracle Text, men det vil gi dårligere ytelse.

Hoveddelen til Oracle Text er contains-operatoren. Denne operatoren er brukt i where-delen til et select-uttrykk, for å spesifisere spørreutrykket til en tekstspørring. Et eksempel kan være å hente ut personnr til kunden som har telefonnr lik '73975126':

```
select personnr
from Kunde
where contains (telefon, '73975126') > 0;
```

For hver returnerte rad vil contains-operatoren også returnere en verdi mellom 0 og 100. Denne verdien indikerer hvor relevant dokumentet er i forhold til spørringen, jo høyere dess bedre. Om verdien 0 blir returnert, ble det ikke funnet noen rader.

contains-funksjonen støtter XPath ved å bruke to operatorer inpath og haspath. inpath sjekker om det gitte ordet finnes i den spesifiserte stien og haspath sjekker om den gitte XPath er til stede i dokumentet.

```
select *
from Kunde k
where contains(k.kundeinfo, 'haspath(/Person[./@CUSTNAME="Knut
Olsen"])') > 0;
```

Hvis dokumentet har en intern struktur, noe som XML har, kan man definere dokumentdeler ved å bruke bestemte tagger før man indekserer. Dokumentdelene kan være område, felt, attributt eller spesielle deler som paragrafer. Dermed kan man spørre i disse dokumentdelene ved å bruke within-operatoren.

9.5 Datatyper for tidsstempel

Tidsdatatypene[13] Oracle støtter er date, timestamp, timestamp with time zone og timestamp with local time zone. date lagrer dato og tid. Mer spesifikt år, måned, dag, time, minutt og sekund. Man kan spesifisere en date-verdi som en litteral, eller konvertere bokstaver eller tall til en date-verdi. Dette gjøres ved funksjonen to_date().

Oracles standard på dato er YY-MON-DD:HH24:MI. Et eksempel på en dato på dette formatet er 02-JUN-11:14:20

Funksjonen sysdate() returnerer den aktuelle system-dato og -tid, altså dato og tid på tidspunktet funksjonen sysdate() kjøres.

timestamp-datatypen er en utvidelse av date-datatypen. Den lagrer år, måned og dag fra date-datatypen, i tillegg til time, minutt og sekund. Man kan velge hvor mange desimaler man vil ha med når det gjelder sekunder. Dette gjøres ved å spesifisere timestamp-datatypen som følger: timestamp [(sekund_presisjon)]. Sekund-presisjonen kan være mellom 0 og 9. 6 er standard.

9.6 Aritmetikk

Oracle gjør det mulig å bruke aritmetiske operasjoner både på date og timestamp. Man kan legge til og subtrahere verdier på en dato. Denne verdien kan være en konstant eller en dato. Oracle tolker antallet i en konstant som antall dager. sysdate+1 er altså i morgen. Man kan ikke multiplisere og dividere date-verdier.

Oracle tilbyr også funksjoner for mange vanlige dato-operasjoner. Et eksempel på en slik funksjon er months_between. Den returnerer antall måneder mellom to datoer. En liste over andre date-funksjoner kan sees på [13].

9.7 XML SQL Utility (XSU)

XML har blitt formatet som ofte brukes til datautveksling. Samtidig er det meste av data lagret i objektrelasjons-databaser. Dette gjør at man trenger muligheter for å transformerer relasjonsdata til XML og motsatt. For å løse dette har Oracle laget XML SQL Utility (XSU).

XSU sin funksjonalitet består av å transformere relasjonsdata til XML, og XML-data til relasjonstabeller. Man kan også bruke XML-dokumenter til å oppdatere og slette verdier i relasjonstabellene. Transformasjonene kan brukes både ved relasjonstabeller og ved såkalte view. For å bruke funksjonaliteten til XSU kan man benytte Java API, PL/SQL API eller Oracle sin egen java kommandolinje.

9.7.1 SQL-til-XML mapping

En SQL-til-XML mapping brukes hvis man vil transformere relasjonsdata til XML-dokumenter. Standarden er at det genereres et XML-dokument av resultatet til en SQL-spørring. Et eksempel på formen til et slikt XML-dokument vil være som følger:

```
<?xml version="1.0"?>
<rowset>
  <row num="1">
    <id>12345</id>
    <navn>Lars</navn>
    <telefon>56565656</telefon>
  </row>
  <!-- flere rader -->
</rowset>
```

XSU kan også benytte en SQL-til-XML mapping mot et objektrelasjons skjema. Da kan man ha komplekse type kolonner. Det vil si at man lager egne typer. For å illustrere dette kan man lage en kompleks type med adresser, som videre benyttes i tabellen Kunde:

```
create type AdresseType as object
(
  gate        varchar2(20),
  postnr     varchar2(20),
  poststed number(4)
);

create table Kunde
(
  personnr  number(11),
  navn         varchar2(20),
  adresse    AresseType
);
```

Når man nå kjører spørringen select * from Kunde, vil resultatet også ha med hele adresse-objektet.

Ofte ønsker man at XML-dokumentet skal ha en spesiell struktur. Da er det ikke alltid at den ønskede strukturen samstemmer med den genererte strukturen som XSU gir i resultatet. For å gi mer kontroll til brukerne, har Oracle laget ulike metoder for å kunne modifisere strukturen til resultatet. Disse metodene er

Kilde tilpasning. Ved å endre databaseskjemaet eller spørringen. I databaseskjemaet kan man lage et objektrelasjonsview som mapper den ønskede strukturen. I spørringen har man ulike operasjoner. Disse går ut på å benytte cursor delspørringer eller cast-multiset konstruksjoner for å få nøsting, alias kolonne/attributt navn eller bruke identifikatorer for å mappe til XML-attributter i steden for til XML-elementer.

Mappe tilpasning. Modifiserer mappingen som brukes i transformasjonen. Dette kan være å endre eller fjerne rowset-tagg, row-tagg eller num-attributtet. Man kan også spesifisere det genererte XML-elementnavnet, formatere datoer eller bestemme hvordan nullverdier skal vises.

Tilpasning etter generering. Hvis de to tilpasningene over ikke kan brukes, kan det benyttes en XSL-transformasjon.

9.7.2 XML-til-SQL mapping

XSU kan som sagt også transformere et XML-dokument inn til en relasjonstabell eller et view. Det er flere operasjoner som kommer under her. Man kan sette inn XML-data i relasjonstabeller, og man kan oppdatere eller slette rader fra relasjonstabellen, som spesifisert i XML-dokumentet.

Ved default XML-til-SQL mapping, mappes et XML-dokument bare inn i èn relasjonstabell. Vil man mappe XML-dokumentet inn til flere tabeller, må man lage et objektrelasjonsview over skjemaet.

Også med denne type mapping kan resultatet modifiseres. Dette kan gjøres på tre måter:

Modifisere målet. Her lager man et objektrelasjonsview over målskjemaet og lar dette view være det nye målet.

Modifisere XML-dokumentet. Her må man benytte XSLT for å transformere XML-dokumentet. Dette bør helst ikke være en permanent løsning.

Modifisere XML-til-SQL mappingen. Man kan la XSU utføre case insensitiv tilpassning av XML-elementene til database-kolonnene eller -attributtene.

For å sette inn innholdet av et XML-dokument inn i en bestemt tabell, henter først XSU ut metadata om måltabellen. Basert på disse metadataene generere XSU et SQL insert-uttrykk. Dette utrykket vil se ut som følger:

```
insert into Kunde (personnr, navn, gate, postnr, poststed) values (?, ?, ?, ?, ?)
```

Neste steg for XSU nå, er å parse XML-dokumentet. For hver post i XML-dokumentet bindes den rette verdien til den rette kolonnen. Til slutt eksekveres uttrykket.

Oppdatering og sletting er litt annerledes fra innsetting, fordi de kan påvirke mer enn en rad i databasen om gangen. XML-elementet kan passe til flere rader hvis den passennde kolonnen ikke er en nøkkelkolonne i tabellen. Ved oppdatering trenger XSU en liste over kolonner som skal brukes i identifiseringen av radene som skal oppdateres. For å vise dette kan vi endre telefonnummeret til Lars fra eksemplet over. Vi vil endre telefonnr fra 56565656 til 12121212, og bruker kolonnen id til identifiseringen. For å gjøre dette brukes følgende XML-dokument:

```
<rowset>
  <row num=”1”>
    <personnr>12345</personnr>
    <telefon>12121212</telefon>
  </row>
</rowset>
```

Dette vil gi update-utrrykket

```
update Person SET telefon = ? where personnr = ?
```

Videre er det likt som ved innsetting, verdiene bindes til rette kolonner og uttrykket eksekveres. Ved oppdatering kan man også spesifisere at ikke alle elementene som er i XML-dokumentet skal oppdateres.

For sletting vil det være likt som ved oppdatering. XSU får en liste over kolonnene som skal brukes til å identifisere de rette radene. Dermed lages et delete-uttrykk, og XSU binder verdier fra XML-dokumentet til dette uttrykket.

En viktig ting å vite er at attributter i XML-dokumentet ikke mappes til relasjonstabellene, men ignoreres. Vil man allikevel lagre attributtene i relasjonstabellen, må man først benytte en XSL-transformasjon for å gjøre attributtene i XML-dokumentet om til elementer.

9.8 Oracle9i database Release 2

XML DB[54] tilbyr en mulighet for å lagre og håndtere både strukturerte og ustrukturerte data. Man kan utføre XML-operasjoner på tabelldata og SQL-operasjoner på XML-dokumenter. XML DBs nøkkel-teknologi er XMLType og XML Repository. XMLType ble introdusert i Oracle8i, men det har skjedd en del forandringer siden da. Disse vil vi komme nærmere inn på. Vi skal også ta for oss repository og indeksering.

XML DB holder struktur-informasjon (som element-tagger, datatyper og lagringslokasjon) i et skjema-cache. Dette gjøres for å minimere tilgangstiden og lagrings kostnader. Dette vil også høyne utførelse og skalerbarheten til store dokumenter, samt store antall med dokumenter.

9.8.1 XMLType

XMLType-datatypen lagrer XML-innhold og kan brukes som datatype til en kolonne. XMLType har flere nyttige metoder for å håndtere XML-innholdet. Den kan lagres på to forskjellige måter, som LOB og som objektrelasjon. Den første metoden opprettholder nøyaktigheten til det originale XML-dokumentet. Den andre metoden som lagrer XML-innholdet som objektrelasjon, opprettholder DOM-trofasthet, som rekkefølge til elementer og attributter, gjør forskjell på elementer og attributter, støtte for XML-datatyper som ikke er tilgjengelig i SQL (for eksempel boolean), o.l. XMLType instanser innholder da skjulte kolonner som lagrer denne ekstra informasjonen. Denne informasjonen kan man få tilgang til via APIer i SQL eller Java, slik som ExtractNode-funksjonen.

9.8.2 Repository

XML DB tilbyr et internet lagerssted for håndtering av XML-data og –dokumenter. Hovedfordelene med lageret er muligheten til å organisere innhold og annotere det slik at det raskt kan hentes ut. En annen fordel er håndtering av komplekse forhold og avhengigheter som deler av innhold kan ha.

9.8.3 Indeksering

De to forskjellige metodene man kan lagre XML-dokumenter på har hver sin måte å indeksere på. LOB bruker Text indeks og objektrelasjon bruker B-tre indeks.

XML DB tilbyr et spesielt hierarkisk indeks for XML-dokumenter lagret i Repository. Dette er for å øke hastigheten på oppløsningen til stinavnet og folder-søk. Man kan

også mappe hierarkiske data automatisk i relasjonstabeller til foldere, hvor hierarkiet er definert av eksisterende relasjonsinformasjon.

DEL III

Vår løsning

Del III tar for seg vår løsning på hvordan man kan gjøre et databasesystem temporalt samtidig som den håndterer XML. Først vil funksjonaliteten til programmet presenteres, før vi går over på arkitekturen og drøfting av valg vi har tatt. I kapittel 12 vil forskjellige løsninger testes og evalueres. Deretter beskrives implementasjonen av den beste løsningen. Til slutt kommer en evaluering av programmet.

Kap 10: Funksjonalitet

Kap 11: Design

Kap 12: Lagringsalternativer

Kap 13: Implementasjon

Kap 14: Evaluering av programmet

10.0 Funksjonalitet

Programmeringsdelen av oppgaven vår går ut på å legge til temporale funksjonaliteter til et XML-enabled databasesystem. Ved hjelp av disse funksjonalitetene skal en brukeren kunne versjonere sine XML-dokumenter. Brukere skal kunne lagre så mange versjoner av et XML-dokument han vil, og senere kunne kjøre spørringer mot både de nye og de gamle versjonene.

For å løse denne oppgaven må vi lage en temporal utvidelse av spørrespråket som benyttes. Vi skal, som skrevet tidligere, bruke databasesystemet Oracle9i. Dermed er det spørrespråket SQL vi skal utvide. Vår temporale utvidelse har vi kalt TXSQL (Temporal XML SQL). Et viktig mål med oppgaven vår er at TXSQL skal være bakover kompatibel med SQL, slik at brukeren kan skrive vanlige SQL-uttrykk. Det vil også være fult mulig å bruke programmet vårt selv om brukeren ikke ønsker temporalitet i sine tabeller. Mer om TXSQL vil komme i kapittel 10.3.

For å implementere TXSQL må vi lage en preprosessor. Denne preprosessoren skal få inn et temporale uttrykk av brukeren, parse og evaluere dette, og gjøre det om til nye uttrykk som skal sendes til databasen. Resultatet fra databasen skal deretter evalueres og vises for brukeren.

Det vil være to typer brukere av preprosessoren. Den ene vil være en bruker som skriver uttrykk direkte inn i et grensesnittvindu. Den andre muligheten er et program som kommuniserer med vårt program. Vårt program vil da være et API[1] for det andre programmet. Det er det siste alternativet som bli brukt mest i praksis. Vi skal først ta for oss kommunikasjonen via et grensesnitt og deretter via API.

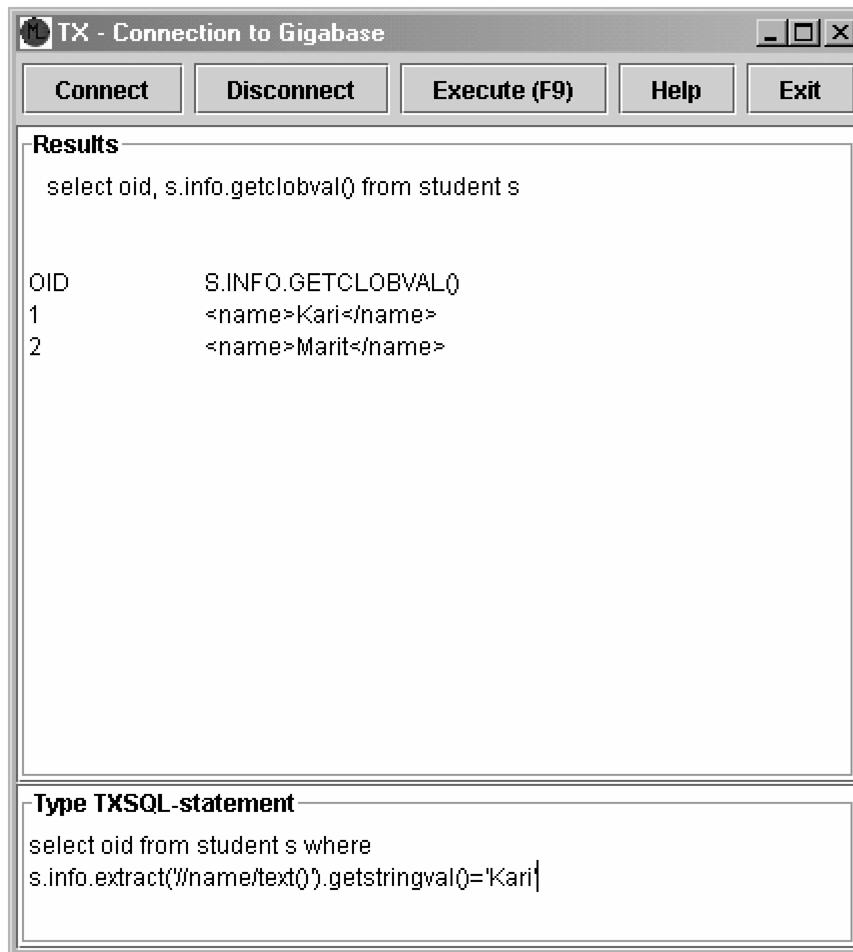
10.1 Grensesnittet

For brukere som vil skrive inn sine uttrykk selv, har vi laget et eget grensesnitt som heter TX (Temporal XML). For å lage grensesnittet har vi tatt utgangspunkt i SQL*Plus, som er brukergrensesnittet som Oracle9i bruker. Vi mener brukeren enkelt må kunne se både hvilket uttrykk han selv har skrevet inn og resultatet av dette uttrykket. Begge disse egenskapene har SQL*Plus. En av ulempene med SQL*Plus er at man ikke kan redigere uttrykket som akkurat er skrevet. Skal man endre på noe i starten av uttrykket må hele linjen slettes til dit man vil redigere. Dette kan være irriterende for brukere som skriver inn uttrykkene selv. En annen svakhet ved SQL*Plus er når uttrykket går over flere linjer i grensesnittet. Da er det ikke mulig å endre på linjer lenger oppe i uttrykket. Dermed må man avbryte og skrive hele uttrykket på nytt. Dette er noe vi velger å endre på.

Grensesnittet vil få to tekstområder og en knapperad, se figur 9. Det minste tekstområde, uttrykksområde, er for å skrive inn de uttrykkene brukeren vil kjøre. Det største tekstområde, resultatområde, er for resultatet av brukerens uttrykk. Knapperaden vil bestå av knapper for opp- og nedkobling til databasen, eksekvering av uttrykket i uttrykksområde, avslutning og en snarvei til hjelpesider. For å utføre et uttrykk som står i uttrykksområde, kan man enten trykke på “Execute“-knappen eller

1. API, Application Programming Interface, er et grensesnitt mellom et applikasjonsprogram og operativ systemer. Brukes for å kommunisere med andre applikasjoner.

på F9-tasten på tastaturet. Da vil uttrykket bli evaluert og sendt til databasesystemet. Resultatet kommer så opp i resultatområdet.



FIGUR 9. TX grensesnittet

Resultatområdet vil vise alle uttrykk med tilhørende resultater. Selve uttrykket vil vi vise med et innrykk foran, for å skille det ut fra resultatet. For hver av de seks uttrykkstypene brukeren kan skrive inn, vil vi ha egne resultater som skal skrives ut. For eksempel ved update-uttrykket vil det stå antall rader som er oppdatert i databasesystemet. For select-uttrykk vil vi vise resultatet som en tabell, hvor kolonnenavn skal stå i headeren.

