UiB | Dept. of Information Science and Media Studies
UiTø | Dept. of Computer Science
NTNU | Dept. of Computer and Information Science
Telenor | Telenor R&I

CAIM
CONTEXT-AWARE IMAGE MANAGEMENT

Utviklingen av

CAIM Maritim Multimediadatabasen

Desember 2007

Øistein Myge

CAIM-UiB

1.0 Innledning

Denne rapporten beskriver arbeidet som ble utført under utviklingen av en maritim multimediadatabase i CAIM-prosjektet. For at denne rapporten skal være mest mulig til nytte for andre, vil rapporten beskrive problemene man møtte under utviklingen samt eventuelle løsninger på disse.

Det er gjerne viktig å merke seg at all kode som følger med dette prosjektet er kode som ble brukt i utviklingen. Det kan derfor forekomme kode som ikke virker helt, gjerne spesielt prosedyrer eller pakker fordi det under utviklingen ble gjort endringer i noen typer og tabeller. Dette er da kode som har vært til nytte for utvikleren, og har ikke lenger særlig stor verdi i prosjektet i seg selv.
For å øke nytteverdien av koden ble den dokumentert, og man har også utført korreksjoner slik at den blir kompilert.

Hovedoppgavene i prosjektet var:

1. Implementere en ny Oracle database basert på de tideligere databasene i prosjektet.
2. ETL (Extract, Transform and Load) dataene fra de andre databasene til den nye.
3. Legge til ny data i databasen og generalisere den.

1.1 Verktøy som ble brukt

For å vite hvilke verktøy som kunne brukes under utviklingen, var det viktig å få kartlagt hvilke oppgaver som måtte løses.
I begynnelsen av prosjektet ble det diskutert hva som var sentralt å ha med i databasen, og disse samtalene ble løsaktig klassifisert etter MoSCoW metoden:

Must have:

- Lagring av bilder, video, audio og dokumenter
- Sammenligning av bilder
- Søk etter bilder basert på QBE (Query-by-Example), og keywords
- Ta vare på informasjon fra de eldre kilde databasene

Should have:

- Søk etter audio og video med keywords
- Lagre informasjon om personer som er knyttet til arter, audio, video, bilde eller dokumenter
- Generalisering av databasen slik at den kan omhandle mer enn bare maritime dyr
- Et grensesnitt for å legge inn nye bilder i databasen

Could have:

- Et grensesnitt for å se bildene i databasen
- Klargjøre databasen slik at VISI-grensesnittet kan implementeres

Won't have this time but Would like in the future:

- Lagre koordinater for bilder, video og muligens også audio
- Sende databasen et bilde av et dyr via MMS, hvor databasen skal svare hvilket dyr det er og gi generell informasjon om dyret.

1.1.1 Oracle Database 10g Enterprise Edition Release 10.0.2

Nettopp fordi databasen skulle baseres på andre databaser og fordi bildebehandling var sentralt, falt det naturlig å ta i bruk den nåværende Oracle 10g databaseserver. Grunnen til at man ikke bruker Oracle 11g er at den er for tiden ikke tilgjengelig ved Universitetet i Bergen.

1.1.2 Notepad / Textedit

Grunnen til at all koden er skrevet i Notepad(Windows) og Textedit (Mac), er fordi utvikleren føler han lærer mer av dette enn ved bruk av programmer som hjelper deg med jobben.

1.1.3 SQL Developer

SQL Developer er et utviklingsverktøy som kan kjøres på forskjellige plattformer, og gjør det enklere å se gjennom databasen når den er implementert. Utvikleren brukte flere forskjellige maskiner, både med Windows og MacOS X som operativsystem, og foretrakk derfor å bruke samme program på alle maskinene.

1.1.4 Apache, Oracle Instant Client og PHP

For å gjøre det mulig å utvikle et PHP grensesnitt for opplasting og visning av bilder var det behov for forskjellige programmer. Selv om universitetet kunne sette opp en webserver med PHP, ønsket ikke utvikleren å gjøre dette siden den delen av prosjektet bare ville bli brukt under utviklingsfasen.
Bruken av disse programmene gav også utvikleren et trygt miljø til å eksperimentere, uten frykt for at noe viktig kunne bli ødelagt.[1]
For å koble seg til en Oracleserver via PHP må man enten installere en Oracleserver, eller Oracle Instant Client. Siden flere maskiner ble brukt under utviklingen og at det alt var en Oracleserver satt opp, falt valget på Oracle Instant Client.

[1] På CDen som følger med prosjektet finner du programmene og en ReadMe.txt fil som forklarer hvordan man installerer disse.

2.0 Modellering – SSM

CAIM – Multimediadatabasen er som nevnt tideligere basert på eldre databaser i prosjektet, og man hadde derfor et godt grunnlag for å utvikle en ny. Databasen er hovedsakelig basert på Lars-Jacob Hoves modell (2003), *Egenskaper ved Oracle interMedia OrdImage*.

Det dukket også opp nye krav som man kan finne under punk 1.1, som førte til at modellen fikk et større fokus på multimedia enn tideligere. Et annet punkt som også førte til forandringer i modellen, var at databasen skulle generaliseres for å kunne omhandle mer enn bare hvaler.

Slik som det vanligvis er med modellering, forandrer modellen seg etter som man kommer uti utviklingen. Disse endringene var vanligvis små, som for eksempel endre, legge til eller fjerne attributter.

