Marjus Nielsen
CPR: 090577-3739

Introduction

This project is based on essential parts of a database for digitalized documents that are expressed in XML. Currently 8 types of documents are digitalized, but more types will follow in the near future. The structure and semantics of the documents are defined in XML schemas and the database will only accept documents that comply with the schemas. In this project thou, XML schema validation is outside the scope.

Every document has exactly one owner. The owner is defined as the company for which the user that uploads the document works for. The owner of a document can send the document to other companies, and can give other companies read access to all documents of a certain type.

The scope of the project is to try out some of the theory on relational databases in practice. That means that some otherwise relevant topics such as exploration of the XMLType in Oracle and dynamic user control based on roles etc. is outside the scope. Instead I will focus on topics relevant for the course "Database Management Systems", in particular the conceptual design and thereafter the data model mapping to actual tables.

This course is all about databases. A database is a collection of facts, not just a collection of data. The facts can be stated as true propositions. The closed world assumption tells us that what is not explicitly stated as true propositions in the database is considered to be false [Date 2005].

> *Types are sets of things we can talk about; relations are (true) statements about those things [Date 2005].*

Requirements

As mentioned in the Introduction the goal is to create a database for XML documents.

The database shall handle XML documents from a number of users.

- Users can be added, modified and deleted.
- Documents can be inserted, modified and deleted.
- Each document has a header which uniquely identifies the document. The header contains the fields:
    - Document class: Defines what type the document is, and thereby which schema it should comply with (schemavalidation is outside the scope of this project.

    - Document number: A number identifying the document within the given document class.
    - Version: Each document can exist in several versions.

- Each document has one owner
- The owner can send specific versions of her own documents to other companies.
- The owner can grant other users rights to read a subset of the documents. For instance it should be possible to grant access to all documents with a given document class or to grant access to all versions of a specific document.
- Queries will be made that list
    - All documents that a given company owns
    - All documents that a given company has read rights to
    - All documents that a given company hass access to (own and received documents)

I will focus on creating a good logical design to maintain integrity and reduce redundancy.

Background

This project will be based on a project from my work where a database will be designed for XML documents. Compared to the real life system I will simplify the modelling of the users. In the real system users can be persons but also be groups of persons or companies. That means that there has to be a more general entity such as a principal, that acts as a user.

The structure of the document table is also more complex. Firstly there has to be one document table pr. document type, since the column for the documents is of the type XMLType and is restricted by an XML schema. Therefore the 8 differend document types generate 8 different XMLContent tables. In this project i will keep the column for the XML document itself as a simple string, because I won't need the benefits from using the XML type in this project.

Analysis

Oracle and many other DBMS's have native support for XML so that make it easy to query directly into the XML content with SQL queries combined with XQuery and XPath. Furthermore – at least in Oracle – it can be specified that the database system should split the XML document into tables for better

performance when searching through large quantities of XML documents. The automatically generated tables provide all the usual support for indexing, constraints and so on.

Users can send documents to other companies and can give others rights to read documents based on a combination of the fields of the document header. I will treat sending of documents and giving reading entitlements as basicly the same thing. Instead of creating a new copy of the document when it is sent, i will create a read entitlement for the specified company on the sent document. This means that if the owner modifies the document after it is sent, the modification will be visible for the recieiver of the sent document.

Conceptual design

This section will list and describe an ER-model for the database. An ER-model shows entities, their relationships and certain constraints such as the cardinality of the relationships, keys and participation. On the other hand the ER-diagram does not show other important constraints such as domains or constraints on values (e.g. version number must be a positive integer).

In the database design in this project not all attributes of the entities will be listed. On the other hand enough attributes should be listed to discuss alternative solutions and enable some degree of optimizing.

Users have the attributes Name, UserName and Password and the composite multi-valued attribute Address. Users work for a Company and the User and Company entities are therefore connected via the WorksFor relationship. Every user must work for a company, hence the participation of User in the WorksFor relationship is total.