Videre vil brukeren kunne kopiere uttrykk som allerede er skrevet. Dermed kan disse limes inn i tekstboksen for uttrykk og enkelt redigeres på.

Vi vil også lage hjelpesider, hvor brukeren kan få informasjon om hvordan TX fungerer og grammatikken til TXSQL. På disse hjelpsidene vil man få en kort beskrivelse av selve grensesnittet. Videre vil det stå forklaring til de temporale utvidelsene vi har på spørrespråket. Disse forklaringene vil vi illustrere med eksempler, slik at det vil være enkelt for brukeren å skjønne hvordan språket er utvidet.

10.2 API

Preprosessoren skal også kunne brukes fra et annet program. Da vil preprosessoren oppføre seg som en ferdigdefinert pakke. Det vil da kunne opprettes en instans av preprosessoren. Denne instansen vil kunne ta inn et uttrykk som parameter og evaluerer dette uttrykket. Resultatet kan hentes opp ved å kalle på en bestemt funksjon til denne instansen. Dermed vil det være enkelt å bruke preprosessoren fra andre programmer.

10.3 Spørrespråket

Spørrespråket vårt, TXSQL, er en utvidelse av spørrespråket SQL som databasesystemet Oracle9i bruker. Denne utvidelsen vil gi brukeren en effektiv måte å spørre på i de versjonerte XML-dokumentene. Det vil både gå an å spørre etter eldre versjoner og på tidsstemplet til de ulike versjonene. Vi vil her komme inn på hvilke utvidelser det er snakk om, og hvilken hensikt de har. Under beskrivelsene vil det bli vist eksempler.

- t_time: kan brukes i create-delen. Det brukes for å deklarere at en tabell er temporal. Det vil da registreres et tidsstempel for hver XML-versjon som legges i denne tabellen, og brukeren kan legge inn flere versjoner. Her vil vi gjøre tabellen usertable temporal:

```
create table usertable t_time (
   oid    number(8),
    xml  sys.xmltype )


Resultat: Table created
```

- t_start: kan brukes i select-delen for de tabeller som er temporale. Ved bruk av dette ordet i spørringen, vil man få ut tiden når versjonen ble lagret i databasesystemet. Velger man alle rader i en tabell ved hjelp av “*“, vil man ikke få ut tidsstemplet. Tidsstemplet kommer bare opp hvis man spør etter det. For å illustrere dette kan vi vise to eksempler. Det første eksemplet spør etter oid og tiden XML-versjonen ble lagt inn i databasesystemet, for tabellen usertable. Det andre eksemplet viser hva som kommer som resultatet ved å spør etter alt (dvs bruk av *) i tabellen usertable:

```
select oid, t_start                  select *
from usertable                         from usertable


Resultat:
oid    t_start                         oid    xml
---------------------------------      ------------------------
1      20.05.02 12:15:00                 1      xmltype()
2      23.05.02 08:05:00                 2      xmltype()
```

Noe annet som også vises i det siste eksemplet over er at det skrives ut xmltype() på XML-kolonnen. Grunnen til dette er at man spør etter selve kolonnenavnet. For å få

ut hele XML-versjonen må man skrive xml.getclobval().

- t_end: kan brukes i select-delen for de tabeller som er temporale. Det vil være tilsvarende som for t_start, men her vil man ha ut sluttiden til XML-versjonen og ikke starttiden. Sluttiden her vil være når XML-versjonen ble oppdatert av en ny versjon eller slettet.

version(all | first | last | history | date | period): kan benyttes i select-uttrykkets where-del. version brukes for å spesifisere hvilke versjoner man vil spørre i, se figur 10. Figuren viser hvilke versjoner som blir returnert etter en spørring ved bruk av de ulike version-parameterne. Hvis man ikke bruker version, vil man automatisk spørre i den siste versjonen. Parametrene til version sier hvilke versjoner det er snakk om og kan være som følger:

all: vil spørre etter alle versjoner av et dokument.

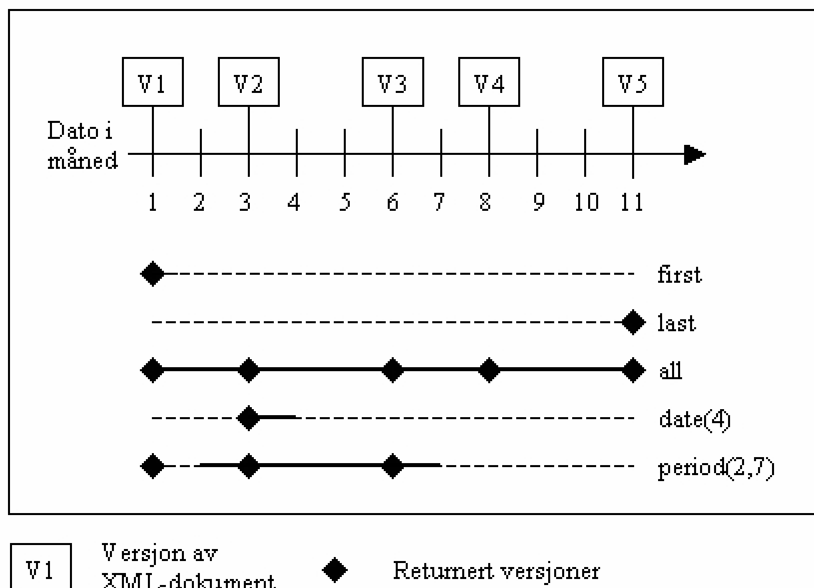
first: vil spørre etter den første versjonen av XML-dokumentet.

last: vil spørre etter den siste versjon av XML-dokumentet, som stemmer med resten av where-delen.

history: brukes for å vise at man vil spørre i alle versjoner av XML-dokumentene, uten å måtte spesifisere hvilke versjoner som skal returneres

date(...): vil ha en dato som parameter, og den versjonen som er gjeldende på denne datoen skal returneres.

period(..., ...): vil ha to datoer som parametere, som vil utgjøre en periode. Alle versjoner som er gjeldende innenfor denne perioden, vil returneres.



FIGUR 10. Viser versjoner som blir returnert ved bruk av ulike version-parametre

For å illustrere bruken av version, kan vi hente ut alle versjoner av XML-dokumentet hvor oid er lik 123:

```
select x.xml.getClobVal()
from usertable
where oid = 123 and version(all)

Resultat:
 x.xml.getClobVal()
-----------------------------------------------------------
<name>Per</name><tel>73882934</tel>
<name>Per</name><tel>56894534</tel>
```

- vacuum(date | period): etter hvert som det blir mange dokumenter og mange versjoner av dokumentene, kan det bli aktuelt å fysisk slette noen av dem. Da kan brukeren benytte vacuum. Parameteren til dette ordet vil bestemme hvilke versjoner som skal bli slettet etter i hvilke tidsrom de ble lagt inn;

  date(...): her kommer det inn en dato. Alle versjoner som ble lagt inn før denne datoen skal slettes. Dette vil bare gjelde for historiske versjoner. Finnes det bare en versjon av et dokument og denne ble lagt inn før denne datoen, vil ikke versjonen bli slettet.

  periode(..., ...): her kommer det inn to datoer som lager en tidsperiode. Alle historiske versjoner som ligger innenfor denne perioden vil bli slettet.

  For å illustrere dette kan vi fysisk slette alle versjoner som ble lagt inn før datoen 01.01.00:

  ```
  alter table usertable vacuum ( date ( 01.01.00 ) )
  ```

- granularity: dette ordet sier noe om hvor tett i tid versjoner kan legges inn. Brukeren kan selv sette denne, og den vil være bra med tanke på opprydning i versjoner. Eksempel på når denne kan være aktuell er hvis det er snakk om at noen holder på å oppdatere en versjon over litt tid. Hvis da versjonen legges midlertidig inn flere ganger i løpet av perioden, vil det bare være den siste versjonen som egentlig er aktuell. Hvis da granularity er satt til for eksempel 1 uke, vil versjoner bli automatisk slettet hvis det legges inn en nye versjon innenfor samme uke. Det vil da bli den nyeste versjonen som vil være lagret i databasesystemet. De andre midlertidige versjonene vil bli slettet.

  Hvis vi har at granulariteten måles i dager, kan vi for eksempel sette at innen periode på 5 dager, skal det bare være en versjon av samme XML-dokument:

  ```
  set granularity day ( 5 )
  ```

  Tilsvarende kan man gjøre med time (time()) og måned (month()).

- granularity_vacuum(date, number): dette ordet vil være ganske lik som vacuum. Her vil det i steden for komme inn to parametre. Den første vil være en dato, mens den andre parameteren vil være et tall som indikere en granularitet. Granularity_vaccum sier egentlig at før den gitte datoen skal det minst være en tidsperiode på over det granularitet viser. Hvis det er en mindre periode mellom to versjoner, vil den eldste av de to versjonene fysisk slettes. For å illustrere dette kan vi si at alle versjoner som ble lagt inn i databasesystemet før 01.01.98, må minst ha en tidsperiode mellom seg på 5 dager:

```
granularity_vaccum ( date ( 01.01.98 ) , 5 )
```

I appendiks C finnes det flere eksempler på spørrespråket TXSQL.

For å få med våre egne utvidelser av spørrespråket i vår implementasjon, velger vi å lage vår egen SQL-parser. I denne parseren kommer vi ikke til å dekke alle deler av SQL, men heller ha fokus på de delene som trengs for å bruke de temporale aspektene vi legger til. Dette gjelder også de deler som brukes ved spørringer i XML-versjoner.

Selv om vårt program lages for hovedsakelig å kunne gjøre spørringer i XML-dokumenter, vil det også være mulig å spørre i vanlige relasjonstabeller. Dette gjelder også å legge til temporale aspekter på vanlige relasjonstabeller.

Videre vil TX være transaksjonsbasert. Det vil si at man kan kjøre flere uttrykk i en transaksjon. Brukeren må selv kjøre commit eller rollback når transaksjonen er ferdig. Dette vil være tilsvarende som for SQL*Plus.

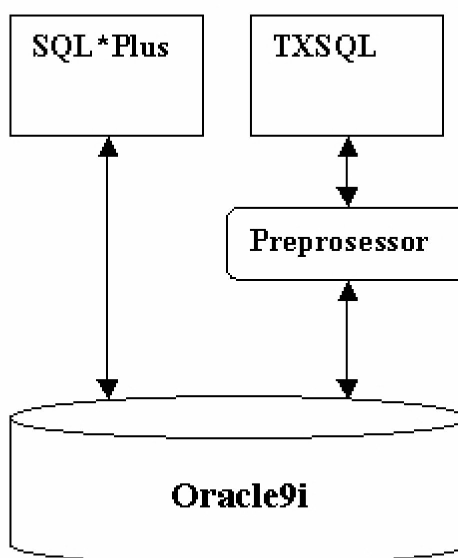
Det vil ikke være alt som er implementert av det som er beskrevet i dette kapitlet. Hva som faktisk er implementert vil være beskrevet i kapittel 13.0.

11.0 Design

I dette kapitlet skal vi ta for oss programmets arkitektur. Vi skal også se på ulike alternativer vi har valgt å bruke, eventuelt forkastet dem og grunnen til dette. Vi har valgt å benytte et Oracle9i databasesystem og som beskrevet i kapittel 10.0, blir det da spørrespråket SQL vi skal utvide.

11.1 Arkitektur

Den temporale preprosessoren skal implementeres på toppen av en eksisterende DBMS. Selve DBMS vil ikke bli modifisert, og vi kan tenke oss at den er en black box. Figur 11 viser et allerede eksisterende grensesnitt man kan bruke for å kommunisere med Oracle, SQL* Plus. Videre vises hvordan vi tenker oss kommunikasjonen mellom vårt grensesnitt og DBMS. Vi ser at det er en preprosessor mellom grensesnittet vårt, TX, og DBMS. Denne preprosessoren skriver om de temporale uttrykkene som kommer fra TXSQL om til SQL. SQL-uttrykkene blir så utført i DBMS og resultatene blir sendt tilbake til preprosessoren, som utfører noen postprosesseringer og sender resultatet til TX.



FIGUR 11. Arkitektur over TXSQL

Denne arkitekturformen kalles lag-arkitektur[15,16]. Med denne lag-arkitektur kan man bruke ”tjenester” fra DBMS, slik som concurrency control, recovery mekanismer, access control, query optimering, indeksering, lagring, osv. Ulempen med lag-arkitektur er at det påløper en ekstra kostnad for klienten når man vil ha tilgang til databasesystemet, og det at det ikke er mulig å manipulere DBMS-interne datastrukturer direkte.

11.2 Lagring

For å få en mest mulig effektiv preprosessor, er det veldig viktig hvordan XML-dokumentene blir lagret. Dette gjelder ting som hvordan dokumentene skal bli lagret i databasesystemet, og hvordan man skal lagre tidsstempler og versjonsnummer på de ulike XML-dokumentene. Her vil vi komme inn på alternative løsninger på disse utfordringene.

11.2.1 Lagring av XML-dokumentene

Når det gjelder hvordan selve XML-dokumentene skal lagres i databasesystemet, har vi som beskrevet i kapittel 9.0 to forskjellige måter å lagre XML-dokumentene i Oracle databasesystemet. Disse er bruk av XMLType (CLOB) eller ved å ekstrahere dokumentene ut i kolonner i tabeller. Vi har valgt at det bare er mulig å lagre dokumentene i XMLType. En av grunnene til dette er at det vil ta for lang tid å implementere begge løsningene. I tillegg støtter ikke Oracle9i en komplett round trip (som beskrevet i kapittel 3.2.2) av forskjellig XML-innhold inn og ut av databasesystemet. Det er heller ingen andre XML-enabled databasesystemer som støtter fullstendig round trip av XML-dokumenter[46].

11.2.2 Legge til tid på elementer

Vi har valgt en løsning der vi legger tidsstempel på hele XML-dokumentet. Tidsstemplet vil da bli lagret i en egen tabell med en identifikator til XML-dokumentet. Vi har valgt denne løsningen for å få tabellene normaliserte. Hvis brukeren lager en tabell med navn Usertable, med kolonnen xml, så vil denne tabellen bli gjort om til følgende tabeller:

```
Usertable { oid, xml }
Time_usertable { doc_id, oid, start_time, end_time }
```

Et annet alternativ ville vært å legge til tidsstempel på enkle elementer. En fordel ved dette er at elementene kan benyttes av flere dokumenter, slik vi har beskrevet i kapittel 4.2. For vår løsning vil dette være for omfattende, og vi velger derfor å ikke ta hensyn til dette.

En annen måte å lagre selve tidsstemplet til XML-dokumentet på, er å manipulere på selve XML-dokumentet. Da kunne man lagt til tidsstemplet i et attributt eller et element. Ulempen med denne løsningen er at man må gå gjennom hele XML-dokumentet for å hente ut tidsstemplet, og det vil ta lenger tid.

11.2.3 Hvordan skille versjonene

Vi må også tenke på hvordan programmet vårt skal vite hvilke versjoner som hører til hvilke dokumenter. Det er flere måter å løse dette på, og vi skal nå se på to av dem. Disse er:

1. Lage et versjonsattributt eller et element i XML-dokumentet
2. Legg til kolonner i tabellen

I det første alternativet må selve XML-dokumentene manipuleres. Ulempen med dette er at programmet vil bli en god del langsommere.

Den andre måte man kan løse problemet på er å legge til ekstra kolonner til tabellene som brukeren lager. Man må da både registrere hvilket dokument versjonen av dokumentet tilhører og en identifikator til selve dokumentet. Dette må gjøres for å vite hvilken versjon av dokumentet som skal velges. Vi mener dette er den beste løsningen og har valgt å benytte den i vår implementasjon.

11.3 Evaluering av uttrykkene

Målet med preprosessoren er å gjøre brukerens tabeller om til temporale tabeller, altså til lagring av alle versjonene av XML-dokumenter brukeren vil ha. Det er flere alternativer for hvordan disse tabellen kan bli modellert. For å velge den beste løsningen, kjørte vi noen tester på de tre beste alternativene. I kapittel 12.0 vil vi komme med en beskrivelse av alternativene og komme med resultatene av testene.

Vi valgte den løsningen som gav oss det beste resultatet etter gitte kriterier. Dermed blir det altså laget 4 tabeller for hver tabell brukeren lager. To av dem lagrer XML-dokumentene, en for de nyeste versjonene og en for historiske versjoner. Hver av disse to tabellene har en annen tabell som lagrer tidsstemplet til XML-dokumentet. Dette gjør at programmet vårt må evaluere de seks ulike uttrykkene som brukeren kan skrive inn, før de sendes til databasesystemet. Usertable vil være den tabellen brukeren lager og evalueringen vil bestå av:

- create: her lages det 4 tabeller av den ene tabellen som brukeren lager. Disse vil hete usertable, time_usertable, usertable_old og time_usertable_old. Disse tabellen vil se ut som på figur 12. Her tar vi utgangspunkt i at brukeren lager en tabell med en kolonne for XML-dokumenter. Dermed vil usertable og usertable_old få denne kolonne og i tillegg en kolonne for objekt identifikator (oid). Time_usertable vil få den samme oid, en dokument identifikator (docId) og en starttid. Start tid vil være når XML-dokumentet ble lagt inn i databasen. Time_usertable_old vil være helt lik, men med en kolonne i tillegg. Dette vil være Sluttid som er når versjonen ikke er den gyldige versjonen lenger.
- insert: her vil det i tillegg til raden som brukerne skal sette inn i usertable, settes inn et tidsstempel for raden i tabellen time_usertable.
- update: for dette uttrykket må det gjøres en del tilleggsoperasjoner. For det første må raden(e) som skal oppdateres hentes ut fra usertable og time_usertable og settes inn i henholdsvis usertable_old og time_usertable_old. Deretter må de samme radene oppdateres i usertable og time_usertable.
- delete: her gjelder mye det samme som for update; hente ut de aktuelle radene og sette disse inn i de historiske tabellene. Deretter slettes radene fra usertable og time_usertable.
- select: her må uttrykket evalueres grundig. For det første må det sjekkes om det spørres etter noe tidsstempel, om så må det også gjøres en spørring i tidstabellene. Videre må det sjekkes om det gjøres en spørring i de historiske tabellene, altså om ordet version er med i uttrykket. Det vil komme en mer omfattende beskrivelse av hva som skjer i evalueringen av version i kapittel 11.3.1. Når alt er evaluert, lages det et eller flere uttrykk av resultatet. Disse sendes så til databasen.
- drop: hvis brukeren vil slette en av sine tabeller, må alt som ligger i usertable og time_usertable flyttes over til de historiske tabellene. Deretter slettes både usertable og time_usertable.