2.1 Klassifisering av arter

Da de gamle innsettingene ble oversatt fra norsk til engelsk ble det oppdaget at enkelte websider var meget flinke til å registrere klassifisering av arter. Det kom ingen klar enighet om hvor viktig dette var for denne databasen, men utvikleren valgt å legge det til for sikkerhetsskyld. Det skal være sagt at man ikke hadde en ekspert på området, så man ble ofte usikker på hvilket innhold som var viktig å få med i databasen.

Her er noen sider man brukte som ressurser som viser en slik kategorisering:

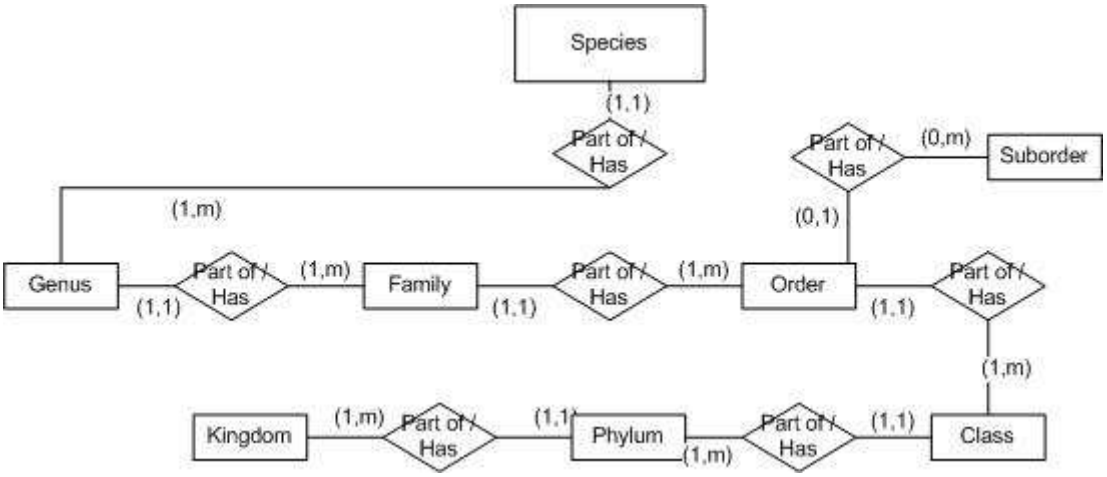
- marinebio.org
- wdcs.org til
- wikipedia.org

For eksempel så hører arten stokkand til slekten Anas som igjen hører til andefamilien (Anatidae), som igjen hører til ordenen andefugler (Anseriformes) som igjen er en del av klassen fugler (Aves), som hører til rekken Chordates som igjen hører til dyreriket (Animalia). D.v.s.

> Art -> Slekt -> Familie -> Orden -> Klasse -> Rekke -> Rike
> Species -> Genus -> Family -> Order -> Class -> Phylum -> Kingdom

En måte å ta vare på dette ville gjerne vært å bruke arv, siden det er naturlig at en and arver alle attributtene opp i hierarkiet frem til animalia. Dette ville ført til at man hadde fått et stort antall subtyper, som fører til lange insert-setninger.
Derfor valgte utvikleren å bruke en-til-mange relasjoner; et rike kan ha mange rekker, en rekke kan ha mange klasser osv.
Dette fører til lettere innsettinger, mer fleksibel koding og lettere å skrive spørringer.

Det er også noen arter som har en suborden, men siden ikke alle har det, valgte man å modellere det slik:



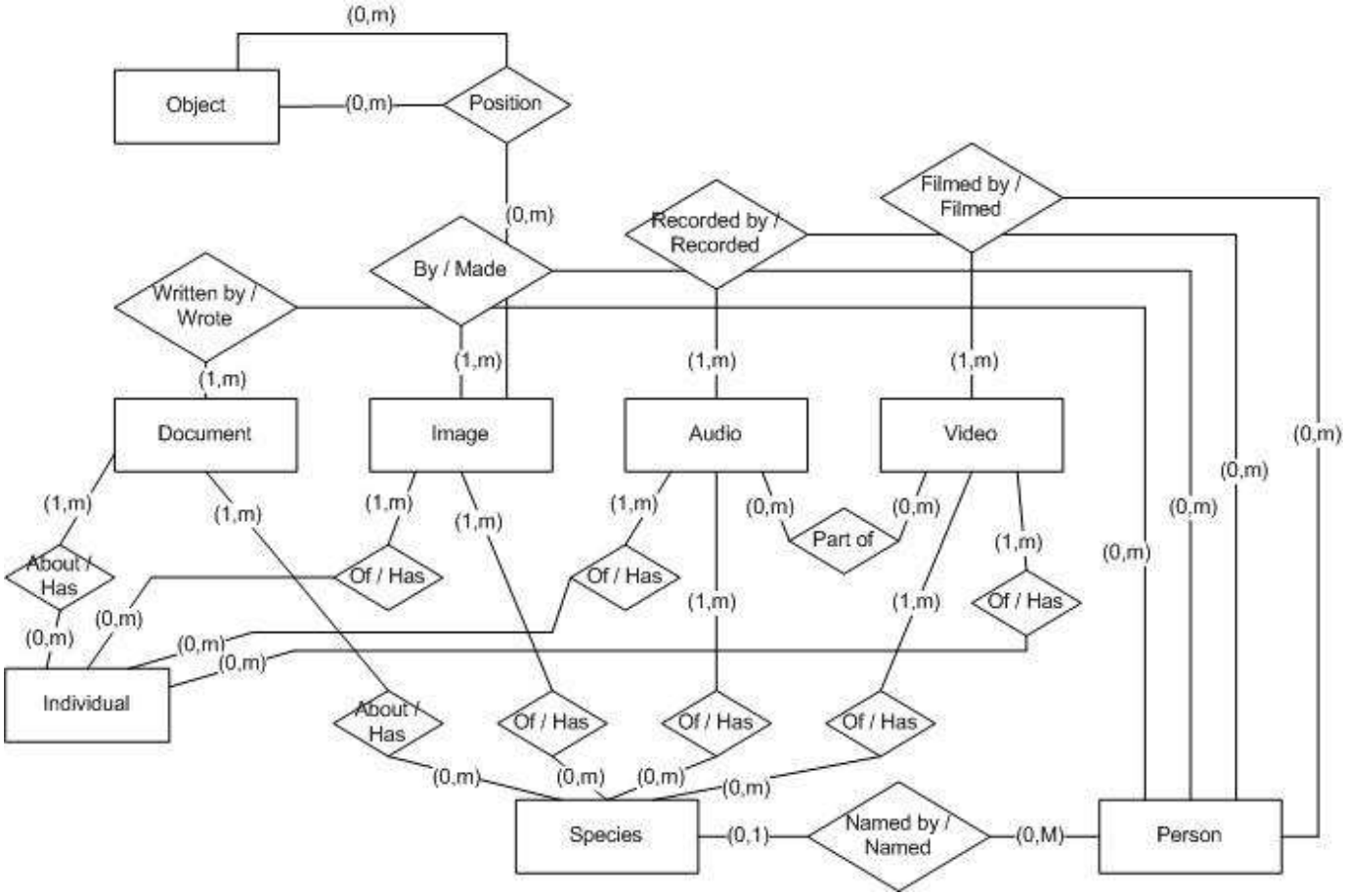
Da kan for eksempel hvaler klassifiseres som tannhval eller bardehval, mens dyr som ikke har en underorden, slipper å ha registrere noe på underorden.

2.2 Multimediaobjekter

Multimediaobjektene; Document, Image, Audio og Video kan alle bli sett på som konteinere av sitt format. Documente inneholder for eksempel ikke noe annet enn dokumentet og spesifikk informasjon om dokumentet i seg selv. Dette gjør at de objektene kan brukes over alt i databasen, noe som er grunnen til at modellen gjerne ser litt kompleks ut.
Bortsett fra to unntak har alle multimediaobjektene like relasjoner:

1. Audio og Video har i tillegg har en mange-til-mange relasjon mellom seg. Dette er fordi det kan komme tilfeller hvor man har hentet og lagret lyd fra en video, eller satt sammen et lydklipp med lyd fra flere forskjellige videoer.
2. Image og Object har en mange til mange relasjon for å ta vare på informasjon om hvor objekter i bildene står i forhold til hverandre. For eksempel hvor i et gitt bilde man kan se en fiskebåt ved siden av en hval.

Hvis man ser på Image som eksempel, så kan det eksistere mange bilder av individet Julius, og de samme bildene av Julius kan også være av arten sjimpanse. Man kan også se at det er en mange-til-mange relasjon mellom Image og Person som kan fortelle deg hvem som tok bildet.



3.0 Typer

Under utviklingen dukket det stadig opp problemstillinger tilknyttet enkelte datatyper og behovet for nye kolonner.
Hvis databasen hadde blitt kodet som en relasjonell database, ville det vært litt enklere å modifisere tabellene, hvor man har muligheten til å endre navn på selve attributtet. I Oracle 10g er det enda ikke mulig å endre navnet på et attributt i et objekt, noe som gjør det litt mer tungvindt å forandre objekter. Utvikleren har også lest at det kan være problematisk å endre datatypen til et attributt på et objekt, men har selv ikke erfart det problemet.

3.1 Objektet : Dokument

Dette objektet skal brukes for å lagre dokumenter i tilknytning til diverse arter som blir lagt inn i databasen. Da man la til dette objektet, ble det tenkt at dokumentene skulle lagres som CLOB slik at man også kan lagre tekststrenger om man ønsker det. Men da prosedyren for å legge til dokumenter ble skrevet, kom utvikleren på at Oracle 10g kan få problemer med indeksering av datatypen CLOB hvis det er filer fremfor tekststrenger som er lagt inn. Dette førte til at falt valget på BLOB for være ekstra sikker på at man ikke skulle få problemer med indeksering et dokument. Dette medfører at man ikke lenger kan sette inn tekststrenger i det attributtet, men det skal heller ikke være nødvendig.[2]

Siden det ikke forelå noen dokumenter som kunne settes inn i databasen, har man dessverre ikke lagt til rette for indeksering. Hvis det er et behov for å kunne søke i dokumentene, må man legge til lexer og indeksere tabellen.

[2] Utvikleren har erfart problemet i Oracle 10g, og løste det da ved å bruke BLOB på samme måte som her. Samme kode ved bruk av CLOB ble også testet på Oracle 9i og en annen 10g server, hvor alt virket smertefritt.