Marjus Nielsen
CPR: 090577-3739

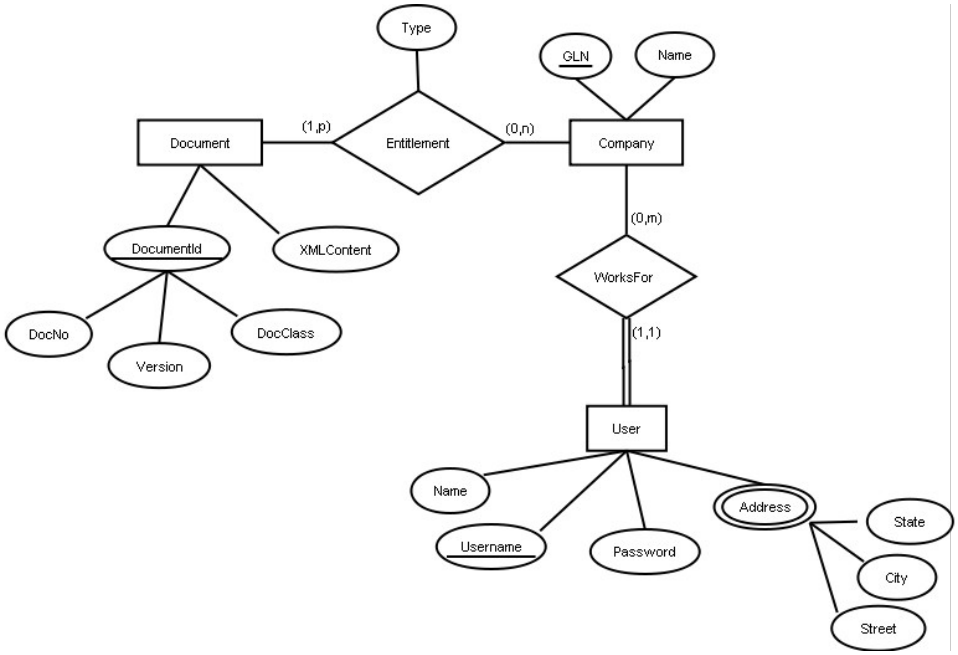

Figure 1 ER-diagram for the document database

Documents are identified by a document number, a type of document, and a version. Documents belong to a company, but the owner can grant other companies an Entitlement to read a document. Each entitlement must be of either of the types “Owner” and “Reader”. Every version of every document must have at least one owner entitlement in the Entitlement relationship.

Data model mapping

This phase of the database modelling converts the conceptual schema (ER-model) to a logical database schema by following a number of design rules. Primary keys are shown underlined and foreign keys in **bold**.

First a table is created for each non-weak entity in the ER-model.

Document(DocNo, DocClass, Version, XMLContent)

Company(GLN, Name)

User(UserName, Password, Name)

In this case we have no weak entities and no 1:1 relationships. The next step is therefore to map 1:m relationships. The WorksFor relationship can be represented by extending the User table with the primary key of Company. GLN thus becomes a foreign key in the user table:

User(UserName, Password, Name, **GLN**)

For the many-to-many Entitlement relationship the primary key of both participating entities is used as key.

Entitlement(**GLN**, **DocNo**, **DocClass**, **Version**, Type)

The last step applicable for this project is to create a table for the multivalued attribute Address of the User entity. The Address table has the primary key of the User table as a foreign key.

Address(State, City, Street, **Username**, Zip)

The tables resulting from the data model mapping are shown with foreign keys below.