For hvert av uttrykkene, med unntak av create, må det først sjekkes om tabellen brukeren vil gjøre noe med, er temporal. Dette sjekkes opp mot en annen tabell vi har laget, som inneholder navnene til alle tabellene som er temporale.

Usertable

Oid	XML
3	<name>Mons</name>
4	<name>Kari</name> <tel>23 23 23 23</tel>
5	<name>Mari</name> <alder>26</alder>

Usertable Old

Oid	XML
1	<name>Kari</name> <tel>21 21 21 21</tel>
2	<name>Mari</name> <alder>24</alder>

Time Usertable

DocId	Oid	Start tid
3	3	20.04.02 12:45:00
1	4	24.04.02 08:05:00
2	5	03.05.02 09:00:00

Time Usertable Old

DocId	Oid	Start tid	Slutt tid
1	1	02.03.02 12:00:00	24.04.02 08:05:00
2	2	20.03.02 07:15:00	03.05.02 09:00:00

FIGUR 12. Oversikt over tabellen som lages av TX når brukeren lager Usertable

For å lagre tidsstemplene til XML-dokumentene har vi valgt å bruke date-variabelen til Oracle9i. Denne variabelen lagrer både dato og tidspunkt, som er interessant for oss.

Nå vil vi komme med en mer omfattende beskrivelse av evalueringen av version-ordet.

11.3.1 Algoritmer til version

De ulike parametrene som kan komme med det temporale ordet `version`, vil ha ulike algoritmer. Her vil vi gi en kort beskrivelse av hver av de ulike parametrene version kan ha. I beskrivelsene vil vi ta utgangspunkt i at det bare er *et* dokument som kommer ut som resultat etter spørringen. Dette er for å lette beskrivelsen litt, men trenger ikke være tilfelle.

Version vil som sagt kunne benyttes i select-uttrykkets where-del. Et eksempel kan være

```
select *
from Ansatt a
where version(first) and a.info.extract('//navn/text()').getstringval()='Kari'
```

Denne spørringen returnerer det første versjonen av dokumentet i tabell Ansatt som inneholder et element navn med verdien Kari. Det kan ikke bare stå version i where-delen, men må spesifiseres mer hvilke dokumenter man vil ha versjoner fra. Where-delen, foruten om delen med version, vil i de neste delene bli kalt for resten av where-delen.

version(first): Her vil man finne den første versjonen av et dokument. For å gjøre dette må først alle dokumenter hentes, både fra gjeldende tabell og historisk tabell, hvor resten av where-delen stemmer. Deretter ser man på hvilken versjon som har den laveste t_start. Det må da spørres i time_usertable og time_usertable_old. Versjonen av dokumentet som har lavest t_start returneres.

Pseudokode:

```
version_first (where_stmt, usertable_name )
  tabell med doc_id, start_time, oid
  hvis usertable_name i temporaltable
    så spør etter doc_id, start_time, oid fra usertable_name med where_stmt
      til database

    for i fra 0 til antall resultatrader
      hvis doc_id ikke finnes i tabell
        så legg til doc_id i tabell
        legg inn start_time i tabell
        legg inn oid i tabell
      stopp hvis
    stopp for

    så spør etter doc_id, start_time, oid fra usertable_name_old med
      where_stmt til database

    for j fra 0 til antall resultatrader
      hvis doc_id ikke finnes i tabell
        så legg til doc_id i tabell
        så legg inn start_time i tabell
        legg inn oid i tabell
      eller hvis doc_id finnes i tabell
        så hvis start_time før start_time fra tabell hvor doc_id stemmer
          så overskriv i tabell med den nye start_time på rett doc_id
          overskriv i tabell med den nye oid på rett doc_id
        stopp hvis
      stopp hvis
    stopp for
  stopp hvis

  returner tabell
```

version(last): Ved å bruke last-parameteren sier man at man vil hente ut den siste versjonen av et dokument. Om man gjør en spørring uten å bruke version-ordet, vil den siste versjonen returneres. Det blir altså søkt bare i den gjeldende tabellen. Men det er ikke i alle tilfeller at versjonen av dokumentet er gyldig nå. Det må da søkes i den historiske tabellen også. Vi gjør da som i version(first), henter alle dokumentene hvor resten av where-delen er rett, både i gjeldende og historisk tabell. Av alle versjonene som blir returnert, henter man ut den versjonen som har høyeste t_start.

Pseudokode:

```
Version_last ( where_stmt, usertable_name)
  tabell med doc_id, start_time, oid

  hvis usertable_name i temporaltable
    så spør etter doc_id, start_time, oid fra usertable_name med where_stmt
      til database

    for i fra 0 til antall resultatrader
      hvis doc_id ikke finnes i tabell
        så legg til doc_id i tabell
        legg inn start_time i tabell
        legg inn oid i tabell
      stopp hvis
    stopp for

    så spør etter doc_id, start_time, oid fra usertable_name_old med
       where_stmt til database

    for j fra 0 til antall resultatrader
      hvis doc_id ikke finnes i tabell
        så legg til doc_id i tabell
        så legg inn start_time i tabell
        legg inn oid i tabell
      eller hvis doc_id finnes i tabell
        så hvis start_time etter start_time fra tabell hvor doc_id stemmer
          så overskriv i tabell med den nye start_time på rett doc_id
          overskriv i tabell med den nye oid på rett doc_id
        stopp hvis
      stopp hvis
    stopp for
  stopp hvis

  returner tabell
```

version(all): Denne parameteren skal returnere alle versjoner av et dokument. Dette gjøres ved at det søkes gjennom både gjeldende og historisk tabeller. Først vil versjonene hvor resten av where-delen inntreffer hentes. Dokumentidentifikatoren blir så hentet ut, og absolutt alle versjoner som har samme dokumentidentifikator blir returnert til brukeren. Det betyr altså at alle versjoner av et dokument returneres, selv om det ikke er alle versjoner hvor resten av where-delen er gyldig.

Pseudokode:

```
version_all (usertable_name, where_stmt )
  tabell med doc_id, oid

  hvis usertable_name i temporaltable
    så spør etter doc_id og oid fra usertable_name med where_stmt til
       database

    for i fra 0 til antall resultatrader
      så legg til doc_id i tabell
      legg inn oid i tabell
    stopp for

    spør etter doc_id, oid fra usertable_name_old med where_stmt til
       database

    for j fra 0 til antall resultatrader
      så legg til doc_id i tabell
      legg inn oid i tabell
    stopp for

    spør etter doc_id, oid fra time_usertable_name med alle doc_id i tabell til
       database
    spør etter doc_id, oid fra time_usertable_name_old med alle doc_id i
       tabell til database
    resultatsett lik begge resultatsettene fra spørringenen

    for k fra 0 til antall resultatrader
      for m fra 0 til alle tabellrader hvor tabellens doc_id er lik doc_id
        så hvis oid finnes i noen av tabell radene
          så fjern rad fra spørringens resultatsett
        stopp hvis
      stopp for
    stopp for

    for n fra 0 til antall resultatrader
      så legg til doc_id i tabell
      legg inn oid i tabell
    stopp for
  stopp hvis

  returner tabell
```

version(date): Ved date-parameteren skal man hente ut versjonen av et dokument som er gyldig på datoen som kom med parameteren. Først hentes alle versjoner ut som stemmer overens med resten av where-delen. Deretter velges den versjonen hvor versjonens t_start er lavere, men nærmest, eller lik datoen som brukeren sender inn.

Pseudokode:

```
version_date( dato ) (where_stmt, usertable_name)
  tabell med doc_id, start_time, oid

  hvis usertable_name i temporaltable
    så spør etter doc_id, start_time, oid fra usertable_name med where_stmt
      til database

    for i fra 0 til antall resultatrader
      hvis doc_id ikke finnes i tabell
        så legg til doc_id i tabell
        hvis start_time er før dato
          så legg inn start_time i tabell
          legg inn oid i tabell
        stopp hvis
      eller hvis doc_id finnes i tabell
        så hvis start_time før dato
          så hvis finnes start_time fra før i tabell med rett doc_id
            så hvis start_time etter start_time fra tabell hvor doc_id stemmer
              så overskriv i tabell med den nye start_time på rett doc_id
              overskriv i tabell med den nye oid på rett doc_id
            stopp hvis
          stopp hvis
          eller hvis ikke finnes start_time fra før i tabell med rett doc_id
            så legg inn start_time i tabell med rett doc_id
            legg inn oid i tabell med rett doc_id
          stopp hvis
        stopp hvis
      stopp hvis
    stopp for

    så spør etter doc_id, start_time, end_time, oid fra usertable_name_old
       med where_stmt til database

    for j fra 0 til antall resultatrader
      hvis doc_id ikke finnes i tabell
        så legg til doc_id i tabell
        hvis start_time er før dato
          så legg inn start_time i tabell
          legg inn oid i tabell
        stopp hvis
      eller hvis doc_id finnes i tabell
        så hvis start_time før dato
          så hvis finnes start_time fra før i tabell med rett doc_id
            så hvis start_time etter start_time fra tabell hvor doc_id stemmer
              så overskriv i tabell med den nye start_time på rett doc_id
              overskriv i tabell med den nye oid på rett doc_id
            stopp hvis
          stopp hvis
          eller hvis ikke finnes start_time fra før i tabell med rett doc_id
            så legg inn start_time i tabell med rett doc_id
            legg inn oid i tabell med rett doc_id
          stopp hvis
        stopp hvis
      stopp hvis
    stopp for
  stopp hvis
returner tabell
```

version(period):Når parameteren period benyttes vil algoritmen være en utvidelse av version(date). Nå er det to datoer som kommer som parametre til period, og alle versjoner som er gyldige mellom disse datoene, så sant resten av where-delen stemmer, skal returneres. Dette gjelder også for versjoner som har t_start mindre, men t_end med i perioden, eller versjoner som er gyldige i og etter perioden som brukeren kommer inn med.

Pseudokode:

```
version_period( dato1, dato2 ) (where_stmt, usertable_name)
  tabell med doc_id, oid

  hvis usertable_name i temporaltable
    så spør etter doc_id, start_time, oid fra usertable_name med where_stmt
      til database

    for i fra 0 til antall resultatrader
      hvis start_time er før dato2
        så legg til doc_id i tabell
        legg inn oid i tabell
      stopp hvis
    stopp for
    så spør etter doc_id, start_time, end_time, oid fra usertable_name_old
      med where_stmt til database

    for j fra 0 til antall resultatrader
      hvis start_time er før dato1
        så hvis end_time er etter dato1
          så legg til doc_id i tabell
          legg inn oid i tabell
        stopp hvis
      eller hvis start_time er etter dato1
        så hvis start_time er før dato2
          så legg til doc_id i tabell
          legg inn oid i tabell
        stopp hvis
      stopp hvis
    stopp for
  stopp hvis

  returner tabell
```

version(history): Om denne parameteren benyttes vil det si at det ikke bare skal søkes i tabellen med gjeldende versjoner, men også i den historiske-tabellen. Da returneres alle versjoner hvor resten av where-delen stemmer.

Pseudokode:

```
version_history (usertable_name, where_stmt)
  tabell med doc_id, oid

  hvis usertable_name i temporaltable
    så spør etter doc_id, oid fra usertable_name med where_stmt til database

    for i fra 0 til antall resultatrader
      så legg til doc_id i tabell
      legg inn oid i tabell
    stopp for

    spør etter doc_id, oid fra usertable_name_old med where_stmt til
      database

    for j fra 0 til antall resultatrader
      så legg til doc_id i tabell
      legg inn oid i tabell
    stopp for
  stopp hvis

  returner tabell
```

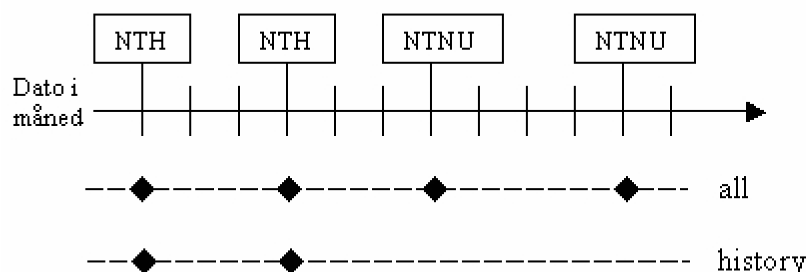
Det hele vil bli noe mer komplisert om det er flere dokumenter som skal returneres som svar. Man må da holde rede på dokumentidentifikatorene og returnere de riktige versjonene for dem alle. Version(first) vil altså returnere de første versjonene til alle dokumentene, hvor resten av where-delen stemmer.

En alternativ måte version kunne fungert på, er at det først ble sjekket i gjeldende versjoner, for å hente ut dokument-identifikatoren. Deretter kan man bruke denne til å søke i historiske versjoner etter siste dokument eller fra en dato alt etter som hvilken parameter som er satt i version. Dette vil gå noe raskere enn slik vi har valgt å gjøre det. Ulempen med denne måten å gjøre det på er at det ikke er alltid at riktig dokument blir returnert. Vi skal nå ta for oss et eksempel hvor denne metoden vil slå feil ut.

Et eksempel kan være at man har fire versjoner av et dokument, hvor for eksempel navnet er oppdatert, se figur 13. Hvis man nå søker etter den første versjonen hvor navnet er NTH, vil man få problemer med dette alternativet. Da vil man ikke finne noen versjoner, fordi det først søkes i den gjeldende tabellen for å finne dokument identifikatoren. I denne tabellen vil navnet være NTNU, og ikke NTH.

Forskjell på version(history) og version(all): Når all brukes vil man få ut alle versjoner av et dokument, selv om ikke alle versjonene stemmer med resten av where-delen. Et eksempel kan være søk i de versjoner som vises i figur 13. Vi kan da benytte all til å søke i alle versjoner av et dokument, som har universitetsnavnet lik NTH. Resultatet vil være at man får ut alle versjonen til dette dokumentet, også de hvor universitetsnavnet er NTNU. Hvis man i steden for benytter history, vil man bare få ut de versjoner som faktisk har et universitetsnavn som er NTH.

History er veldig fin å bruke hvis man ikke vet om det man søker etter faktisk er den siste versjonen av dokumentet. Hvis man søker etter dokumenter med universitetsnavn lik NTH, uten bruk av version, vil man ikke få noe resultater. Grunnen til dette er at det da bare vil kjøres søk i de gjeldende tabellene.



FIGUR 13. Viser hvilke versjoner som returneres ved bruk av version(all) og version(history)

11.4 Transaksjonshåndtering

Vår løsning er transaksjonsbasert. Det vil si at brukeren legger inn flere uttrykk og bestemmer selv når transaksjonen er over, ved å skrive commit eller rollback. Når brukeren sender inn det først uttrykket i en transaksjon, vil det hentes ut et tidsstempel fra databasen. Dette tidsstemplet vil bli brukt for alle uttrykk som legger inn tidsstemplet til transaksjonstidspunktet. Dette vil si at hvis brukeren kjører flere insert-uttrykk i en transaksjon, vil alle disse få det samme start tidspunktet. Grunnen til at vi gjør dette er at vi mener det er viktig at alle rader som legges inn under transaksjon, skal få det samme tidsstemplet.

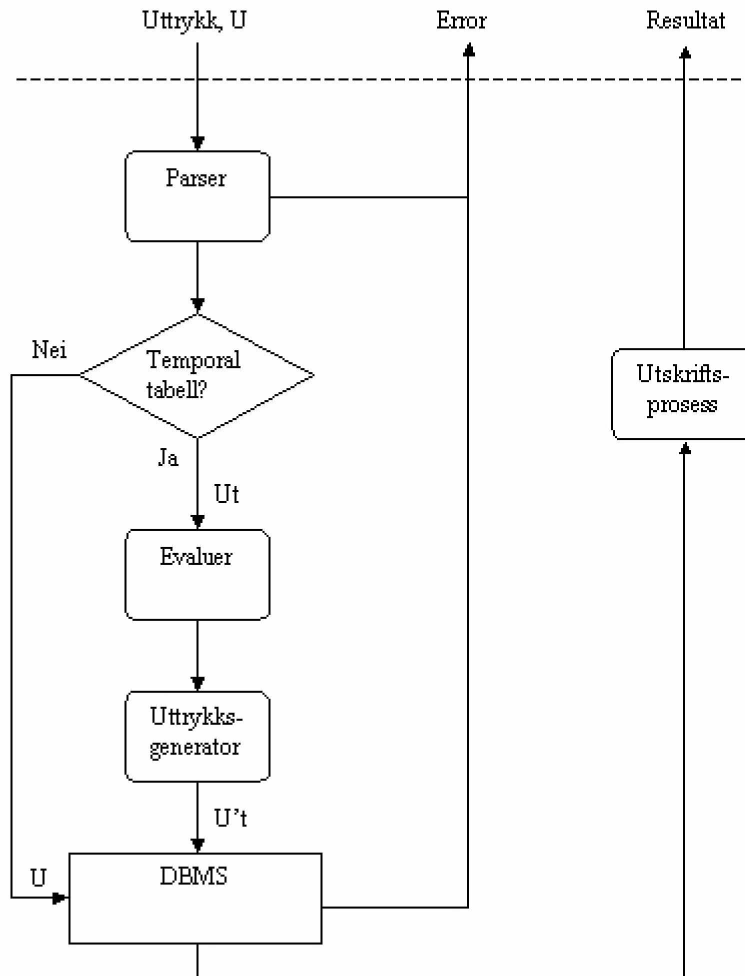
Et alternativ kan være å sette tidsstemplet på slutten av en transaksjon. Et argument for å gjøre dette vil være at da får det tidsstemplet når transaksjonen faktisk ble lagt inn i databasen. Hvis tidsstemplet settes i starten, så vil det si at flere rader blir lagt inn etter at tidsstemplet ble satt. Dette vil da ikke stemme helt. Allikevel har vi valgt å hente ut tidspunktet på starten. For da slipper vi å kjøre en oppdatering for å oppdatere alle tidsstemplene, når brukeren kjører en commit.

11.5 Kommunikasjon

Som skrevet over skal preprosessoren vår få inn ett uttrykk som den skal parse og evaluere. Ut i fra evalueringen lages det nye uttrykk som skal sendes til databasen. Resultatet fra databasen vil deretter bli evaluert og vist for brukeren. Denne prosessen er illustrert i figur 14.

Her kommer det inn et uttrykk U fra brukeren. Dette blir parset av vår egen SQL-parser. Går alt bra under parsing, så sjekkes det om tabellen det er snakk om er temporal. Grunnen til dette er at uttrykk som går på ikke-temporale tabeller skal sendes direkte til databasen, uten noen videre evaluering. Hvis tabellen er temporal, blir uttrykket evaluert. Ut i fra resultatet av evalueringen, lages det nye uttrykk. Disse nye uttrykkene vil ikke bestå av noen temporale aspekter, for at databasesystemet skal

skjønne dem. Resultatet fra databasen vil bli lagt inn i en utskriftsprosess, som viser resultatet til brukeren. Hvis det skjer noe feil under vår egen parsing eller fra databasen, vil det komme opp feilmeldinger til brukeren.