3.2 Objektet : Species

Dette objektet kan bli sett på som kjernen i databasen, og det har vært spesielt vanskelig å velge ut hvilke attributter som skal lagres her. Da de andre databasene ble studert, ble det funnet mange forskjellige attributter som kunne være interessante å ha med i databasen. Dette førte til at man valgte noen av de, men at man i tillegg lagde en nøstet tabell som heter facts, hvor man kan legge til egne attributter. Hvis det skal være effektivt å søke i FACTS, burde attributtnavnene bli generaliserte i brukergrensesnittet.
Det er selvsagt mulig å legge begrensninger i databasen for å unngå skrivefeil, men det gjør det tungvindt å legge til nye attributter som ikke har blitt brukt før.

4.0 Tabeller

Utvikleren hadde hele tiden i tankene at databasen kom til å bli brukt på et annet grensesnitt enn SQL Worksheet, høyst sannsynlig et web-grensesnitt. Grunnen til at det var viktig å tenke på var for å begrense diverse restriksjoner i databasen, men likevel forsøke å sørge for at en eventuell bruker ikke kan gjøre for store feil.

4.1 NAME – nøstet tabell

I mange tabeller blir det brukt en nøstet tabell som behandlet navn, som sørger for at man kan legge til navn i så mange språk man ønsker.
Her er en demonstrasjon som viser det første forsøket på å bruke Name i objektet Family:

```
… NESTED TABLE NAME STORE AS FAMILY_NAME_LIST((
PRIMARY KEY(NESTED_TABLE_ID,NAME))…
```

Tanken var å unngå at en bruker skulle legge inn samme navn flere ganger ved et uhell. Da denne tabellen ble testet med inndata, ble man oppmerksom på to språk som hadde samme navn på en familie, noe som ikke gikk an å legge inn. Nå var heldigvis dette ikke så vanskelig å løse, siden der er mulig å utvide en sammensatt primærnøkkel:

```
… NESTED TABLE NAME STORE AS FAMILY_NAME_LIST((
PRIMARY KEY(NESTED_TABLE_ID,NAME,LANGUAGE))…
```

Med denne endringen er det mulig at to språk har samme navn på for eksempel en familie, men man kan ikke legge inn duplikater.
Denne løsningen fører potensielt til et større problem, og det er at en bruker kan selv bestemme om norsk skal hete ”norsk”,”norwegian”,”noors”. Problemet er altså at man ikke har en generalisert standard verdi for attributtet Language.
En løsning på dette i databasenivået kan vær å bruke CHECK kommandoen når man lager tabellen. Grunnen til at CHECK ikke blir brukt er fordi det skal være opp til administratoren av web-grensesnittet å velge standardiseringen. Teknisk sett er sjansen stor for at problemet løser seg selv når et grensesnitt blir utviklet.
De som kommer til å bruke databasen, kan via web-grensesnittet lage en liste med standardverdier for det feltet, slik at en bruker ikke skriver det inn selv.

4.2 Mange-til-mange relasjoner

Databasen inneholder en del mange-til-mange relasjoner, spesielt på grunn av multimediaobjektene. For å løse dette ved hjelp av objekt-relasjonell Oracledatabase, har man i det minste to forskjellige løsninger:
Bruk av nøstet tabell:

```
CREATE TYPE IMAGE_SPECIES_TP AS OBJECT(
SPECIES REF SPECIES_TP);
/
CREATE TYPE IMAGE_SPECIES_NT AS TABLE OF
IMAGE_SPECIES_TP;
/
CREATE TYPE IMAGE_TP AS OBJECT(
…
SPECIES_RELATION IMAGE_SPECIES_NT,
…);
/
```

Denne gjør at når man setter inn et bilde, så må man i samme setning legge inn en verdi i den nøstede tabellen som peker på hvilken art bildet er av. Er det flere arter i samme bildet, kan man legge til en ny verdi i den nøstede tabellen som peker på den andre arten.
Denne løsningen gjør det litt vanskelig å holde styr på mange-til-mange relasjoner, derfor bruker denne databasen en annen løsning.

Bruk av tabell

```
CREATE TABLE IMAGE_SPECIES_TBL(
IMAGE REF IMAGE_TP,
SPECIES REF SPECIES_TP);
```

Denne metoden er svært lik den man ville brukt i en relasjonell database, bare at man bruker REF fremfor CONSTRAINT.

Men det er også et problem med denne metoden.
Det kunne vært naturlig å tenke:

```
CREATE TABLE IMAGE_SPECIES_TBL(
IMAGE REF IMAGE_TP,
SPECIES REF SPECIES_TP,
PRIMARY KEY(IMAGE,SPECIES)
CONSTRAINT IMAGE_FK FOREIGN KEY (IMAGE) REFERENCES
IMAGE_TBL(IMAGEID),
CONSTRAINT SPECIES_FK….);
```

Her har man også primærnøkkel på både image og species, som hindrer at man setter inn duplikater av samme forbindelse. Det er dessverre ikke lov å sette referanser (REF) til primærnøkler, noe som kan være en ulempe.. En løsning på dette kan være å lage en prosedyre for innsettingen, som sjekker om forbindelsen eksisterer før den settes inn.

NB! Hvis det skulle vise seg at metoden med referanser blir for vrien å håndtere, kan man bare bytte til den relasjonelle metoden.

5.0 Pakker og prosedyrer

For å forenkle prosessen om å legge til, endre og fjerne data fra databasen ble det utviklet en del prosedyrer. Under utviklingen oppdaget man at en del attributter kunne være ønskelig å manipulere i tabeller som for eksempel Species. Dette førte til at det ble skrevet en del prosedyrer, som er grunnen til at de ble lagt i pakker for å holde en bedre orden på det. Dette gjør at alle prosedyrene som har med Species å gjøre, ligger i pakken Species. Hvis man for eksempel ønsker å legge til et navn på en art på fransk kan man nå skrive: species.add_name(1,'navet på arten','fransk'); hvor tallet 1 peker på arten man ønsker å legge til navnet.