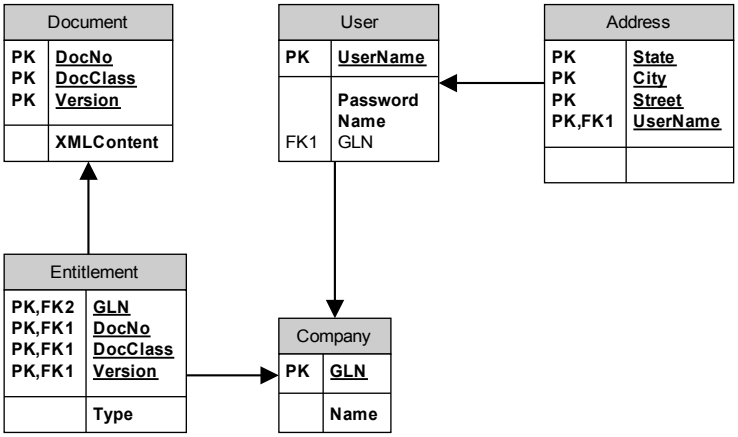

Figure 2 Database tables

Logical design

In this part I will normalize the database design to 3NF or BCNF to reduce reduntant information in the tables which could lead to update anomalies and leave the database in an inconsistent state. At the end of this chapter i will briefly discuss tasks and benefits in normalizing to higher normal forms.

An XML document in the Document table is uniquely identified by a DocNo, DocClass and a Version. The uniqueness property together with the irreducibility property make these three attributes form a candidate key for the Document table. The XMLContent is functionally dependent on DocNo, DocClass and Version.

DocNo, DocClass, Version → XMLContent

Because XMLContent is dependent on the key, and no other non-trivial functional dependencies exist in Document, it is in BCNF.

The Entitlement table is in BCNF as well, since the only non-trivial dependency is:

GLN, DocNo, DocClass, Version → Type

In the Company table we have only a key and one additional attribute. This table is also in BCNF because the only functional dependency for the Company table is:

GLN → Name

The same goes for the User table:

UserName → Password, Name, GLN

The Address table on the other hand is not in BCNF since State is functionally dependent on Zip which is not the primary key.

State, City → Zip
Zip → State

In order to normalize this table to BCNF we should decompose the table in a lossless and dependency preserving way. Lossless means that it should always be possible to join the resulting tables to get back the original table. But in this case it is not possible to decompose the attributes State, City and Zip while preserving the dependency State, City → Zip. The table is however in the less restrictive 3NF which is like BCNF but allows the dependent attribute if it is contained in a key of the table. In this case Zip → State which was a problem for BCNF is allowed in 3NF since State is part of the primary key. 3NF allows more redundancy in data than BCNF does.

BCNF isn't the highest normal form. 3NF and BCNF are concerned with functional dependencies, higher normal forms are focused on reducing redundancy by focusing on join dependencies. To cite Chris Date again:

> *The only JDs [Join Dependencies] satisfied by a 5NF relvar are ones we can't get rid of.*

Data definition language and Data manipulation language

The User table is created with the SQL code in Code snippet 1 below. The table has four columns (username, password, name and gln) and all columns but username are explicitly prohibited from using the value NULL. The username column is primary key for the table and has therefore also implicitly the NOT NULL and a unique constraint. The foreign key specification is a referential integrity constraint that requires every value for gln to exist in the Company table before it can be inserted into the User table. The datatypes used define which values are allowed for each column. Username can be any combination of up to 32 characters. The same goes for password, but for name we allow up to 64 characters. The type corresponds to the domains of the relational model.

```
CREATE TABLE user
(
  username CHAR(32),
  password CHAR(32) NOT NULL,
  name     CHAR(64) NOT NULL,
  gln      CHAR(14) NOT NULL,
  PRIMARY KEY(username),
  FOREIGN KEY(gln) REFERENCES company(gln)
)
```

Code snippet 1 Create table User

Code snippet 2 shows a table definition for the Document table with a primary key consisting of three columns. Here the combination of the three values has to be unique, and none of them can be null. This table also has a constraint that ensures that only version numbers equal or higher than 1 are used.

```
CREATE TABLE document
(
  docno      CHAR(32),
  docclass   CHAR(32),
  version    INT,
  xmlcontent VARCHAR(2000),
  PRIMARY KEY(docno, docclass, version),
  CHECK(version > 0)
)
```