FIGUR 14. Flyt skjema for preprosessor

Vi vil nå vise et enkelt eksempel på hva som skjer med et uttrykk som brukeren skriver inn. Det kommer inn et insert-uttrykk fra brukeren som ser ut som følger:

```
insert into usertable values ( sys.xmltype.createxml( ‘<name>Lise</name>’ ))
```

Dette uttrykket vil først sendes gjennom parseren, for så å bli evaluert. Under evalueringen vil det bli laget et select-uttrykk som sjekker om tabellen er temporal:

```
select *
from temporal_table
where tablename = ‘usertable’
```

Om tabellen er temporal vil det bli generert to nye uttrykk som skal sendes til databasen. Dette vil være et uttrykk til tabellen usertable. Dette uttrykket vil bestå av det som brukeren ville legge inn og i tillegg en oid. Det andre uttrykket vil være tidsstemplet til XML-dokumentet som brukeren vil sette inn. Uttrykkene vil se ut som følger:

```
insert into usertable values ( 7, sys.xmltype.createxml('<name>Ida</name>'))
insert into time_usertable values ( 3, 7, sysdate )
```

Resultatet som databasen, vil videre sendes til en utskriftsprosess. Denne utskriftsprosessen vil videre skrive følgende uttrykk til brukeren:

```
insert into usertable values ( sys.xmltype.createxml( '<name>Ida</name>' ) )
1 row inserted
```

11.6 Ytelsesvurderinger

For at spørringene brukeren skriver skal gi et raskt svar er det viktig at tabellene er indeksert. Man må da tenke på hva som bør indekseres, altså hva brukeren kommer til å spørre om, og hvilke spørringer som kommer til å ta lang tid om tabellen ikke er indeksert.

I Oracle9i kan man lage flere indekser på XML-dokumenter. Vi har valgt å gjøre bruk av functional index på ulike elementer i XML-dokumentet. Disse kan som forklart under kapittel 9.3, legges både på existsNode- og extract-funksjonene. Det er viktig at disse brukes på de rette elementene. Vi har derfor valgt at brukeren selv må legge på indekser på de elementene som er viktige for brukerne. Vi kommer derfor ikke til å gjøre noe ekstra indeksering på XML-delen.

Det er flere andre alternativer for hvordan spørringer kan gi raskere svar til brukeren. Man kan for eksempel lage en tabell med flere av ordene som er i XML-dokumentene. Det man må tenke på da er hvilke ord som skal registreres i tabellen. Om alle ordene som fins i XML-dokumentene blir lagt i tabellen vil det ta nokså lang tid å søke etter ordene. En ting man da kan gjøre er å la brukeren selv velge hvilke ord han mener er viktige eller har stor sannsynlighet for å bli søket i ved senere anledninger. Dette kan gjøres ved for eksempel å ha et eget element med alle ordene.

Videre bør man ha god ytelse selv om man lagrer alle versjoner i databasesystemet. Dette gjør at vi måtte tenke ut en god løsning for hvordan man skulle lagre XML-dokumentene. Vi kom med flere løsninger, som vi vurderte mot hverandre. En beskrivelse av disse kommer vi inn på i kapittel 12.0.

XML tar mye større plass enn andre relasjoner. Forskning har påvist at samme data i XML-tekst er typisk tre ganger størrelsen på data i et java-objekt eller i en relasjonstabell [54]. Hovedgrunnene til dette er at taggene tar opp mye plass og det at all data er konvertert til streng-form. For eksempel vil et nummer, etter at det er konvertert til en streng, ta dobbelt så stor plass som når den ble representert som nummer.

Hvert XMLTYPE-objekt beholder ikke automatisk alle taggene, men peker på en post i Oracles bibliotek-cache som inneholder all tagg-informasjonen.

11.7 Edit script

Det finnes flere måter å versjonere XML-dokumenter på. En måte er bruk av såkalte edit script (også kalt delta, forklart under kapittel 5.0). Det vil si at man ikke lagrer alle versjoner som et helt dokument, men lager heller edit script for enkelte versjoner som inneholder de endringer som har skjedd fra en versjon til en annen.

Fordelen med disse er at man stort sett bruker en del mindre lagringsplass enn det å lagre alle versjoner som hele dokumenter i databasen. Man kan også samtidig legge tidsstempel på enkle elementer i XML-dokumentet, og ikke bare på hele XML-dokumentet.

En annen fordel med edit script er at det vil være enkelt å finne differansen mellom to versjoner. Da sammenligner man hva som er endret på og når endringen var, ved å se på tidsstemplet til de ulike elementene.

Denne løsningen har også flere ulemper, særlig nå som det begynner å bli billigere og billigere med disker. Dermed er det ikke så stor fordel at edit script tar mindre plass. Videre er det ikke alltid det er så mye plass å spare på å benytte edit script. Hvis det skjer mye endringer mellom versjonene av XML-dokumentet, kan edit script bli nesten like store som selve XML-dokumentet. Det vil også ta en del tid å lage disse edit script og å bruke dem. Hvis man spør etter en hel versjon, så må det kanskje leses gjennom flere edit script for å få generert XML-dokumentet. Dette kan ta lang tid, særlig hvis de ligger på ulike plasser på disken. Da kan det bli mye flytting av diskhode, som ofte er det som tar lengst tid. Da kan det være vel så bra å heller kunne hente ut hele dokumentet fra en plass på disken.

Dette gjør at vi har kommet frem til at det ikke er så mye å spare på å bruke edit script. Det vil også bli mye mer programmering ved bruk av edit script, og vi har derfor heller ikke tid nok til å programmere dem.

12.0 Lagringsalternativer

Når brukeren av vårt program skriver inn et uttrykk, så er det en del evaluering av uttrykket. Det må blant annet fjernes alle temporale ord som vi har lagt til spørrespråket. Videre blir uttrykket gjort om til flere nye uttrykk som skal sendes til databasesystemet. Med tanke på at spørringer i XML-dokumenter vil ta en del tid, er det om å gjøre å finne en så optimal løsning som mulig når de temporale tabellene skal bli modellert. Vi har kommet frem til flere mulige måter å modellere tabellene på. For de ulike modelleringsmetodene, vil det også være ulike uttrykk som må sendes til databasesystemet.

For å sammenligne de ulike løsningene, må vi derfor sammenligne de ulike løsningene. Denne sammenligningen vil gå på hvor stor kostnaden vil være for disse uttrykkene. For å finne denne kostnaden har vi valgt å kjøre noen enkle tester. Testene vil være en forenkling av virkelige kostnader, men vi mener det er nok nøyaktig til å kunne vurdere de ulike løsningene i forhold til hverandre.

Testresultatene vil kunne være en indikasjon på hvilken løsning som bør velges. På grunn av en del forenklinger vil det måtte være en del forskjell mellom resultatene til løsningene, for å kunne trekke ut hvilken som er best egnet til bruk i vårt program. Testene vil også være en god indikasjon for oss om hva som vil ha størst innflytelse på kostnaden. Dette kan man ta hensyn til i implementasjonen.

Dette kapitlet vil være lagt opp som følger. Vi starter med en beskrivelse av de løsningene vi har tenkt å sammenligne. Deretter vil vi beskrive selve kostnadsmodellen vår, før vi kommer inn på selve gjennomføringen av testen. Til slutt vil vi komme med resultatene og en konklusjon.

12.1 Løsningsforslag

Vi velger å sammenligne tre modelleringsforslag. Her vil vi komme inn på hva de ulike forslagene går ut på. figur 15 viser hvordan forslagene vil se ut. Her illustreres det med hvor mange tabeller som trengs, og hvilke type versjoner som blir lagret i dem. De tre forskjellige løsningen er:

- **TX1:** En tabell med gjeldende og historiske
- **TX2:** To tabeller; en for gjeldende og en for historiske
- **TX3:** To tabeller; en for gjeldende, og en for historiske og gjeldende

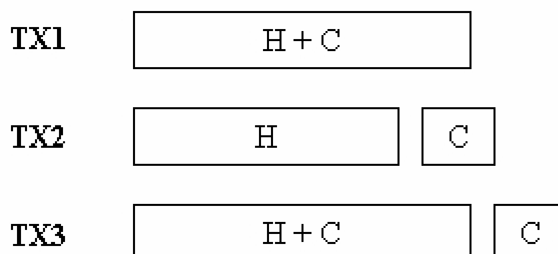


FIGURE 15. Oversikt over antall tabeller og hva de lagrer for alternativene.
H - historiske versjoner og C - gjeldende versjoner

I beskrivelsen av forslagene under, vil vi bruke navnet usertable på navnet til den tabellen som brukeren lager. For hver av de tabellene som vi beskrev under de ulike alternativene, vil det komme en tabell i tillegg som vil lagre tidsstemplet til XML-dokumentet. Denne tabellen vil få ordet time_ foran tabellnavnet.

12.1.1 TX1: En tabell med gjeldende og historiske

TX1 er det enkleste alternativet vi har modellert. Her har vi to tabeller, usertable og time_usertable. Tabellen usertable vil være lik den tabellen som brukeren lager i create-uttrykket sitt, bortsett fra at det legges til en ny kolonne med en objekt identifikator (oid). oid-variablen vil bli brukt til å gi alle versjoner av alle XML-dokumenter en unik identifikator. Identifikatoren vil følge en versjon gjennom hele livssyklusen, og den vil ikke bli brukt på ny om en versjon blir slettet. Variablen sier ingenting om til hvilke dokument de ulike versjonene hører til.

time_usertable vil inneholde docid, oid, t_start og t_end. oid er fremmednøklen til oid i usertable, mens doc_id angir hvilket dokument en versjon hører til. Alle versjoner til samme dokument vil få den samme doc_id. t_start og t_end angir henholdsvis tidspunktet når versjonen av dokumentet ble lagt inne i databasesystemet og tidspunktet når versjonen ikke lengre er den siste, altså at versjonen ikke lengre er den aktuelle. figur 16 viser hvordan tabellene vil se ut, for dette alternativet.

Usertable

Oid	XML
3	<name>Mons</name> <age>51</age>
4	<name>Kari</name> <tel>23232323</tel>
6	<name>Mari</name> <age>26</age>

Time Usertable

DocId	Oid	Start tid	Slutt tid
3	3	02.03.02 12:45:00	27.03.0 07:14:0
3	4	27.03.02 07:14:00	UC
6	5	04.04.02 12:14:00	UC

FIGURE 16. Tabeller som lages for TX1

12.1.2 TX2: To tabeller; en for gjeldende og en for historiske

I TX2 er tanken at vi har fire forskjellige tabeller. Disse er usertable, usertable_old, time_usertable og time_usertable_old, se figur 17. usertable ser lik ut som den tabellen brukeren oppretter bortsett fra at vi har lagt til en oid. I usertable vil bare den siste versjonen av dokumentene ligge. Tabellen usertable_old vil være helt lik som usertable. Forskjellen ligger i hva tabellen lagrer, som vil være gamle versjoner av dokumentene. Dette vil si alle versjoner foruten om den siste versjonen.

Tabellene time_usertable og time_usertable_old lagrer tidsstemplene for henholdsvis XML-dokumentene i usertable og usertable_old. time_usertable vil bestå av kolonnene oid, doc_id og t_start. oid vil være fremmednøklen til oid i usertable, mens docid er en dokument identifikator for å holde rede på hvilken

versjon som hører til hvilket dokument. t_start lagrer tiden som versjonen ble lagt i databasen. time_usertable_old vil være tilsvarende, men med en ekstra kolonne; t_end. t_end vil være tidspunktet når versjonen ble slettet eller oppdatert med en ny versjon.

Usertable

Oid	XML
3	<name>Mons</name>
4	<name>Kari</name> <tel>23 23 23 23</tel>
5	<name>Mari</name> <alder>26</alder>

Usertable Old

Oid	XML
1	<name>Kari</name> <tel>21 21 21 21</tel>
2	<name>Mari</name> <alder>24</alder>

Time Usertable

DocId	Oid	Start tid
3	3	20.04.02 12:45:00
1	4	24.04.02 08:05:00
2	5	03.05.02 09:00:00

Time Usertable Old

DocId	Oid	Start tid	Slutt tid
1	1	02.03.02 12:00:00	24.04.02 08:05:00
2	2	20.03.02 07:15:00	03.05.02 09:00:00

FIGURE 17. Tabellene som lages for TX2 og TX3

12.1.3 TX3: To tabeller; en for gjeldende og en for historiske og gjeldende

TX3 er nokså lik TX2 med sine fire tabeller usertable, usertable_old, time_usertable og time_usertable_old. Det er en helt lik oppbygning av tabellene. Forskjellen mellom dette alternativet og TX2 er at de usertable_old vil inneholde alle versjoner, både nye og gamle versjoner. Det vil være tilsvarende for time_usertable_old. Dermed vil de siste versjonene være lagret i to tabeller.

Dette alternativet er med på grunn av at det kan være en fordel å bare søke i en tabell hvis man vil søke i absolutt alle versjoner av et dokument. Da trenger man bare å søke i usertable_old.

12.1.4 Sammenligning

Sammenligner man disse løsningsalternativene når det gjelder hvor mye som blir lagret, vil det være forskjeller. TX1 og TX2 vil være ganske like. Når det gjelder TX3 vil det være den løsningen som bruker mest lagringsplass. Den vil være en relativt dårlig løsning for systemer som har veldig mange ulike dokumenter, og ikke så mange versjoner av hver. Da vil TX3 nærme seg det dobbelte av lagringsplassen som TX1 og TX2 vil bruke. Ved systemer hvor det er få dokumenter, men mange versjoner av hvert, vil man ikke merke så mye forskjell mellom alternativene. Dermed er det avgjørende for valg av løsning, i hvilken retning hvert systemer heller mot. Videre vil det være avgjørende å vite hvor mye lagringsplass man har til rådighet. Hvis lagringsplassen

ikke er en avgjørende faktor, kan man heller se på kostnaden til de ulike løsningene når en bruker kjører en spørring.

12.2 Kostnadsmodell

For å kunne sammenligne de tre versjonene ut i fra resultatene vi fikk, har vi laget en kostnadsmodell for hver av løsningene.

12.2.1 Antakelser

Vi har gjort antagelser om hvor mange % hvert av de forskjellige uttrykkene bruker i et databasesystem. Vi har laget flere sett med antagelser, og disse kan finnes i tabell 5.

Videre har vi valgt å se bort fra create- og drop-uttrykkene. Vi mener at disse uttrykkene sjeldent vil bli brukt i praksis, slik at de vil ha veldig liten innvirkning på resultatet.

12.2.2 Innflytelser på resultatet

Oracle9i er et stor databasesystem, som takler stor pågang. I dette systemet, som ved andre systemer, kan det bli for mange ressurser i bruk på en gang, og det kan oppstå en flaskehals. Vi har en egen disk til rådighet, slik at antall bruker eller ressurser som er i bruk i databasen ikke vil ha noe betydelig innvirkning på resultatet vårt.

Bufferstørrelsen vil ha mye så si for kostnadene til de forskjellige uttrykkene. Om bufferstørrelsen er høy vil dette redusere antall diskoperasjoner. Bufferstørrelsen til DBMS er satt til 50 MB gjennom forsøkene. Dette vil være en tilstrekkelig lav størrelse for at DBMS tvinges til å skrive/lese til disk, og at ikke for mye blir liggende i buffer. Selv om vi har satt ned bufferstørrelsen til DBMS, vil operativsystemets buffer også bli brukt. Størrelsen på dette bufferet får vi ikke gjort noe med.

12.2.3 Kostnadsmodellen

For å sette opp en kostnadsmodell har vi lagd en forkortelse for hvert av uttrykkene som vi kommer til å bruke i formlene. Disse er som følger:

I - Insert uttrykk

D - Delete uttrykk

U - Update uttrykk

S - Select uttrykk

Den totale tiden til programmet vil være den tiden det tar fra en bruker har skrevet inn et uttrykk og til det kommer opp et resultat. Denne tiden kan vises av formelen:

$$Total_{time} = app_{time} + statement_{time}$$

hvor

app$_{time}$ er tiden programmet bruker på å parse brukerens uttrykk, gjøre det om til nye uttrykk til databasesystemet, og motta resultatet fra databasesystemet og skrive det ut til brukeren.

statement$_{time}$ er tiden det tar for databasesystemet å kjøre uttrykket og sende resultatet til programmet. Dette kan være insert, delete, update og select. Formlene til disse uttrykkene står i tabell 4.

TABELL 4. Kostnadsfunksjoner til TX1, TX2 og TX3 for insert, delete, update og select

Uttrykk	**TX1**	**TX2**	**TX3**
Insert	$S + (2 \times I)$	$S + (2 \times I)$	$S + (4 \times I)$
Delete	$(2 \times S) + U$	$(2 \times S) + (2 \times I \times rad)$ $+ (2 \times D)$	$(2 \times S) + (2 \times D) + U$
Update	$(2 \times S) + (2 \times I) + U$	$(2 \times S) + (2 \times I \times rad)$ $+ (2 \times U)$	$(2 \times S) + (2 \times I \times rad)$ $+ (3 \times U)$
Select	$3 \times S$	$5 \times S$	$3 \times S$

Vi tar utgangspunkt i at XML-dokumentene er indekserte, slik at det ikke vil være noe særlig forskjell i tid om man søker på XML eller om man søker på andre kolonnetyper. I de videre beregningene gjør vi også en del forenklinger. Vi vil bare beregne tidskostnader på fasen hvor DBMS er i normal modus. Vi ser altså for eksempel bort fra hvordan DBMS reagerer rett etter krasj. Vi setter også begrensning på at vi bare tester på lagring og uthenting av XML-dokumenter.

Ved in-place oppdatering kan en transaksjon commite etter at dens loggposter har blitt skrevet til disk. Sider som har blitt modifisert blir ikke skrevet til disk umiddelbart. Dette blir gjort som del av bufferutskifting og bruk av sjekkpunkt. I et sjekkpunktintervall (avstanden mellom to sjekkpunkter) kan et dokument oppdateres flere ganger før det blir skrevet til disk.

12.2.4 Sammenligning av løsningene

For å sammenligne de ulike løsningene har vi satt en faktor på hver av uttrykkene. Dette for å kunne prioritere de ulike uttrykkene etter hvor ofte de blir brukt. Faktorene vil kunne variere fra hva databasen blir brukt til.