5.1 Problemstilling: Mange-til-mange

Det ble utviklet prosedyrer som setter inn mange-til-mange relasjonene mellom objektene. Tanken var at det ville være logisk å knytte de sammen ved å skrive navnene på objektene fremfor IDen deres. For eksempel at man peker på tittelen på et bilde, fremfor IDen. Problemet med dette er at databasene tillater at flere forskjellige bilder har nøyaktig den samme tittelen. Dette gjør at tittelen ikke er unik, noe som gjør at man må peke på IDen selv om den ikke er så beskrivende.

5.1.1 Forslag til løsning:

Dette kan løses ved at man ikke tillater at flere bilder har samme tittel, ved å sette tittel som primærnøkkel eller tittel som unique.
En annen løsning kan skje i grensesnittet hvor man kan velge hvordan informasjonen skal bli presentert, man kan for eksempel presentere brukeren med bildene fra databasen visuelt.

5.2 Legge til rette for VISI-prototypen

VISI-prototypen er et image retrieval system som bruker hovedsakelig objekter og tabeller som har med bilde å gjøre.

Det ble avgjort at dette prosjektet skulle legge opp til at den nye databasen skulle bli klargjort for VISI-grensesnittet. Det ble laget prosedyrer, funksjoner og nye typer for å legge det til rette slik at det eksisterende grensesnittet kunne brukes på nytt. Det eneste som mangler for å få dette til å virke, er at man konfigurerer ColdFusion serveren til å jobbe mot den nye databasen.

6.0 Utvikling av web-grensesnitt

Da dataene fra de eldre databasene ble satt inn i den nye databasen, oppdaget man at noen av innsettingene ikke stemte. Det var for eksempel noen mange-til-mange relasjon mellom bilde og art som ikke så ut til å stemme i forhold til bildetittel og navn på art. Dette førte til et behov for å kunne se bildene som allerede eksisterte i databasen, og det ble derfor bestemt at et enkelt grensesnitt måtte utvikles for å løse dette.

En utfordring var at bildene er lagret som ordimage, noe som gjør det vanskeligere enn hvis det hadde blitt lagret som BLOB. En annen ting er at det har vært få her på Universitetet i Bergen som har utviklet en løsning for dette, jeg fant bare noen som hadde klart det med ColdFusion – noe utvikleren ikke ønsket å bruke.
Valget falt derfor på PHP som er et gratis alternativ mange har kjennskap til.

NB! Kodeeksempler fra grensesnittet vil ikke bli vist her. Ønsker du å lære mer om det kan du lese den dokumenterte koden som følger med prosjektet på en CD.

6.1 Hente ut bilder lagret som ordimage med PHP

Ordimage kan deles opp slikt at man kan gi direkte adgang til elementene, inkludert den interne BLOB-datatypen. Med denne informasjonen ble det utviklet en webside hvor man skrev inn IDen på hvilket bilde man ønsket å se, som ledet til en annen webside hvor bildet ble vist. Med litt hjelp fra Siv Hansen, kom vi frem til en løsning som gjorde at bildet ble vist på samme side.

Denne løsningen krever at man legger til en funksjon i databasen som får inn en ID og returnerer bildet som BLOB sammen med mimetype. Ved hjelp av PHP kan man da sende en ID til denne funksjonen slik at man får returnert bildet man ønsker å se.

6.1.1 Alternativ løsning

Løsningen med å bruke en funksjon gjør at man er nødt til å ha rettigheter til databasen for å implementere løsningen. Dette førte til det ble utviklet en ny side for å vise bilder som ikke trengte å gå omveien gjennom funksjonen (Se på filen displayraw-alt.php).
Denne koden er ikke testet like mye som displayraw.php og er derfor ikke implementert.

6.2 Et grensesnitt som lar man bla gjennom bildene

Det første som ble prøvd ut var å bruke IDen til bildet, så plusse med 1 for neste bilde. Problemet med denne løsningen var hvis noen IDer manglet i databasen (for eksempel hvis et bilde var slettet) møtte man bare en tom side.

Det neste forsøket var å lage en spørring som rangerte IDene, og hentet ut IDen fra neste rad (ROWID). Dette fungerte fint, men det viste seg å være unødvendig å bruke en spørring for neste id og en spørring for forrige.

6.2.1 Alternativ ved bruk av PHP array

Jan Anton viste meg en svært effektiv løsning på dette, han hentet alle IDene fra bildetabellen inn i en PHP array. Deretter benyttet han nye variabler med en arrayfunksjon som heter key, slik at vi kunne hente neste og forrige bilde uten møte tomme sider dersom et bilde var blitt slettet.

6.2.2 Gjøre det slik at hvis det ikke er et neste eller forrige bilde, så forsvinner linken

Det ble gjort en del forsøk på dette, men etter at utvikleren hadde blitt introduser til PHP arrays, brukte man bare arrayfunksjonene end() og current() som peker på nåværende arrayID. Grunnen til at current() aldri bytter ID selv om du skulle trykke på neste bilde, er at funksjonene next() og prev() aldri blir brukt på arrayen. Dette fører til at pekeren alltid vil peke på den første IDen i arrayen.