TABELL 5. Oversikt over faktorer for de ulike uttrykkene i prosent

Uttrykk	Sett 1	Sett 2	Sett 3	Sett 4	Sett 5
Insert	16	10	10	12	18
Delete	1	3	2	3	5
Update	3	15	8	10	12
Select	80	72	80	75	65

Ved å bruke disse faktorene kan vi regne ut totalformeler for løsningsalternativet. Her er totalformelen ved bruk av faktorsett 1:

$$Total_{time} = app_{time} + (16 \times Insert) + (1 \times Delete) + (3 \times Update) + (80 \times Select)$$

Denne og de tilsvarende formlene for de andre faktorsettene vil bli brukt i utregning av resultatet. Dette kommer vi tilbake til i (ref fap resultat)

12.3 Beskrivelse av testmateriale

Som testmateriale har vi ca 270000 XML-dokumenter. Disse dokumentene er ikke versjonerte. Grunnen til dette er at det ikke er relevant for testingen om XML-dokumentene er versjonerte.

12.4 Gjennomføring av testene

Selve testen består av ulike uttrykk som vi skal teste mot databasesystemet. For hvert uttrykk registreres det hvor mange fysiske lese- og skriveoperasjoner databasesystemet gjør til disk for å utføre uttrykkene. Vi har laget flere uttrykk for hver av de ulike SQL-uttrykkene; select, insert, delete og update. Vi har valgt å ikke teste på uttrykkene create table og drop table, siden de er så sjeldent brukt og vil derfor ha liten relevans i testresultatene våre.

Vi bruker en meget forenklet måte å beregne kostnaden til uttrykkene på, og den vil heller ikke være helt nøyaktig. Vi mener allikevel at den vil være god nok i vårt tilfelle for å kunne å skille mellom de tre lagringsmåtene, slik at den beste løsningen kan implementeres.

Alle XML-dokumentene vil bli lagt inn i en tabell. Deretter kjører vi en transaksjon med uttrykk til databasen. Denne transaksjonen vil bestå av en type uttrykk om gangen, men med et stort antall av dette uttrykket. Grunnen til at vi kjører en så stor transaksjon, er for å finne de virkelige diskoperasjonene, og unngår at DBMS bare

skriver til logg. Når transaksjonen er kjørt gjennom, henter vi ut antallet diskoperasjoner som transaksjonen produserte.

For å finne antall diskoperasjoner, både lesing og skriving, har vi brukt Oracle Enterprice Manager.

12.4.1 Tidsbruk for innlegging i oracle database

Vi har også gjort tester på hvor lang tid det tar å legge inn dokumenter i databasen. Dette ble gjort ved at vi laget et sorterings/innleggings program til å legge inn dokumenter i databasen. Vi har valgt å legge inn alle dokumentene som begynner på A og dette er ca 20 000 dokumenter. Tiden det tar å legge inn alle dokumentene som begynner på A ble så registrert. Resultatet av denne testen kan man se i figur 18.

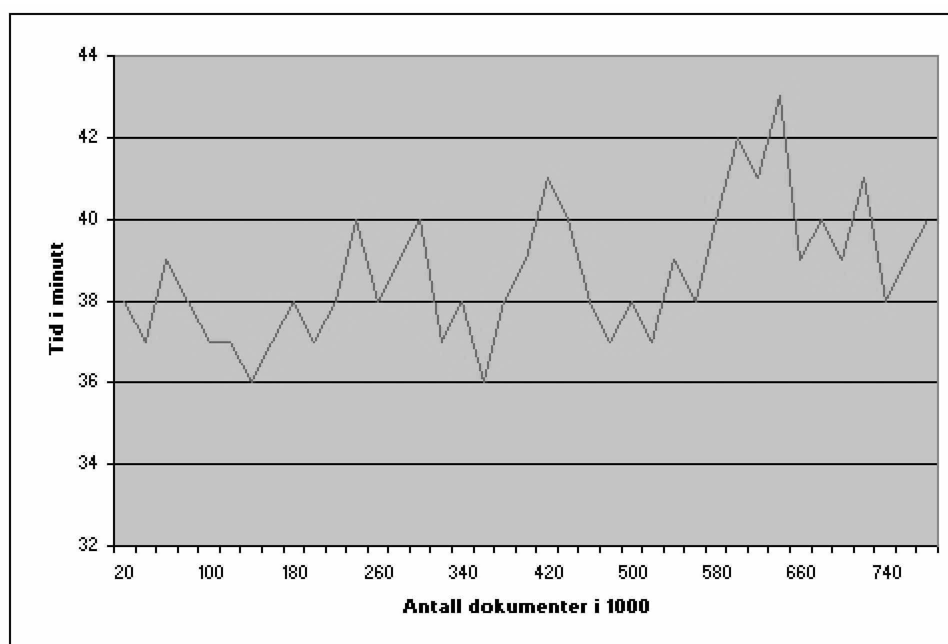


FIGURE 18. Viser tidsbruken for å legge inn dokumenter i Oracle-databasen

Vi ser her at tidsbruket er nokså stabilt i forhold til hvor mange dokumenter som blir lagt inn i databasen. Tidsbruken øker noe etter hvert som det blir lagt inn flere data, men tidsbruken er likevel stabil. Dette kan tyde på at Oracle legger inn dokumentene i sekvensielt.

12.5 Resultater

Figur 19 viser en sammenligning av kostnader for de ulike utrykkene. Figur 20 viser en graf over de totale kostnadene til hver av lagringsalternativene, TX1, TX2 og TX3, når det er brukt ulike vektingsfaktorer for hver av de forskjellige uttrykkene. Faktorene her er hentet fra tabell 5.

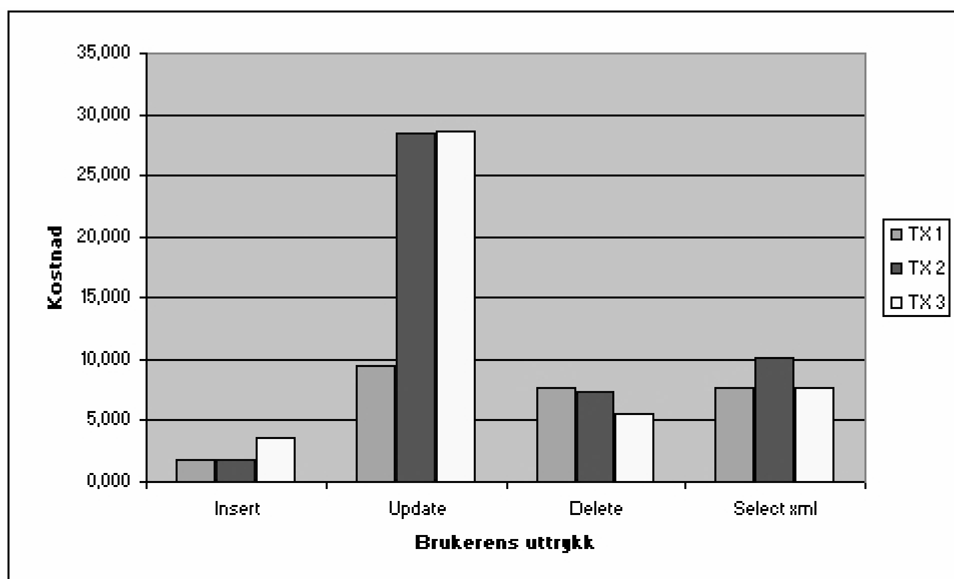


FIGURE 19. Sammenligning av kostnader for de ulike uttrykk

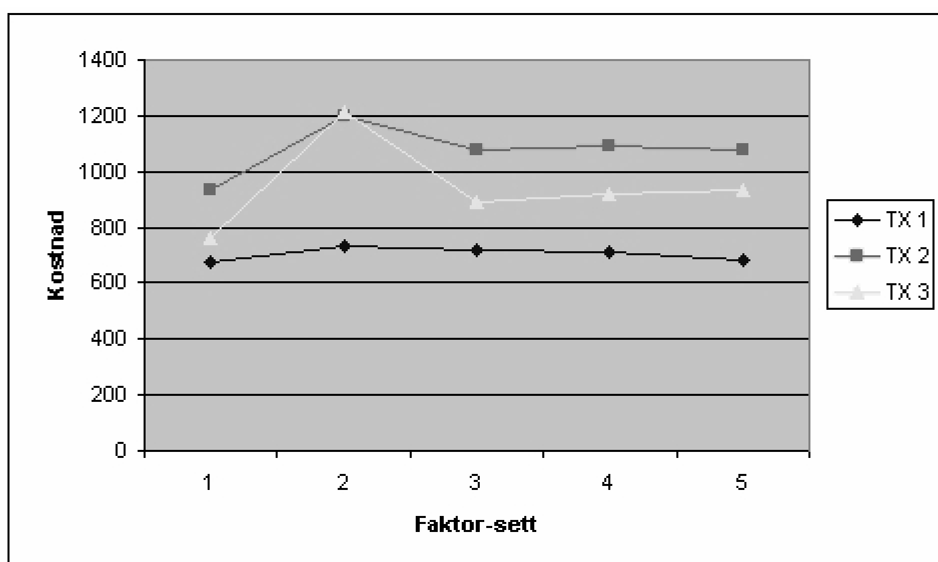


FIGURE 20. Totalkostnader ved ulik vekting av uttrykkene

12.6 Konklusjon

Vi ser av figur 19 at uttrykket som bruker flest diskoperasjoner er update. Delete og select er det nokså like kostnader på, mens insert har noe færre diskoperasjoner. Create og drop vil være operasjoner som blir utført så sjeldent at disse er valgt å ikke ta med.

Målet med testingen var å se om det var mulig å finne ut om det var en av lagringsalternativene som utmerket seg ved å være mer kostnadsøkonomisk enn de andre. Ved insert ser vi at TX1 og TX2 er operasjoner som er noe billigere å gjennomføre enn for TX3. Ved update utmerker TX1 seg til å være en meget billigere løsning enn de andre to. Det er ikke de helt store forskjellene ved delete og select.

Ut fra disse resultatene kan vi ikke trekke ut en lagringsmetode som er gjennomgående mer kostnadseffektiv enn de andre. Siden de forskjellige uttrykkene blir benyttet i ulik mengde har vi altså valgt å se på fem forskjellige sett med vektingsfaktorer. Resultatet fra disse beregningene befinner seg i figur 20.

Select er det uttrykket som brukeren vil benytte desidert oftest. Insert og update vil kunne være nokså like, avhengig av hvilket system det er snakk om. delete vil bli kjørt nokså sjeldent, men ofte nok til at vi har valgt å ta den med.

I det første settet er det en stor overvekt av select og insert. Vi ser her at TX1 og TX3 komme godt ut, mens TX2 vil være litt mer kostbar.

I sett to er hovedvekten på select og update. Dette vil kunne være et system hvor dataene oftere blir oppdatert, enn slettet eller lagt inn i databasen. Et eksempel på dette kan være en database hvor dataene blir lagt inn en gang, og deretter blir ofte oppdatert heller enn at det blir lagt inn nye data. I dette settet ser vi at TX2 og TX3 får like kostnader, mens TX1 gir lave kostnader.

Sett tre, fire og fem har alle insert-verdier som er høyere enn update-verdiene. Vi ser da at kurven stabiliserer seg.

Vi ser at TX1 kommer best ut i testene. Likevel mener vi det er flere grunner til å ikke velge dette lagringsalternativet. For det første vil det problematisk for TX1 være om man bare vil søke i gjeldende tabeller. Siden historiske og gjeldende versjoner ligger i en og samme tabell, vil det ta mye lengre tid å utføre søket.

En annen ting som er negativt med å bruke TX1 som lagringsalternativ er at tabellene kan bli veldig store. I store systemer vil dette kunne være nokså merkbart. For små systemer vil TX1 være en god løsning.

For systemer som bruker edit script og ombruk av dokumentdeler vil TX1 kunne komme dårligere ut enn TX2 og TX3. Grunnen til dette er at når dokumentene blir delt opp, og ikke lagret på en og samme plass, som for eksempel i en CLOB på en kolonneplass, vil det være viktig med gruppering av dokumentdelene. Dette vil kunne la seg gjøre i en gjeldende tabell, mens i en historisk vil det bli noe verre. Her vil det antakeligvis være mye mer data og vil derfor ta lang tid å søke etter ulike versjoner.

Fordelen med TX3 er at det er mulig å søke i begge delene. Det er veldig ofte det vil være bedre å søke i tabellen med historiske og gjeldende dokumentversjoner. Om man

bare vil søke etter historiske versjoner, vil det være mange flere dokumenter som må søkes gjennom i TX3 enn tilsvarende i TX2. Grunnen til dette er at i TX2 ligger de historiske versjonene i en egen tabell, mens det gjør dem som sagt ikke i TX3. En annen ulempe med TX3 er at den historiske tabellen vil være mye større enn den historiske tabellen til TX2. Når det gjelder XML-dokumenter som lagres i databasen, så vil det merkes når mengden XML-dokumenter øker. Det vil gå greit så lenge man gjør bruk av indekser, men det blir fort verre uten indekser. I appendiks E har vi lagt inn noen testdata for en annen test vi kjørte. Den tester på hvor lang tid det tar å søke i XML-dokumenter som ikke er indeksert. Vi testet på ca 50 000 XML-dokumenter i databasen, og fikk veldig høye søketider. Søkte man etter elementer fra dokumentene som fantes i omtrent alle XML-dokumentene, tok det over 1 time. Vi gjorde også søk på elementer som bare var i ca halvparten av XML-dokumentene, og da tok det i underkant av 10 minutter. Dette viser at det er absolutt en nødvendighet med indekser på XML-dokumentene. Her vil det bli en avvegning på om man vil bruke masse lagringsplass til flere indekser, eller om man skal tillate at det tar så lang tid.

Med tanke på hvor mye mer lagringsplass TX3 bruker enn TX2, og at de fikk ganske like testresultater, bestemte vi oss for å jobbe videre med TX2.

13.0 Implementasjon

Vår oppgave var å designe og implementere store deler av en preprosessor som skal gjøre en bruker-tabell til en temporal tabell. For å klare dette måtte vi lage et eget grensesnitt over preprosessoren som brukeren skal benytte. Preprosessoren evaluerer uttrykket som brukeren skriver inn og gjør det om til nye uttrykk som sendes til databasesystemet.

Vi har allerede beskrevet funksjonalitet og designet til preprosessoren. Vi vil nå gå inn på hva som faktisk ble implementert og hvordan. Det vi ikke fikk implementert og de fremtidsutsikter vi har, vil bli beskrevet i kapittel 14.0. Først vil vi kort skrive om de verktøyene vi har brukt, før vi går dypere inn i det vi har implementert. Da vil vi gi en beskrivelse av transaksjonshåndteringen vi har implementert, parseren vi har lagd og til slutt hvilke klasser preprosessoren består av.

13.1 Verktøy og databasesystem

Vi har valgt å utvide Oracle9i-databasesystemet, som vi beskrev i kapittel 9.0. Dette databasesystemet er installert på skolen, og er derfor helt klar til bruk. Videre valgte vi å bruke Java 2 SDK 1.4.0 for å programmere preprosessoren til databasesystemet.

For å kunne parse uttrykket som brukeren skriver inn slik vi ville, måtte vi lage en egen SQL-parser. Denne parseren lagde vi ved hjelp av programmet JavaCC [35]. Der måtte vi lage en egen fil med funksjoner over hvordan ulike kommandoer kan skrives. Deretter ble denne filen kompilert av JavaCC som lagde 7 java-filer av parseren. Denne parseren vil bli nærmere beskrevet i kapittel 13.2.

13.2 SQL parser

I kapittel 10.3 beskrev vi ordene vi ville legge til spørrespråket SQL, for å kunne benytte spørrespråket til spørring i temporale tabeller. For å kunne bruke disse ordene i tillegg til SQL, måtte vi kunne evaluere uttykket som brukeren skriver inn, før det ble sendt til databasesystemet. Dette ble gjort blant annet for å fjerne de ord vi har lagt til og fordi uttykket ofte skal endres til et eller flere andre uttrykk. Dermed ble det til at vi måtte lage vår egen SQL-parser. Dette gjorde vi med programmet JavaCC.

JavaCC skal ha inn en jj-fil, som forteller hvilke uttrykk brukeren kan skrive inn og hva som skal gjøres med uttrykkene. For hvert uttrykk som kommer inn, blir det lagret ulike variabler. Disse variablene bruker vi i neste steg, som er evaluering av uttrykket. Her nevner vi de variablene som sendes videre til evalueringen:

- *Uttrykket:* Selve uttykket sendes videre til evaluering
- *Version:* Det sendes tilbake en boolsk variabel som sier om brukeren har benyttet seg av ordet version
- *Tabellnavnet:* For å gjøre evalueringen raskere, trekker vi også ut tabellnavnet som brukeren skriver inn, i en egen variabel. Når det gjelder for select-uttrykket, kan det være flere tabellnavn. Disse legges da inn i en vektor
- *Kolonnenavn:* For select-uttrykket lagres alle kolonnenavnene som brukeren vil ha ut etter spørringen

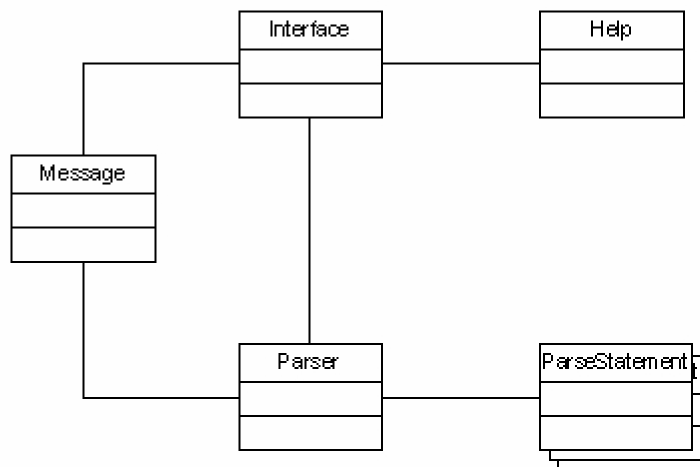
- *Set-verdiene til update-uttrykket:* For update-uttrykket lagres de kolonnene som skal oppdateres i en egen tabell sammen med de verdiene de skal settes til

For de XPath- og XML-uttrykk som brukeren skriver inn, skal det ikke gjøres noen endringer under evalueringen av dem. Derfor har vi valgt at parseren bare leser dem og sender dem videre uten å se på synktaksen. Dermed blir bare XPath- og XML-uttrykkene med som de står, i uttrykket som sendes videre til evaluering

Når denne jj-filen så er skrevet ferdig, kjøres den gjennom kompilatoren til JavaCC. Kompilatoren genererer syv java-filer. Disse java-filene kopierer vi inn i vårt program. I neste delkapittel kommer vi nærmere inn på hvordan disse filene er satt sammen med resten av programmet vårt.

13.3 Klasser

I dette kapitlet kommer vi inn på hvilke klasser vi har laget for preprosessoren og gir en kort beskrivelse av dem. Vi har laget fire klasser, som vises i figur 21.



FIGUR 21. Klassediagram over vår preprosessor

Her kommer vi med en beskrivelse av hver av klassene.

13.3.1 Interface

Denne klassen lager selve grensesnittet mot brukeren, figur 9. Alle mus-, tastatur- og knappe-hendelsene blir tatt opp fra denne klassen. Når brukeren velger å kjøre et uttrykk, hentes uttrykket ut og sendes videre til Parser-klassen. Etter evaluering og eksekvering mot databasesystemet, returneres resultatet tilbake til Interface-klassen. Da hentes en funksjon opp fra Message-instans som kom fra Parser-klassen. Denne funksjonen returnerer en streng som skal vises i resultatområde.

Mus-hendelsene blir brukt hvis brukeren vil kopiere eller lime inn tekst fra grensesnittet. Da kalles det opp en popup-boks som har to valg; kopier og lim inn. Når

det gjelder tastatur-hendelsene, er det en tast brukeren kan bruke. Dette er F9-tasten som brukes for å kjøre et uttrykk som brukeren har skrevet inn.

For de fem knappene som står i menyen til grensesnittet, må vi ha en egen hendelsestrigger. Denne triggeren sjekker hvilken hendelse det er og utfører operasjoner etter det.

- *Connect:* kobler opp mot databasen Oracle9i. I vår løsning kobles brukeren opp automatisk når knappen trykkes. Brukeren trenger ikke å skrive inn noe selv. Dette er noe som må endres på i en utvidelse. Da bør brukeren selv kunne skrive inn brukernavn, passord og vertsmaskinnavn til databasen det skal kobles opp mot.
- *Disconnect:* kobler brukeren fra databasen.
- *Help:* kaller opp hjelpesiden vi har laget. Denne er beskrevet under klassen Help.
- *Execute:* her vil uttrykket som brukeren har skrevet inn i uttrykksområde kjøres. Det vil være samme funksjonalitet om man trykker på F9-tasten på tastaturet.
- *Exit:* avslutter programmet. Hvis det fortsatt er en databasekobling mot databasen, så vil den først bli koblet ned.

13.3.2 Parser

Denne klassen står for hele evalueringen av uttrykket. Uttrykket kommer fra Interface-klassen, og blir sendt direkte videre til ParseStatement-klassen for parsing av uttrykket. Hvis alt går bra under parsing, starter evalueringen. Dette gjøres ved at hvert uttrykk har hver sin funksjon, som blir kalt opp fra ParseStatement-klassen. Evalueringsfunksjonen går steg for steg gjennom uttrykket ved hjelp av de variabler som kom fra parseren (disse ble beskrevet i kapittel 13.2). Det lages nye uttrykk som kjøres mot databasesystemet.

For hvert uttrykk fra brukeren som evalueres, blir det laget en message-instans, som lagrer all nyttig informasjon om evalueringen. Hvilken informasjon dette er kommer vi tilbake til i neste delkapittel. Når evalueringen er ferdig sendes denne message-instansen til Interface-klassen.

13.3.3 Message

Dette er en instans-klasse som lagrer all informasjon om hvordan evalueringen av de ulike uttrykkene går. Informasjon som lagres er:

- uttrykket og hvilken type uttrykk det er
- om det er en temporal tabell
- om transaksjonen ble commitet
- feilmelding og i hvilket steg den kom, hvis transaksjonen ikke ble commitet
- resultatet etter kjøring mot databasesystemet
- antall rader og evt kolonner resultatet består av. For select gjelder dette antallet for resultatsettet, mens for update og delete sier det hvor mange rader som ble påvirket

Klassen har funksjoner for å sette hver av disse variablene. Videre har klassen er konstruktør som initsialiserer alle variablene og en Print-funksjon som lager en resultat-melding.

For Print-funksjonen genereres det ulike meldinger for hver av de ulike uttrykkene. For create og drop, vil det stå at tabellen er laget/slettet. Mens for update, insert og delete vil det stå hvor mange rader som ble påvirket. Når det gjelder select-uttrykk, så vil det være litt flere operasjoner. Her har vi laget en algoritme, som sammenligner lengden på teksten i header og første rad. Dette ble gjort for å finne korrekt størrelse på kolonnene. Hvis det er noen rader lenger nede som har en større bredde under en bestemt kolonne enn den som ble satt, vil teksten her komme over flere linjer.

Til slutt for Print-funksjonen, blir meldingen konkatenert til en streng som kommer som input til funksjonen. Denne strengen returneres til Interface-klassen.

13.3.4 Help

For at brukeren skal kunne få hjelp med enten grensesnittet eller selve spørrespråket som vi har laget, har vi laget egne hjelpe-sider. Denne klassen lager et vindu som viser disse hjelpefilene. Dette vinduet blir kalt opp av programmet, når brukeren trykker på help-tasten.

13.3.5 ParseStatement

Dette gjelder de syv filene som dekker SQL-parseren. For å knytte dem sammen med resten av programmet, måtte vi bare gjøre noen små endringer. Dette gjaldt for hovedfilen (ParseStatement), som hadde en main-funksjon. Vi hadde allerede en main-funksjon som står i Interface-klassen, og denne funksjonen måtte dermed gjøres om til en annen funksjon. I tillegg måtte vi legge til noen kommandoer for å få alt til å fungere fint. Ellers gjorde vi ingen endringer, og kommer derfor ikke til å beskrive disse klassene noe nærmere. Refererer heller til kapittel 13.2 , for en mer detaljert beskrivelse.

14.0 Evaluering av programmet

Vi vil i dette kapitlet komme med vår egen evaluering av programmet vårt. Da programmet ikke er helt ferdig, vil det være en del svakheter som trenger å forbedres. Vi kommer til å starte med en de styrker vi mener programmet har, før vi går over til svakhetene. Til slutt kommer vi med en del utvidelsesmuligheter. Flere av disse utvidelsesmulighetene vil være basert på de svakheter programmet vårt har.

14.1 Styrker

Måten vi har modellert de temporale tabellene på, mener vi er den beste måten å gjøre det på. Vi har kjørt en del tester som underbygger valget vårt. Dette gjør at programmet vil være raskt.... se litt på hvilke resultater vi får.

Programmet vårt har støtte for både XML- og relasjons-data. Dette er en vesentlig fordel, med tanke på at en bruker ofte vil bruke begge deler, hver for seg eller sammen. Programmet er likevel optimalisert for XML.

14.1.1 Grensesnitt

Oracle9i bruker blant annet SQL*Plus som grensesnitt mot databasesystemet for de brukere som vil skrive inn sine uttrykk manuelt. Etter å ha brukt dette grensesnittet, mener vi at det har en del mangler (jamfør kapittel 10.1). Vi mener å ha utbedret disse manglene på en god og brukervennlig måte. Brukeren vil fortsatt kjenne seg igjen og vet hvordan grensesnittet skal brukes. Disse forbedringen går blant annet på at brukeren kan endre på et uttrykk han holder på å skrive.

I tillegg har vi lagt til hjelpesider som forklarer både grensesnittet vårt, og de temporale utvidelsene vi har gjort på spørrespråket. Dermed blir det enkelt for en bruker å få hjelp, om det er ønskelig.

14.1.2 Spørrespråket

De temporale utvidelsene vi har gjort på SQL mener vi er enkle å forstå og bruke. Vi mener vi har gjort dem effektive og de dekker de meste sentrale delene som trengs i et temporalt spørrespråk. Videre påvirker det ikke språket som Oracle allerede brukere.

Måten vi har satt opp programmet på, gjør det enkelt å utvide spørrespråket. Dette gjelder både utvidelser i forhold til SQL som Oracle allerede bruker, og temporale utvidelser som senere kan være interessante.

Når brukeren kjører en spørring mot en temporal tabell, vil bare tidsstemplet vises hvis brukeren spør etter det. Om brukeren for eksempel skriver en spørring `select * from ...` vil bare de kolonnene som brukeren selv har lagt i tabellen bli skrevet ut. Dette viser at programmet vårt støtter både temporale og ikke-temporale tabeller.

Videre må brukeren også spesifisere om han vil at det skal søkes i de historiske tabellene. Dette mener vi er en absolutt fordel på grunn av at det kan være mange versjoner som ligger i de historiske tabellen, og det vil derfor ta en del mer tid hvis det også skal

søkes i historiske versjoner. Dette vil si at hvis brukeren ikke har med noen temporale ord i spørreuttrykket, så vil uttrykket bli kjørt som om tabellen ikke var temporal.

14.2 Svakheter

Vi har bare programmert en enkel opp og nedkobling for brukeren. Når brukeren trykker på "connect"-knappen, blir han automatisk koblet opp mot Oracle9i databasesystemet. Han trenger ikke å skrive inn brukernavn, passord eller vertsmaskinnavnet som det skal kobles opp mot. Om programmet skulle vært brukt i praksis måtte det selvsagt være slik at en bruker logger seg opp mot databasen selv.

I kapittel 11.3 beskrev vi algoritmene for å benytte vårt temporale version-ord. Disse algoritmene er ikke implementert fullt ut. Blant annet så gjøres det bare søk i de gjeldende tabellene for å finne dokument identifikatoren. Denne brukes videre for å finne resten av versjonen som har lik dokument identifikator. Her trengs det en del forbedringer. En annen svakhet er at det kan bli problemer hvis det søkes etter versjoner fra flere ulike dokumenter. Vi fikk ikke tid til å implementere hele løsningen på dette feltet.

Hvis brukeren vil spørre i tabeller som ikke er temporale, så vil det ta lenger tid enn hvis spørringen hadde blitt kjørt gjennom SQL*Plus. Grunnen til dette er at spørreuttrykket vil først bli sendt gjennom vår SQL-parser, før det sendes til databasen. Dette vil ta opp litt tid.

Når et uttrykk blir kjørt vil det bli lagt til meldinger sammen med resultatet fra utrykket før det sendes til brukeren. Dette vil være både melding om at det har skjedd en feil, men også når alt har gått som det skulle gjøre. Det som ikke har blitt gjort i forbindelse med meldingene, er å gi feilmeldinger når det skjer en feil under vår egen parsing av uttrykket.

14.3 Utvidelser av programmet

Vi har tidligere skrevet om at det kan være flere typer brukere av programmet vårt. I vår implementasjon har vi bare tatt brukere som selv vil skrive inn sine uttrykk i betraktning. En utvidelse vil være å gjøre programmet tilgjengelig som et API for andre programmer.

For hjelpesidene våre er det muligheter for en del utvidelser. Her kan det blant annet være mulig å skrive litt mer om spørrespråket SQL som Oracle benytter. Eventuelt kan det legges inn en lenke til andre hjelpesider som Oracle selv har laget om sitt spørrespråk.

14.3.1 Utvidelser i forhold til transaksjoner

Tidsstemplet som brukes i en transaksjon, settes når transaksjonen starter. Vi valgte denne løsningen, framfor å sette den på slutten, på grunn av at det var den enkleste løsningen. Videre risikerte vi å måtte oppdatere flere tidsstempel etter alle uttrykk var kjørt, hvis vi vil sette det på slutten.

I en forbedret løsning vil det antakeligvis være bedre å sette tidsstemplet på slutten. Grunnen er jo at de siste uttrykkene som kjøres i en transaksjon, vil få et tidsstempel som er før uttrykket faktisk skjedde.

14.3.2 Utvidelser av tidsaspektet

Vårt program tar bare med dimensjonen transaksjonstid til XML-dokumentene. Det vil være mulig å utvide programmet slik at det også dekker gyldighetstid. For å gjøre dette må vi legge til enda et tidsstempel på XML-dokumentene, som da vil være for gyldighetstid. Dette tidsstemplet kan legges i samme tabell som tidsstemplet for transaksjonstid ligger. Nå har tidstabellen for den gjeldende tabellen, bare starttiden. En gjeldende versjon vil ha en udefinert sluttid, slik at sluttiden ikke trengs før versjonen blir oppdatert/slettet. Versjonen vil da bli lagt i den historiske tabellen. For gyldighetstid vil det være annerledes. Her må både starttid og sluttid registreres for de gjeldende versjonene av XML-dokumentene. Dette er fordi tidsintervallet for når en versjon er gjeldene kan settes før og etter tiden virkelig inntreffer. En versjon kan altså få sluttiden satt selv om den ligger i gjeldende tabell.

En utvidelse av vårt program kan også ta for seg tidsstempel på noder i XML-dokumentene og ikke bare på hele dokumentet. Om dette gjøres vil dokumentet kunne bli ekstrahert til flere deler. En del for hver node. En node vil i dette tilfellet være et element med innhold, flere elementer med innhold, attributter osv. Disse delene kan så få tidsstempler slik som vi har gjort for hvert dokument. En fordel ved å sette tidsstempel på noder, er at versjoner av et dokument kan benytte samme node. Om en node ikke blir forandret fra en versjon til en annen, trenger man ikke lagre den to ganger.

14.3.3 Utvidelser av spørrespråket

Parseren tar som sagt, bare for seg de enkleste delene av SQL. Det som er med i parseren er det som trengs for å lage et program som lager temporale XML-tabeller og gjøre enkel spørringer mot tabellene. Dermed vil det her være store utvidelsesmuligheter, ved å legge til de andre mulighetene som SQL tilbyr. Dette vil være greit å gjøre ved å legge dem til i vår parser, og programmere dem inn i vår program.

Det er også en del muligheter med å utvide den temporale delen av spørrespråket som vi har laget. Her har vi kommet med noen flere forslag enn de vi har implementert, i kapittel 10.3. Utvidelsesmulighetene vil kunne legges til ved at parseren utvides, og man må programmere dem inn i preprosessoren.

DEL IV

Avslutning

Etter å ha beskrevet alle deler av vår oppgave, vil vi her komme med en konklusjon av det praktiske arbeidet vårt. I kapittel 16 vil det stå en referanseliste fra hele arbeidet vårt.

Kap 15: Konklusjon

Kap 16: Referanser

15.0 Konklusjon

Vi startet oppgaven vår med et forstudium for å lære oss nødvendig bakgrunnsteori, og undersøke hva som finnes på markedet av temporale XML-databasesystemer. Under denne prosessen fikk vi også ideer til hvordan vi skulle modellere de temporale aspektene av vår oppgave.

Vi bestemte oss tidlig for å utvide Oracle9i-databasesystemet. Hovedgrunnen til dette var at Oracle9i er installert på universitetet. Vi har likevel tatt for oss hvordan andre databasesystemer har støtte for XML. Dermed fikk vi i praksis se det vi hadde lest om under forarbeidet. Ingen av de databasesystemene vi så på har noen god form for versjonering av XML-dokumenter. Dette gjelder også Oracle9i, som vi skulle utvide.

Etter å ha sett på hvordan XML-støtten var for Oracle9i-databasesystemet, begynte vi å utarbeide våre temporale utvidelser til spørrespråket. Vi valgte å bruke transaksjonstid som tidsdimensjon i vår utvidelse. Denne dimensjonen er enklere å implementere og er en god løsning i mange tilfeller. Vårt mål var at de temporale delene av spørrespråket skulle være enkle for brukeren å forstå. Samtidig skulle det utvidede spørrespråket være kompatibelt med spørrespråket som allerede brukes.

For å lage en implementasjon av dette temporale spørrespråket måtte vi også modellere hvordan XML-dokumentene skulle bli lagret i databasen. Vi fant flere alternativer på modellering av tabellene for XML-dokumentene. Deretter ville vi finne hvor stor kostnaden var for å spørre med ulike uttrykk, for de ulike alternativene. Uttrykk her vil være de uttrykk som brukeren sender til databasesystemet. Dette uttrykket vil bli gjort om til et eller flere nye uttrykk, når det spørres etter temporale aspekter eller i temporale tabeller. Disse nye uttrykkene vil variere etter hvilken modelleringsalternativ som velges. Vi valgte derfor å gjennomføre noen tester for å se på hvilken lagringsmodell som egner seg mest. Det alternativet som ble valgt var TX2, altså lagring i en gjeldende tabell og en historisk tabell.

Etter å ha funnet det beste alternativet for modellering, kunne vi implementere denne modellen. Vi gjorde dette ved å lage en preprosessor. Preprosessoren legger til tidsstempel på versjonene av XML-dokumentene, og implementerer vår utvidelse av spørrespråket SQL. For at brukeren enkelt skal kunne bruke denne preprosessoren, har vi laget et eget grensesnitt. Brukeren kan bruke dette grensesnittet til å skrive sine uttrykk og få opp resultatene av uttrykkene. For å kunne utføre en god evaluering av brukerens uttrykk, har vi laget en egen SQL-parser. Denne parseren består bare av de enkleste delene av Oracle’s SQL. Det er bare tatt med de deler av SQL, som trengs for å bruke være temporale utvidelser. Grunnen til denne forenklingen er at dette ikke var så stor del av vår oppgave og ville heller bruke tid på de temporale aspektene.

Noe av utfordringen med oppgaven var å legge et temporalt lag over et allerede eksisterende databasesystem. Det ville vært lettere om vi kunne satt datatypene selv og tilpasset dem, og ikke brukt de som allerede er definert. Likevel var det fullt mulig å implementere et overbygg på databasesystemet Oracle9i.

16.0 Referanser

[1] A. Eisenberg, J. Melton. SQL/XML and the SQLX Informal Group of Companies. ACM SIGMOD Record 30 (3): 105-108. September 2001.

[2] J. Allen. Maintaining knowledge about temporal intervals. *Comm. ACM* 26 (11): 832-843. 1983.

[3] M.A. Syafi'i, B. Searle, E.G. Masters. The Use of Object-Relational Database Management Systems (ORDBMS). The first Australian Marine and Coastal Data Management Conference. Hobart, November 1998.

[4] W3C-standard: XML Path Language (XPath) Version 1.0. November 1999. 30.01.02. http://www.w3.org/TR/xpath

[5] W3School: XPath Tutorial. 30.01.02. http://www.w3schools.com/xpath/default.asp

[6] K. Staken. Introduction to Native XML Databases. 20.02.02. http://www.xml.com./pub/a/2001/10/31/nativexmldb.html

[7] Databases. 08.05.02. http://www.cl.cam.ac.uk/Research/Security/resources/BPM/english/b/92.htm

[8] State of the Art in Database Systems. 08.05.02. http://www.cs.indiana.edu/hyplan/asengupt/thesis/oral/subsection3.2.1.html

[9] Basic concepts for using an ODBMS. 08.05.02. http://www.odbmsfacts.com/articles/basic_concepts_for_using_an_odbms.html

[10] P. Xeros. Storing XML. 08.05.02. http://www.dbnet.ece.ntua.gr/seminar/01-11-26_StoringXML_pxeros.ppt

[11] C. Babcock. Internet Insight. 08.05.02. http://www.eweek.com/article/0,3658,s=722&a=22642,00.asp

[12] K. Williams. XML for Data: An early look at XQuery. 08.05.02. http://www-106.ibm.com/developerworks/library/x-xdxqry.html

[13] Oracle9i documentation: Basic elements of Oracle SQL. 10.03.02. http://otn.oracle.com/docs/products/oracle9i/doc_library/901_doc/server.901/a90125/sql_elements2.htm

[14] K. Nørvåg. Algorithms for Temporal Query Operators in XML databases. Technical Report IDI-8/2001. Oktober 2001

[15] K. Torp, C. S. Jensen, og M. Böhlen. Layered Temporal DBMS's -Concepts and Techniques. 5th International Conference on Database Systems for Advanced Applications. April 1997.