6.3 Utvikling av webgrensesnitt for å sette inn bilder

Dersom bildene allerede ligger på serveren, er det ingen sak å sette inn filer. Det finnes god dokumentasjon på Oracle sine sider om det. I dette prosjekt, var det nødvendig å sette inn ca. 300 nye bilder fra ekstern filer, noe som førte til behovet etter en løsning på dette.

Et problem som oppsto, var at det så ut til at et bilde ble satt inn, men det viste seg å bare være empty_blob(). Årsaken til dette var at PHP har vært gjennom en god del utvikling, og en del kommandoer har fått nye navn. Det skal for eksempel i utgangspunktet gå an å bruke kommandoene oci_connect(), oci_new_connect() ,OCIPLogon() og kanskje noen flere, men av og til så virker det kanskje ikke. Det dukket opp noen problemer med commit, $db->commit() som ikke virket helt (hvor $db peker på login koden), så jeg måtte bytte til, oci_commit() eller ocicommit().

Siden bildene lagres som OrdImage fikk man også andre problemer som måtte løses for å ta vare på informasjon.
OrdImage lagrer mye mer informasjon om et bilde enn det en vanlig blob gjør, blant annet hvor originalfilen kommer fra, og hva den het. Det var ønskelig å vite hva filen

het før den ble satt inn i tabellen, slik at filnavnet i seg selv kunne være et stikkord for hva bildet var av. Det er to måter å gjøre dette på, bruk SetSource, eller sette inn verdiene ved å bryte opp OrdImage, hvor dette prosjektet benytter den siste løsningen. Hvis man ser på filen ”display_class.php” som finnes på CDen, er det verdt å legge merke til at man har kalt en verdi for ’CD-ROM’ i funksjonen ”insert_image()”, dette er en egendefinert verdi og ikke en standardverdi som FILE eller URL.
I Oracle dokumentasjonen står det at når bruker man FILE eller URL må man være sikker på at kildeområdet eksisterer før man setter det inn fordi man henter filen fra området man peker på.

I dette tilfellet blir filen hentet fra et HTML form, slik at det var kun for å ta vare på informasjonen om at filene ble hentet fra en CD-ROM at det ble gjort slik.
I samme setning ser du også ”:OraFileName”, som setter inn filnavnet fra PHP funksjonen $_FILES.

6.4 Om display_class.php

Denne filen inneholder unødvendig mye kode hvis alt du ønsker er å vise et bilde eller sette inn et nytt. Hvis du har tenkt å bruke denne koden til en annen database eller et annet formål, må du redigere, slette eller legge til nye funksjoner etter behov.
Da grensesnittet som ble utviklet vokste, valgte man å ta i bruk objektorientert PHP programmering slik at man skulle slippe å skrive samme kode om igjen.
Hvis du ikke er så kjent med PHP fra før så ser koden gjerne vanskeligere ut, men det er ikke så tungt å sette seg inn i. Den store fordelen med objektorientert PHP programmering er altså gjenbruk.

6.5 Spørringer i PHP

Det ble ikke tatt høyde for tabeller med NULL verdier i seg da utvikleren kjørte spørringer. Hvis du har behov for det, er det bare å legge til OCI_RETURN_NULLS:
oci_fetch_array($parse, oci_both + OCI_RETURN_NULLS)

7.0 Oppsummering

Hensikten med dette prosjektet var å skape én database med utgangspunk i de forskjellige databaseprototypene som allerede eksisterte. Det var også ønskelig å sette inn mer data, lage databasen mer generell, samt legge opp til fremtidige utvidelser. Prosjektet har resultert i en større database enn tideligere som også inneholder mer nøyaktig data, 300 nye bilder og muligheten for å inkludere andre dyr enn hvaler og delfiner.

Man har også starten på et PHP-grensesnitt som man kan bygge videre på dersom det er ønskelig å gå bort fra ColdFusion som blir brukt i VISI-prototyen.

7.1 Forslag til forbedringer

Det kan være ønskelig å få se et bilde av hvilke områder for eksempel man kan finne en gitt hval. Det kunne derfor vært en ide og inkludert et bilde i form av et kart i objektet Species, som har fargelagt områdene den gitte hvalen befinner seg i.
Siden også denne databasen er en prototype, inneholder den ikke så mye data. Det er derfor ønskelig å sette inn flere dyrearter, samt å standardisere innsettingene slik at dataene blir sammenlignbare.

8.0 Referanser:

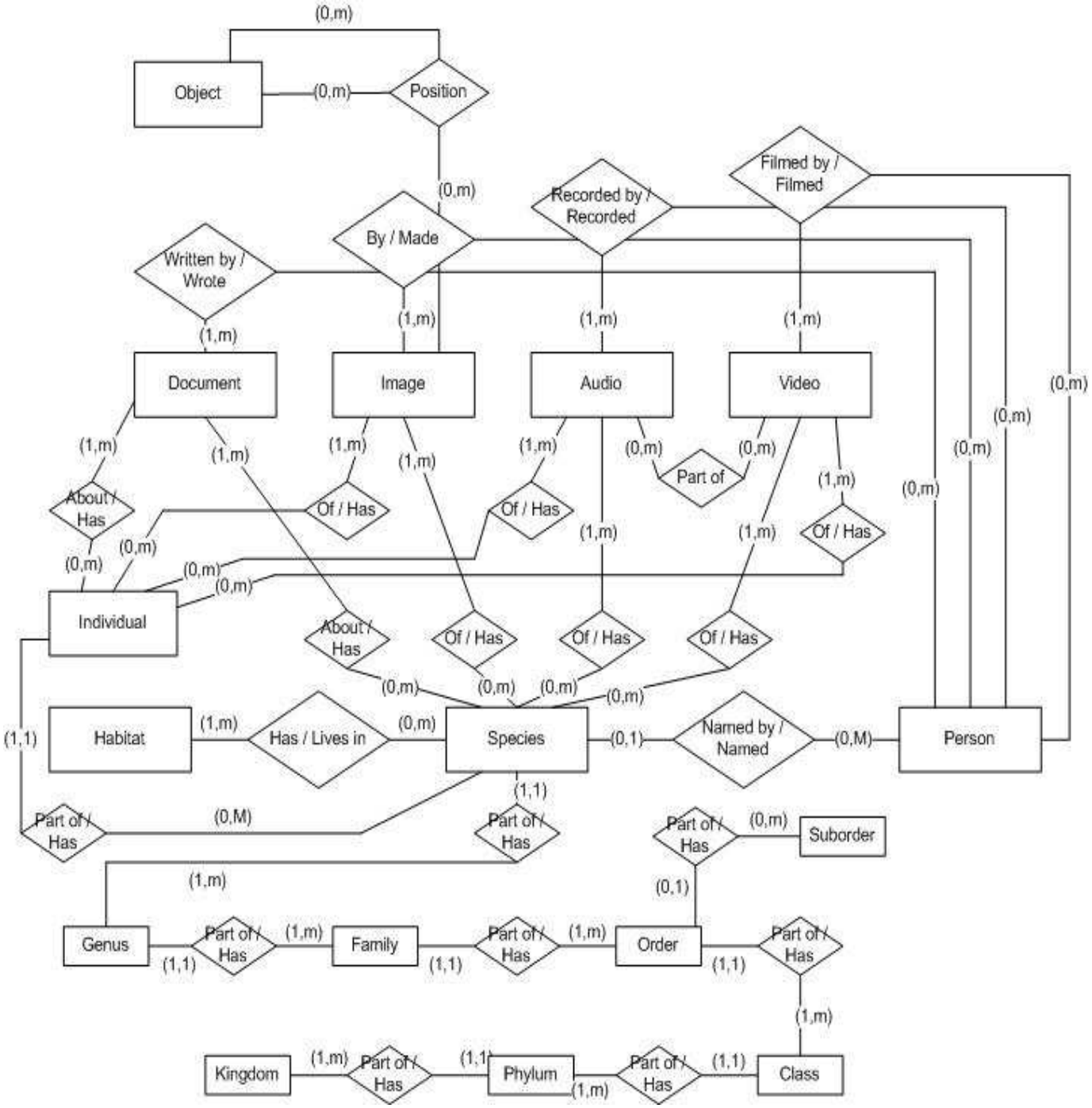
Hove, L-J. (2003) Characteristics of Oracle interMedia OrdImage. TR: IFI/VED-LJH 09-03. Dept. of Information Science, University of Bergen. In Norwegian. Available at http://nordbotten.ifi.uib.no/VirtualMuseum/Publications/VED-Report-lars.pdf.

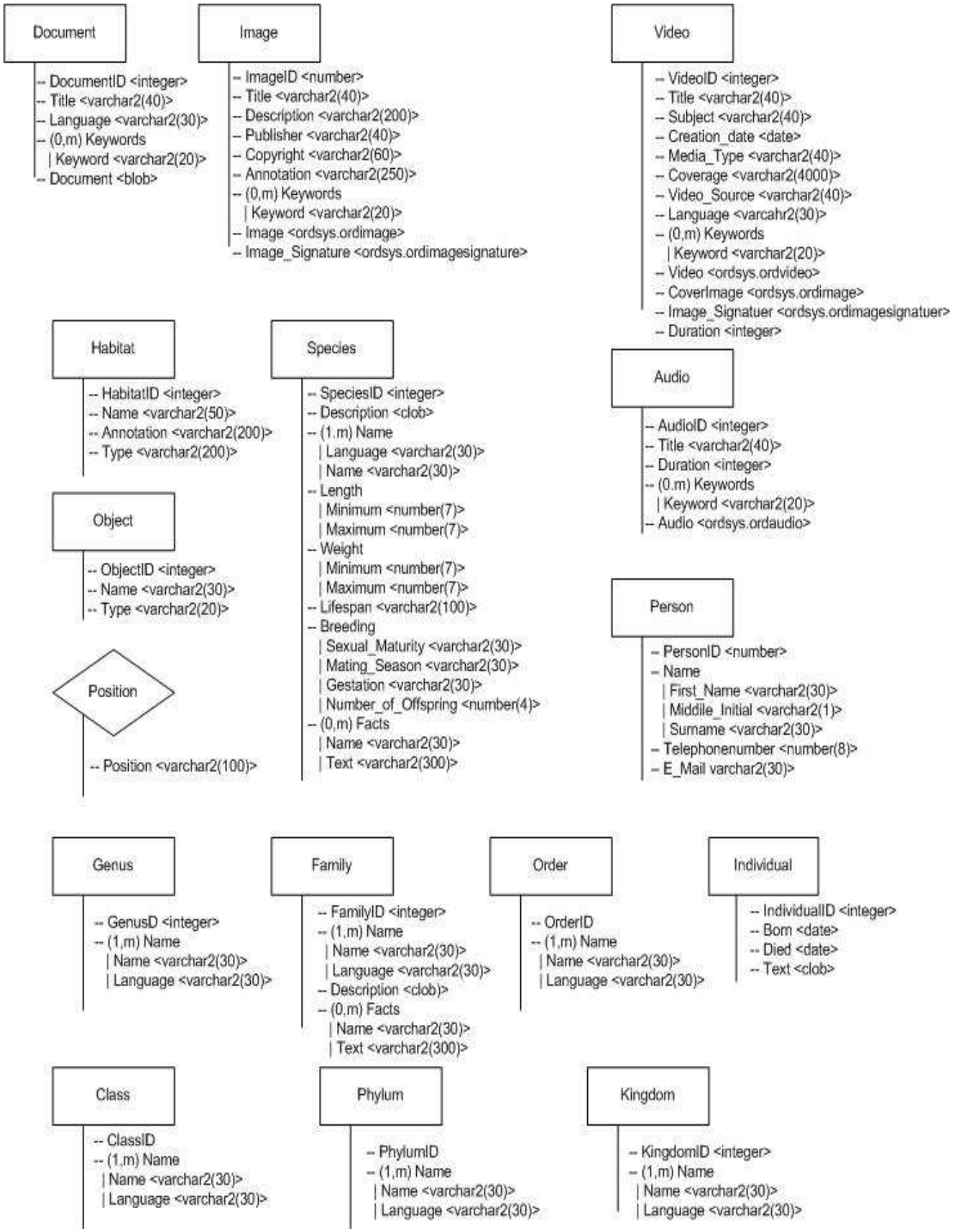
Osdal, Øyvind and Hove, L-J (2007) Visi-prototyp. Dept. of Information Sience, University of Bergen. In English. Available at http://bulmeurt.uib.no:8500/caim/VISI/