[16] R. Land. A Brief Survey of Software Architecture. Mälardalen Real-Time Research Center, Department of Computer Engineering, Mälardalen University. Februar 2002.

[17] S. H. Chien, V. J. Tsotras, C. Zanilolo. Storing and querying Multi-version XML Documents using Durable Numbers. Computer Science Department University of California, Los Angeles. Lecture notes in Computer Science.

[18] S. H. Chien, V. J. Tsotras, C. Zaniolo. A Comparative Study of Version Management Schemes for XML Documents. TimeCenter Technical Report TR-51, september 2000.

[19] S.H. Chien, V. J. Tsotras, C. Zaniolo, D. Zhang. Efficient Complex Query Support for Multiversion XML Documents. The 8th Conference on Extending Database Technology, Prague, Czech Republic, 2002.

[20] S. H. Chien, V. J. Tsotras, C. Zaniolo. XML Document Versioning. Sigmod Records, Volume 30 Number 3, September 2001.

[21] R. Bourret. XML and Databases. Technical University of Darmstadt, 2000.

[22] Beskrivelse av ord. http://www.cs.auc.dk/~csj/Glossary/main/pages/general_db/schema_versioning.html

[23] Hjemmeside til DB2 Universal Database. http://www-3.ibm.com/software/data/db2/udb/

[24] S. Lawson, M. Hubel, G. Mordini. DB2 Universal Database Version 7, The Database Solution for e-business. The IDUG Solutions Journal, Summer 2000 - Volume 7, number 2.

[25] C. Jensen, Introduction to temporal database research. 21.01.02. http://dimlab.usc.edu/cs599-2000/paper/chapter1.pdf

[26] Elmasri, Navathe. Fundamentals of Database Systems. Third Edition. 2000.

[27] Eisenberg, Melton. SQL:1999, Formerly known as SQL3. http://geochem.gsc.nrcan.gc.ca/miscellaneous_resources/sql1999.pdf

[28] Kauffman, Spencer, Matsik. Beginning SQL Programming. Mars 2001

[29] W3C-standard: XQuery 1.0: An XML Query Language. 21.01.02. http://www.w3.org/TR/xquery/

[30] H. Katz. An introduction to XQuery. 21.01.02. http://www-106.ibm.com/developerworks/xml/library/x-xquery.html

[31] W3C-standard: XML Query Requirements. 21.01.02. http://www.w3.org/TR/xmlquery-req

[32] S. Bressan, M. L. Lee, Y. G. Li, B. Wadhwa, G. Dobbie, Z. Lacroix, U. Nambiar. X007: Applying 007 Benchmark to XML Query Processing Tools. ACM CIKM

International Conference on Information and Knowledge Management. Atlanta, Georgia, USA. November 2001.

[33] A. Bonifati, S. Ceri. Comparative analysis of five XML Query Languages. Sigmod Records, Volume 29, number 1. Mars 2000.

[34] A. Møller, M. I. Schwartzbach. XQuery. 21.01.02. http://www.brics.dk/~amoeller/XML/querying/

[35] Java Parser: JavaCC. 05.03.02. http://www.webgain.com/products/java_cc/

[36] V. Aguilera, S. Cluet, P. Veltri, D. Vodislav, F. Wattez. Querying XML Documents in Xyleme. Verso Report Number 185. 2000.

[37] L. Fegaras, R. Elmasri. A Temporal Object Query Language. 5th Workshop on Temporal Representation and Reasoning, TIME ‘98. Sansibel Islands, Florida, USA. Mai 1998.

[38] Temporal Databases. 18.03.02. http://www.geo.unizh.ch/~imfeld/diss/node24.htm

[39] W3C-standard: Extensible Markup Language (XML) 1.0 Second edition. 20.01.02. http://www.w3.org/TR/2000/REC-xml-20001006

[40] World Wide Web Consortium. http://www.w3.org/

[41] W3C-standard: HTML 4.01 Specification. 20.01.02 http://www.w3.org/TR/html4/

[42] W3C-standard: XML Schema. 20.01.02. http://www.w3.org/TR/xmlschema-0/

[43] W3C-standard: XSL Transformations (XSLT). 20.01.02. http://www.w3.org/TR/xslt

[44] Elliotte Rusty Harold, W. Scott Means. XML in a nutshell. 2001.

[45] XML:DB mailing list. 10.02.02. http://www.xmldb.org/projects.html

[46] M. Champion. Storing XML in Databases. eAI Journal. Oktober 2001

[47] G. Özsoyoglu, R. T. Snodgrass. Temporal and Real-Time Databases: A Survey. IEEE Transaction om Knowledge and Data Enginering. August 1995

[48] Hjemmeside til Tamino XML Server. 01.02.02. http://www.softwareag.com/tamino/

[49] Hjemmeside til eXtensible Information Server. 01.02.02. http://www.exceloncorp.com/products/xis/

[50] Hjemmeside til X-hive/DB. 01.02.02. http://www.x-hive.com/

[51] Hjemmeside til Microsoft SQL Server. 01.02.02. http://www.microsoft.com/sql/default.asp

[52] Ronald Bourret. XML Database products. 12.06.02. http://www.rpbourret.com/xml/XMLDatabaseProds.htm

[53] Hjemmeside til Xyleme, 21.03.02 http://www.xyleme.com/

[54] Oracle XML DB White Paper, 30.01.02, http://otn.oracle.com/tech/xml/xmldb/pdf/xmldb_92twp.pdf

[55] Oracle XML DB: Uniting XML Content and Data. 19.05.02. http://www.oracle.com/ip/deploy/database/oracle9i/collateral/xmldb_buswp.pdf

[56] Hjemmeside til Oracle9i. 15.02.02. http://www.oracle.com/

DEL V

Appendiks

Denne delen vil bestå av vårt appendiks. Her vil det først komme en ordliste for vår rapport. Deretter vil det i appendiks B være to tabeller med sammenligning av produktene vi har sett på. Appendiks C vil bestå av beskrivelser over de uttrykkene vi har testet. Deretter kommer det en beskrivelse av noen små tester vi gjorde på spørringer av XML. Til slutt vil all koden vi har implementer stå.

Appendiks A: Ordliste

Appendiks B: Sammenligning av NXD og enabled databser

Appendiks C: Native XML database produkter

Appendiks D: Test

Appendiks E: Test av XML

Appendiks F: Kode

Appendiks A: Ordliste

API: Application programming interface er et grensesnitt mellom et applikasjonsprogram og operativ systemer. Brukes av programmerere til å kommunisere med andre applikasjoner.

Black box: En abstraksjon av et utstyr eller et system hvor bare dens eksterne synlige oppførsel er betraktet, og ikke dens implementasjon eller hvordan den arbeider.

BLOB : Binary large object er en stor blokk med data lagret i en database. En BLOB har en struktur som kan bli interpretert av DBMS, men er bare kjent for sin størrelse og lokasjon.

CDATA: Character Data brukes for å sette inn tekst i XML-dokumenter som man ikke vil skal parses. Dette kan for eksempel være skripter eller programkoder.

CLOB: Character large objecter en stor blokk hvor verdien er laget av karakterdata.

Database management system (DBMS): En samling av programmer som gjør det mulig for brukeren å lage og opprettholde en database.

Databaseskjema: Den logiske og fysiske definisjonen på data elementer, fysiske karakteristikker og relasjoner til databasen.

Dokumentskjema: Den logiske og fysiske definisjonen på data elementer, fysiske karakteristikker og relasjoner til dokumentet.

DOM: Document Object Model er et plattform uavhengig grensesnitt for å få tilgang til HTML og XML dokumenter.

DTD: Document Type Definiton definerer strukturen til et XML-dokument. Den inneholder en liste over dens lovlige elementer og attributter.

Edit script: For å slippe å lagre hver versjon av et dokument kan man lage et redigeringsskript for hver versjon. Det må være lagret en hel versjon av dokumentet. For å rekonstruere en versjon bruker man da den hele versjonen og redigeringsskriptet.

Gyldighetstid (valid time): Gyldighetstiden til en fact er tiden hvor facten er sann i den modellerte realiteten.

HTTP: HyperText Transfer Protocol er en overføringsprotokoll som overfører HTML-dokumenter på WWW.

LOB: Large object er en datatype som brukes til å lagre store mengder data.

Namespace: Et namespace, eller navnerom, er en samling av navn, identifisert av en URIreferanse, som blir brukt i XML-dokumenter som elementtyper og attributtnavn. XML-navnerom er forskjellig fra navnerom tradisjonelt brukt i “computing disiplines“ i det at i XML-versjonen har en intern struktur og er ikke, i matematisk språk, ett sett.

ODBC: En standard for å få tilgang til forskjellige databasesystemer. Det er grensesnitt for Visual Basic, Visual C++, SQL.

Online transaction processing (OLTP): En type beregning med visse karakteristikker eller en en type beregning i kontrast med mer tradisjonelle batch(gruppe)-prosessering.

PCDATA: Parsed Character Data er selve innholdet mellom start- og slutt-tagger i et XML-dokument.

Primær nøkkel: En kolonne eller en samling av kolonner hvor verdien(e) identifiserer hver rad som unik. Ingen rader kan ha lik primær nøkkel.

Round trip: Betyr at man kan lagre et dokument i database, for så å kunne hente ut det samme dokumentet fra databasen.

SAX: Simple API for XML er et hendelses-drevet grensesnitt hvor parseren kaller opp en eller flere metoder når en hendelse skjer. Hendelser her kan være når parseren gjenkjenner XML-tagger, finner feil i XML-dokumentet ol.

Sekundær nøkkel: Er en eller flere kolonner i en tabell hvor verdien(e) stemmer med primær nøklen i en annen tabell.

Snapshot: Informasjon som hentes ut på et bestemt tidspunkt og som raskt gir et inntrykk av hvordan situasjonen er på dette tidspunkt.

Tagg: En tagg er en liten bit i et markup som starter med en venstre hake og slutter med en høyre hake, slik som <navn>. Den er brukt for å markere begynnelse og slutt på et element. Et tagg-par omgir det virkelige innholdet de skal formatere og betegnes start-tagg og ende-tagg. Den siste taggen i tagg-paret er lik som den første, men med en slash foran tagg-navnet. Eksempel på et tagg-par med innhold kan være <navn>Ola Nilsen</navn>. taggparet kan også være uten innhold, og skrives da som <navn />

Thin client: All funksjonaliteten som trengs kan støttes ved et minimum av ressurser på klientsiden.

Timestamp (tidsstempel): Er en tidsverdi som er assosiert med et objekt, for eksempel en attributtverdi eller en rad.

Transaksjon: En gruppe av databaseinstruksjoner som skal ses på som en enkle hendelse.

Transaksjonstid (transaction time): Transaksjonstiden til en fact er tiden når facten ble lagret i databasen.

View: I RDBMS er det en logisk tabell som er laget gjennom spesifisering av en eller flere operasjoner på en eller flere tabeller.

XML: Extensible Markup Language tillater å definere tagger som identifiserer data og tekst i et XML-dokument.

XML Schema: Brukes for å spesifisere betingelser på XML-dokumenter ved å bruke et XML-basert språk.

XLink: Brukes til å spesifisere lenker mellom XML-dokumenter. Man kan både lage enkle lenker, som HTML-lenker, og mer avanserte lenker.

XPointer: brukes av url eller uri til å identifisere seksjoner i XML-dokumenter.

XSLT: et språk for å transformere XML-dokumenter til nye dokumenter. Dette kan både være XML-dokumenter eller andre typer dokumenter, som HTML.

Appendiks B: Tabeller over produkter vi har sett på

Her kommer to tabeller med en oppsummering over egenskapene til produktene vi har sett på. Dette gjelder de native XML-databasesystemene (kapittel 7.0) og de XML-enablede (kapittel 8.0 og kapittel 9.0).

Appendiks C: Native XML Database produkter

Her kommer en liste over native XML databaser. Listen er hentet fra [52].

Product	Developer	License	DB Type	URL
4Suite, 4Suite Server	FourThought	Open Source	Object-oriented	http://4suite.org/index.xhtml
Birdstep RDM Mobile	Birdstep	Commercial	Object-oriented	http://www.birdstep.com/database_technology/rdm_mobile.php3
Centor Interaction Server	Centor Software Corp.	Commercial	Proprietary	http://www.centor.com/solutions/technology.shtml
Cerisent XQE	Cerisent	Commercial	Proprietary(?)	http://cerisent.com/products.xqe
Coherity XML Database	Coherity	Commercial	Proprietary	http://www.coherity.com/products/xml_database.html
DBDOM	K. Ari Krupnikov	Open Source	Relational	http://dbdom.sourceforge.net/
DOM-Safe	Ellipsis	Commercial	Proprietary	http://www.ellipsis.nl/content/products.htm
eXist	Wolfgang Meier	Open Source	Relational	http://exist.sourceforge.net/
eXtc	M/Gateway Developments Ltd.	Commercial	Cache	http://www.mgateway.tzo.com/
eXtensible Information Server (XIS)	eXcelon Corp.	Commercial	Object-oriented (ObjectStore). Relational and other data through Data Junction	http://www.exln.com/products/xis/
GoXML DB	XML Global	Commercial	Proprietary (Text-based)	http://www.xmlglobal.com/prod/db/index.jsp
Infonyte DB	Infonyte	Commercial	Proprietary (Model-based)	http://www.infonyte.com/en/index.html

Ipedo XML Database	Ipedo	Commercial	Proprietary	http://www.ipedo.com/html/products_xml_dat.html
Lore	Stanford University	Research	Semi-structured	http://www-db.stanford.edu/ lore/home/index.html
MindSuite XDB	Wired Minds	Commercial	Object-oriented	http://xdb.wiredminds.com/
Natix	data ex machina	Commercial	File system(?)	http://www.dataexmachina .de/natix.html
Neocore XML Management System	NeoCore	Commercial	Proprietary	http://www.neocore.com/ products/products.htm
ozone	ozone-db.org	Open Source	Object-oriented	http://ozone-db.org/ozone_main.html
Sekaiju / Yggdrasill	Media Fusion	Commercial	Proprietary	http://www.mediafusion-usa.com/
SIM (Structured Information Manager)	RMIT MDS Group	Commercial	Proprietary	http://www.simdb.com/simdb%20content%2Fabout%20SIM
Sybase ASE 12.5	Sybase	Commercial	Relational	http://www.sybase.com/products/databaseservers/ase
Tamino	Software AG	Commercial	Proprietary. Relational through ODBC.	http://www.softwareag.com/tamino/architecture.htm
Tendara Mobile XML Database	Tendara	Commercial	Proprietary	http://www.tendara.com/products/native-xml-database.html
TEXTML Server	IXIA, Inc.	Commercial	Proprietary (Text-based)	http://www.ixiasoft.com/products/textmlserver/
Virtuoso	OpenLink Software	Commercial	Proprietary. Relational through ODBC	http://www.openlinksw.com/virtuoso/

XDBM	Matthew Parry, Paul Sokolovsky	Open Source	Proprietary (Model-based)	http://sourceforge.net/projects/xdbm/
XDB	ZVON.org	Open Source	Relational (PostgreSQL only?)	http://zvon.org/index.php?nav_id=61
Xfinity Server	B-Bop Associates, Inc.	Commercial	Relational. Others through "data connectors".	http://www.b-bop.com/products_xfinity_server .htm
X-Hive/DB	X-Hive Corporation	Commercial	Object-oriented (Objectivity/DB). Relational through JDBC	http://www.x-hive.com/
Xindice	Apache Software Foundation	Open Source	Proprietary (Model-based)	http://xml.apache.org/xindice/
XYZFind Server	XYZFind Corporation	Commercial	Proprietary	http://www.interwoven. com/

Appendiks D: Testing

Her vil vi beskrive hva som vil skje ved de ulike uttrykkene som en bruker kan skrive inn. Ut i fra disse beskrivelsene har vi valgt ut hvilke uttrykk vi velger å teste.

Det vil gitt en forklaring for hver av de 3 lagringsalternativene som er beskrevet i kapittel 12.1. Disse var kort beskrevet:

- **TX1:** En tabell med gjeldende og historiske
- **TX2:** To tabeller; en for gjeldende og en for historiske
- **TX3:** To tabeller; en for gjeldende, og en for historiske og gjeldende

Under beskrivelsene vil navnet usertable bli brukt om tabellen som brukeren lager. Vi tar utgangspunkt at tabellen er temporale for alle uttrykkene.