Wikipedia.org (2007) MoSCoW Method. Accessed 17.12.2007. Available at: http://en.wikipedia.org/wiki/MoSCoW_Method

9.0 Vedlegg:

SSM – CAIM – Maritim multimediadatabase




Document
-- DocumentID <integer>
-- Title <varchar2(40)>
-- Language <varchar2(30)>
-- (0,m) Keywords
| Keyword <varchar2(20)>
-- Document <blob>
Image
-- ImageID <number>
-- Title <varchar2(40)>
-- Description <varchar2(200)>
-- Publisher <varchar2(40)>
-- Copyright <varchar2(60)>
-- Annotation <varchar2(250)>
-- (0,m) Keywords
| Keyword <varchar2(20)>
-- Image <ordsys.ordimage>
-- Image_Signature <ordsys.ordimagesignature>
Video
-- VideoID <integer>
-- Title <varchar2(40)>
-- Subject <varchar2(40)>
-- Creation_date <date>
-- Media_Type <varchar2(40)>
-- Coverage <varchar2(4000)>
-- Video_Source <varchar2(40)>
-- Language <varcahr2(30)>
-- (0,m) Keywords
| Keyword <varchar2(20)>
-- Video <ordsys.ordvideo>
-- CoverImage <ordsys.ordimage>
-- Image_Signatuer <ordsys.ordimagesignatuer>
-- Duration <integer>
Habitat
-- HabitatID <integer>
-- Name <varchar2(50)>
-- Annotation <varchar2(200)>
-- Type <varchar2(200)>
Species
-- SpeciesID <integer>
-- Description <clob>
-- (1,m) Name
| Language <varchar2(30)>
| Name <varchar2(30)>
-- Length
| Minimum <number(7)>
| Maximum <number(7)>
-- Weight
| Minimum <number(7)>
| Maximum <number(7)>
-- Lifespan <varchar2(100)>
-- Breeding
| Sexual_Maturity <varchar2(30)>
| Mating_Season <varchar2(30)>
| Gestation <varchar2(30)>
| Number_of_Offspring <number(4)>
-- (0,m) Facts
| Name <varchar2(30)>
| Text <varchar2(300)>
Audio
-- AudioID <integer>
-- Title <varchar2(40)>
-- Duration <integer>
-- (0,m) Keywords
| Keyword <varchar2(20)>
-- Audio <ordsys.ordaudio>
Object
-- ObjectID <integer>
-- Name <varchar2(30)>
-- Type <varchar2(20)>
Position
-- Position <varchar2(100)>
Person
-- PersonID <number>
-- Name
| First_Name <varchar2(30)>
| Middile_Initial <varchar2(1)>
| Surname <varchar2(30)>
-- Telephonenumber <number(8)>
-- E_Mail varchar2(30)>
Genus
-- GenusD <integer>
-- (1,m) Name
| Name <varchar2(30)>
| Language <varchar2(30)>
Family
-- FamilyID <integer>
-- (1,m) Name
| Name <varchar2(30)>
| Language <varchar2(30)>
-- Description <clob>
-- (0,m) Facts
| Name <varchar2(30)>
| Text <varchar2(300)>
Order
-- OrderID
-- (1,m) Name
| Name <varchar2(30)>
| Language <varchar2(30)>
Individual
-- IndividualID <integer>
-- Born <date>
-- Died <date>
-- Text <clob>
Class
-- ClassID
-- (1,m) Name
| Name <varchar2(30)>
| Language <varchar2(30)>
Phylum
-- PhylumID
-- (1,m) Name
| Name <varchar2(30)>
| Language <varchar2(30)>
Kingdom
-- KingdomID <integer>
-- (1,m) Name
| Name <varchar2(30)>
| Language <varchar2(30)>