16.1 Select

select "tid" from usertable

TX1, TX2, TX3: Må starte med å sjekke at tabellen finnes i tabellen temporal_table. Siden det spørres etter deler eller hele tidsstemplet, vil man også spørre i tabellen time_usertable, hvor tidsstemplene er lagret. Dermed må man skrive om hele select-uttrykket og legge inn navnene på tids-kolennene fra time_usertable i select-delen, legge til tabellnavnet time_usertable i from-delen og legge til en join mellom usertable og time_usertable til where-delen.
(2 select)

select * from usertable where version (last) and snr = 5

TX1: Aller først sjekkes om tabellen er temporal. For dette uttrykket skal den siste versjonen som har snr lik fem hentes ut. Da brukes hele uttrykket med unntak av version(last)-delen til å søke i usertable. Det som hentes ut er oid og t_start. Det sjekkes så hvilen starttid som er høyest, og det sendes en spørring etter versjonen med oid som har høyeste starttid.
(3 select)

TX2: Etter at det er funnet ut at tabellen er temporal, kjøres det en spørring etter versjoner som har snr lik fem til usertable og time_usertable der oid, doc_id og t_start hentes ut. Denne spørringen kjøres også på usertable_old og time_usertable_old. For hver doc_id blir det deretter sjekket hvilken versjon som har høyest t_start. Deretter blir det kjørt en spørring som henter ut alle versjoner som har høyeste t_start.
(4 select)

TX3: TX3 vil være likt TX1, foruten om at det sjekkes på usertable_old heller enn usertable, time_usertable_old heller enn time_usertable.
(3 select)

select * from usertable where version (all) and snr = 5

TX1: Aller først blir det sendt en spørring for å sjekke om usertable er en temporal tabell. Deretter sendes en spørring til usertable etter alle doc_id'ene hvor snr er lik fem. Det sendes så en spørring som henter ut alle versjoner av dokumenter med en av doc_id'ene. Disse versjonene er resultatet.
(3 select)

TX2: Først av alt må man sjekke om usertable er temproale, ved å sende en spørring til temporal_table. Videre må man ved denne løsningen hente ut rader både fra de tabellene for nye versjoner og tabeller for historiske versjoner. Dette gjøres ved å først kjøre et select-uttrykk som henter ut doc_id for de versjoner hvor snr er lik fem. Denne spørringen gjøres mot tabellen usertable og time_usertable. Tilsvarende må også gjøres mot usertable_old og time_usertable_old. Deretter sendes det en spørring som henter ut alle versjoner som har en av doc_id'ene som ble funnet i de to foregående spørringene.Denne spørringen spørres mot både usertable og usertable_old.
(5 select)

TX3: Først av alt må man sjekke om usertable er temproale, ved å sende en spørring til temporal_table. Siden alle versjonen ligger i den historiske tabellen, holder det å lete etter versjoner der. Først må man likevel finne doc_id for versjoner hvor snr er lik fem. Dette gjøres mot tabellen usertable. Dermed kan man bruke resultatet fra select-uttrykket til å hente ut alle versjoner, fra usertable, som har en av doc_id'ene fra foregående spørring.
(3 select)

select * from usertable where version (first) and snr = 5

TX1: Når version (first) benyttes betyr dette at den første versjonen av dokumentet skal returneres til brukeren. Det sendes da ut en spørring som henter ut alle versjoner med snr lik fem. Det hentes også ut t_start og doc_id. Det blir altså søkt i både usertable og time_usertable. For hver doc_id skal versjoen som har lavest starttid hentes ut. Det betyr at for hver doc_id sjekkes det hvilken rad som har lavest t_start. Etter at det er funnet en versjon for hver doc_id, sendes det en spørring til usertable hvor disse dokumentene hentes ut. Dette gjøres forøvring ved å benytte oid.
(3 select)

TX2: Det startes med å sjekke om usertable er temporal. Når dette er sjekket må man hente ut oid fra usertable hvor snr er fem, samtidig som t_start og doc_id hentes ut fra time_usertable. Dette må også gjøres for usertable_old og time_usertable_old. Programmet sammenligner så, for hver doc_id, t_start-verdien til de ulike versjonen og velger oid til de versjonene som har lavest t_start innenfor hver doc_id. Disse oid'ene brukes til å hente ut versjonene både fra usertable og usertable_old
(5 select)

TX3: Starter også her om med å sjekke om usertable er temporal. Når dette er sjekket må man hente ut oid fra usertable_old hvor snr er fem. s_time og doc_id fra time_usertable_old hvor oid er lik oid fra usertable_old hvor snr er fem, blir også hentet. Deretter blir det for hver doc_id funnet hvilken t_start-verdi som er lavest. Oid'ene til de versjoner med lavest t_start blir så hentet ut fra usertable_old. I

denne historiske tabellen vil også den siste versjonen ligge, slik at man slipper å søke i usertable også.
(3 select)

select * from usertable where version (date 'dato') and snr = 5

TX1: Nå skal den versjonen som er gjeldende på 'dato' og som har snr lik fem returneres. Etter at man har funnet ut at usertable er temporal, søkes det i usertable etter versjoner hvor snr er fem. Doc_id og t_start hentes da ut. Det sjekkes så hvilken t_start som har minst differanse til 'dato' og er lavere enn 'dato'. Dette må gjøres for hver doc_id. Det kjøres så en spørring mot usertable som henter ut alle versjoner som har oid fra sjekkingen hvor t_start har minste differanse til 'dato' og er lavere enn 'dato'.
(3 select)

TX2: Det sjekkes aller først om usertable er temporal. Videre må oid, t_start og doc_id hentes fra henholdsvis usertable og time_usertable. Dette må også gjøres for usertable_old og time_usertable_old. Til hver doc_id blir det sjekket hvilket rad som har t_start som har minste differanse til 'dato' og som samtidig er lavere enn 'dato'. For hver av doc_id'ene vil det være en versjon som er gyldig på tidspunktet 'dato'. Det hentes så ut den gyldige versjonen til hver at doc_id'ene, både fra usertable og usertable_old.
(5 select)

TX3: Aller først sjekkes det om usertable er temporal. Deretter hentes oid fra usertable_old, samtidig som t_start og doc_id hentes fra time_usertable_old. Til hver doc_id blir det sjekket hvilket rad som har t_start som har minste differanse til 'dato' og som samtidig er lavere enn 'dato'. Det sendes så en spørring mot usertable_old som henter ut versjonene.
(3 select)

select * from usertable where version (period ('dato1', 'dato2')) and snr = 5

TX1: Ved dette uttrykket skal det søkes etter versjoner som er gjeldende på og mellom 'dato1' og 'dato2'. Først sjekkes det om usertable er temporal. Deretter hentes oid fra usertable, samtidig som t_start og t_end hentes ut fra time_usertable hvor snr er fem. For hver doc_id sjekkes så hvilke versjon som er gjeldenede på 'dato1', altså det som har t_start med minste differanse og mindre enn 'dato1'. Alle versjoner som er gjeldende på 'dato1' samt versjoner som har t_start og t_end mellom 'dato1' og 'dato2' returneres.
(3 select)

TX2: Vil først sjekke om usertable er temporal. Etter dette hentes oid fra usertable hvor snr er fem, og t_start og doc_id fra time_usertable. Dette må også gjøres mot henholdsvis usertable_old og time_usertable_old. For hver doc_id sjekkes så hvilke versjon som er gjeldenede på 'dato1', altså det som har t_start med minste differanse og mindre enn 'dato1'. Alle versjoner som er gjeldende på 'dato1' samt versjoner som har t_start og t_end mellom 'dato1' og 'dato2' returneres. Det må da sjkkes i både usertable og usertable_old.
(5 select)

TX3: Aller først sjekkes det om usertable er temporal. Videre hentes oid fra usertable hvor snr er fem. Samtidig hentes t_start og doc_id fra time_usertable hvor oid er lik oid fra usertable. For hver doc_id sjekkes så hvilke versjon som er gjeldenede på 'dato1', altså det som har t_start med minste differanse og mindre enn 'dato1'. Alle versjoner som er gjeldende på 'dato1' samt versjoner som har t_start og t_end mellom 'dato1' og 'dato2' returneres.
(3 select)

16.2 Insert

insert into tablename

Ved innsetting i en tabell blir det først sjekket om tabellen er temporal. Siden den i dette tilfellet ikke er det blir deretter insert-uttrykket sendt direkte til databasesystemet slik det er.
(1 select og 1 insert)

insert into usertable

TX2: Et select-uttrykk benyttes til å finne ut om usertable er en temporal tabell, noe den i dette tillfellet er. Derfor må en rad settes inn i usertable og og en i time_usertable.
(1 select og 2 insert)

TX3: I TX3 vil det i tillegg til uttrykkene i TX2, være to insert-uttrykk til. Disse uttrykkene vil være like som de to første foruten om at de vil bli lagt inn i henholdsvis usertable_old og time_usertable_old.
(1 select og 4 insert)

16.3 Delete

delete from tablename

Her må man først sjekke om tabellen er temporal. Når den ikke er det, kan man bare sende uttrykket videre til databasesystemet.
(1 select og 1 delete)

delete from usertable

TX2: Starter med å sjekke om tabellen er temporal. For å slette en rad, må man hente ut den bestemte raden(e) fra databasesystemet. Dette gjelder både fra usertable og time_usertable. Måten å gjøre dette på er å først hente ut både XML-versjonen og oid fra usertable. Oid brukes til å hente ut de tilsvarende radene i time_usertable. Deretter må man lage insert-uttrykk fra resultatet som skal settes inn i henholdsvis usertable_old og time_usertable_old. I insert-uttrykket til time_usertable_old må man endre på t_end til å ha verdien til sysdate. Til slutt kan man slette de gitte radene fra usertable og time_usertable, ved å bruke oid som man allerede har lagret.
(3 select, 2 insert x antall rader og 2 delete)

TX3: Her vil man finne usertable i tabellen temporal_table. Videre henter man ut doid til de radene det gjelder fra tabellen usertable, ved hjelp av where-delen fra brukerens uttrykk. Siden alle de gitte radene allerede ligger i de historiske tabellene, er det nok å oppdatere de gitte radene i time_usertable_old til å ha verdien til sysdate i variablen t_end. Til slutt kan man slette de gitte radene i usertable og time_usertable.
(2 select, 1 update og 2 delete)

16.4 Update

update tablename

Her finner man ut at tablename ikke er temporal og kan dermed bare sende uttrykket videre til databasesystemet.
(1 select og 1 update)

update from usertable

TX2: Fra temporal_table vet man at usertable er temporal. For å utføre update-uttrykket må man derfor først hente ut alle de aktuelle radene fra usertable og time_usertable. Deretter må man lage insert-uttrykk for hver av disse radene, hvor man oppdaterer t_end verdien til sysdate. Disse insert-uttrykk kjøres til henholdvis usertable_old og time_usertable_old. Når dette er gjort kan man oppdatere de aktuelle radene. Da har man et update-uttrykk mot usertable som oppdaterer til det brukeren ønsker. Deretter må man ha et update-uttrykk mot time_usertable som oppdaterer t_start til sysdate og til ny oid.
(2 select, 2 insert og 2 update)

TX3: Fra temporal_tablename vet man at tabellen er temporal. Først av alt kjøres et select-uttrykk mot usertable for å finne oid. Alle de aktuelle radene ligger allerede i de historiske tabellene. Dermed må man først oppdatere de aktuelle radene som ligger i de historiske tabellene. Det eneste som trengs da er å oppdatere t_end fra uc til sysdate i time_usertable_old, ved hjelp av oid. Neste steg er å oppdatere radene i usertable og time_usertable. Da har man et update-uttrykk mot usertable som oppdaterer til det brukeren ønsker. Deretter må man ha et update-uttrykk mot time_usertable som oppdaterer t_start til sysdate. Til slutt må man lage et insert-uttrykk av de nye radene som skal settes inn i usertable_old og time_usertable_old. Dette gjøres ved å hente ut det nye XML-versjonene fra usertable, ved hjelp av den nye oid. Dataene til insert-uttrykket til time_usertable_old har vi allerede lagret i programmet.
(2 select, 3 update og 2 insert)

Appendiks E: Test av XML

I dette appendikset vil vi presentere resultater og kommentarer til resultatene fra bi-testene.

16.5 Beskrivelse av testmateriale

Tabellen vi kjører spørringer på vil inneholde ca 50000 XML-dokumenter. XML-dokumentene har som beskrevet der nokså lik lengde. Hvilke elementer de inneholder vil variere en del. Alle dokumentene er engelske.

For å få fram store nok forskjeller i tiden uttrykkene bruker, har vi valgt å kjøre spørringer på tabeller som ikke er indekserte.

Disse dokumentene er ikke versjonerte. Grunnen til dette er at det ikke er relevant for testingen om XML-dokumentene er versjonerte. Det eneste vi vil finne ut under testing er hvor lang tid de ulike uttrykkene vil ta.

Mengden testmaterialet mener vi er tilstrekkelig for å sjekke tidsbruken til uttrykkene.

16.6 Gjennomføring av testene

Til gjennomføring av testene måtte vi først få alle testdatene inn i databasesystemet. For å gjøre dette lagde vi et program som la inn dokumentene alfabetisk. Grunnen til at vi la dem inn alfabetisk var på grunn av at det var så mange XML-dokumenter. Dermed måtte vi legge inn en hvis mengde dokumenter om gangen. Samtidig som XML-dokumentet ble lagt inn, ble også tidsstemplet lagt inn i en annen tabell.

Selve testen består av ulike uttrykk som vi skal teste mot databasesystemet. For hvert uttrykk registreres tiden databasesystemet bruker fra uttrykket blir kjørt til resultatet kommer opp. Vi har laget flere uttrykk for hver av de ulike SQL-uttrykkene; select, insert, delete og update. Vi har valgt å ikke teste på uttrykkene create table og drop table, på grunn av at de er så sjeldent brukt og vil derfor ha liten relevans i testene våre.

16.7 Resultater

Tabell 6 viser hvilke uttrykk vi har valgt å bruke i testene våre.

TABELL 6. Oversikt over uttrykk som testes

Nr	Uttrykk
1	select * from xmldoc x where x.xml.extract('//author/text()').getStringVal() = 'Foto N. Afrati';
2	select * from xmldoc x where x.xml.extract('//author/text()').getStringVal() = 'Mark S. Ackerman';
3	select * from xmldoc x where x.xml.extract('//author/text()').getStringVal() = 'Oliver Aberth';
4	select * from xmldoc x where x.xml.extract('//title/text()').getStringVal() = 'Generalized Probabilistic Grammars.';
5	select * from xmldoc x where x.xml.extract('//title/text()').getStringVal() = 'Engineering and Scientific Subroutine Library Release 3 for IBM ES/3090 Vector Multiprocessors.';
6	select * from xmldoc x where x.xml.extract('//title/text()').getStringVal() = 'The Discrete 2-Center Problem.';
7	select * from xmldoc x where x.xml.extract('//title/text()').getStringVal() = 'A Worst-Case Analysis of the LZ2 Compression Algorithm.';
8	select * from xmldoc x where x.xml.extract('//title/text()').getStringVal() = 'The Theory of Joins in Relational Databases.';
9	select * from xmldoc x where x.xml.extract('//journal/text()').getStringVal() = 'TODS';
10	select * from xmldoc x where x.xml.extract('//journal/text()').getStringVal() = 'IEEE Transactions on Computers';
11	select * from xmldoc x where x.xml.extract('//journal/text()').getStringVal() = 'TKDE';
12	select * from xmldoc x where x.xml.extract('//journal/text()').getStringVal() = 'Journal of Logic, Language and Information';
13	select * from xmldoc x where x.xml.extract('//journal/text()').getStringVal() = 'Computer Graphics Forum';
14	select * from xmldoc x where x.xml.extract('//booktitle/text()').getStringVal() = 'IJCAI';
15	select * from xmldoc x where x.xml.extract('//booktitle/text()').getStringVal() = 'Data Compression Conference';

TABELL 7. Resultatet etter spørringene

Nr	Tid 1	Tid 2	Tid 3	Gjennomsnitt	Antall rader i resultatet
1	1:27:39	1:38:53	1:24:23	1:30:27	3 rader
2	1:38:02	1:48:52	1:49:42	1:45:43	3 rader
3	1:42:53	1:50:39	1:41.09	1:44:53	4 rader
4	1:20:28	1:31:55	1:23:12	1:25:12	1 rad
5	1:26:33	1:40:21	1:29:22	1:32:05	1 rad
6	1:30:09	1:41:34	1:25:42	1:32:28	1 rad
7	1:36:36	1:43:33	1:42:01	1:40:43	1 rad
8	1:30:43	1:21:48	1:20:05	1:24:12	1 rad
9	0:06:30	0:07:21	0:07:22	0:07:04	99 rader
10	0:04:31	0:04:34	0:04:04	0:04:23	792 rader
11	0:05:12	0:06:01	0:05:34	0:05:35	202 rader
12	0:20:33	0:23:34	0:20:54	0:21:41	30 rader
13	0:09:11	0:08:40	0:07:12	0:08:21	67 rader
14	0:05:17	0:05:56	0:06:25	0:05:52	435 rader
15	0:07:38	0:07:59	0:07:03	0:07:33	131 rader

Tabell 7 viser hvor lang tid uttrykkene brukte når de ble kjør. Vi har også regnet ut gjennomsnittet og tatt med hvor mange rader som ble skrevet ut. Testene ble kjør på tre forskjellige tidspunkter.

Ettersom en del av spørringene tar veldig lang tid har vi valgt å kjøre hver spørring kun tre ganger. Vi mener likevel det er tilstrekkelig til å se tendenser. Disse tendensene vil bli kommentert i diskusjon. Tabell 8 viser antall dokumenter som inneholder elementene vi har valgt å teste.

TABELL 8. Antall dokumenter som inneholder bestemt element

Element	Antall dokumenter med elementet
author	46414
title	46591
journal	18809
booktitle	27645

16.8 Diskusjon av resultatene

Vi skal nå ta for oss hva vi mener er årsaken til differanse i tiden spørringene tar. Vi vil bare gå inn på årsaker som ligger i forskjell på dokumentene.

16.8.1 Antall dokumenter som inneholder elementet

Vi ser av tabell 8 at elementene author og title er i omtrent alle dokumentene. Booktitle er i over halvparten av dokumentene, mens journal er i ca 40 %. Vi ser av tabell 7 at spørringene med author og title, altså fra 1 til og med 8, tar rundt en og en halv time.

De andre spørringene tar under 10 minutter. Siden det er en så stor forskjell på tidsbruken kan man konkludere med at antall dokumenter som inneholder et element har veldig stor betydning for hvor lang tid spørringen på elementet vil ta.

Vi kan også trekke slutningen at Oracle indekserer hvilke dokumenter som inneholder hvilke elementer. En annen grunn til at vi kan trekke denne slutningen er at når vi legger inn dokumenter, tar dem veldig mye større lagringsplass enn om de er lagret i en flatfil. Det ble lagret 67 XML-filer på 10 mb. Hvert XML-dokument er på ca 400 kb. Vi har ikke likevel klart å bekrefte dette i noe av dokumentasjonen til Oracle.

16.8.2 Størrelsen på innholdet i elementer

Det ser ut til at spørringer på elementer med lengre innhold bruker lengre tid enn spørringer på elementer med mindre innhold. Dette ser ut til å være en tendens for alle elementene vi har testet.

16.8.3 Antall like elementer i samme dokument

Det er mange dokumenter som har flere forfattere, men ingen med flere titler. Siden det er nokså likt antall med dokumenter som har disse to taggene (author og title) vil disse to kunne sammenlignes. Vi ser at gjennomsnittlig bruker spørringer på author-elementer lengre tid enn spørringer på title-elementer. Som beskrevet over vil størrelsen på innholdet i elementene ha en del å si. Innholdet i author-elementene fra tabell 6 er mindre enn innholdet i title-elementene. Dette forsterker inntrykket om at det har noe å si hvilket antall det er med samme elementer i samme dokument.

Appendiks F: Kode

Her kommer all koden vi har produsert. Både test-programmet og selve programmet.

Parser.jj	**11 sider**
Interface.java	**6 sider**
Parser.java	**35 sider**
Message.java	**5 sider**
Help.java	**1 sider**