Code snippet 2 Create table Document

A view is a virtual table based on the result of a SQL query. Views can be used to restrict the access users get to the database to certain predefined tables and to simplify the access, for instance the view in Code snippet 3 simplifies the access to documents belonging to companies by performing the join of documents, entitlements and companies. To use the view users can execute the query in Code snippet 5.

```
CREATE VIEW company_documents as
(
  SELECT *
  FROM document d, entitlement e, company c
  WHERE d.docno = e.docno AND d.docclass = e.docclass
     AND d.version = e.version AND e.gln = c.gln
)
```

Code snippet 3 Create view with all documents belonging to each company

Code snippet 4 shows that views are not limited to joins but can also restrict[1] just like in any SQL query.

[1] Here I use the word restrict for the σ operation of the relational algebra to avoid conflicting use of the word select. Codd's original paper on the relational model talks about restrict, and [Date 2005] consistently uses the word restrict. [Silberschatz et al. 2006] use the word select.

```
CREATE VIEW companys_own_documents as
(
  SELECT d.docno, d.docclass, d.version,
     e.type AS entitlementtype, c.gln, c.name AS cname
  FROM document d, entitlement e, company c
  WHERE d.docno = e.docno AND d.docclass = e.docclass
     AND d.version = e.version AND e.gln = c.gln
     AND e.type = 'owner'
)
```

Code snippet 4 Create view with all documents that each company owns

In the view above we restrict the query on the Type attribute of the Entitlement table. Since this attribute is not a primary key it is a candidate for an index. On the other hand this attribute will contain very few distinct values (currently only "owner" and "reader" are identified) so the benefit of an index will be limited in this case, reducing the operation from a full table scan to, on average, a half table scan.

To select from the view all documents that a certain company, say Coop, has inserted the select query would look like this:

```
SELECT docno, docclass, version, entitlementtype, gln, cname
FROM companys_own_documents where cname = 'Coop'
```

Code snippet 5 Select from view

The query corresponds to the following relational algebra (the list of project attributes is shortened):

$$\Pi_{\text{docno, docclass .., cname}}(\sigma_{\text{cname='Coop'}}(\text{companys\_own\_documents}))$$

The database then expands this query with the view so the query that gets executed by the DBMS is this:

```
SELECT * FROM
(
  SELECT d.docno, d.docclass, d.version,
     e.type AS entitlementtype, c.gln, c.name AS cname
  FROM document d, entitlement e, company c
  WHERE d.docno = e.docno AND d.docclass = e.docclass
     AND d.version = e.version AND e.gln = c.gln
     AND e.type = 'owner'
) AS temp
WHERE cname = 'Coop'
```

Code snippet 6 Query that is expanded with the view definition

The expanded query becomes the following relational algebra expression (Again the list of project attributes is shortened.

$$\Pi_{\text{docno, docclass .., cname}}(\sigma_{\text{cname='Coop'}}(\text{document} \bowtie \text{entitlement} \bowtie \text{company}))$$

Here we restrict on the Name column of the Company table (and of course still on the Type column of the Entitlement table) and no index is on the Name column. There will potentially be many companies in this table, so here an index would have a significant effect on performance. We create the index with the following statement:

```
CREATE INDEX comp_name_idx ON company(name)
```

Code snippet 7 Create index

NULL's

If not othervise specified all columns are allowed to contain null values. We saw in the create table statement for the document table that the XMLContent column does not have the `not null` constraint.

So in this case the XMLContent can contain the value null. But what does that mean? It could mean that the document exists but its body contains no elements. It could also mean that we don't know whether the body has any content or not. Another posibility is that XMLContent has content but it's confidential, and therefore not registered here.

There have been identified 29 distinct interpretations for the value NULL, and this greatly violates the principle of the database as a set of facts and it also violates the closed world assumption because when we use nulls we often don't know whether a statement evaluates to true or false, which is why we have to use three state logic, with the values true, false and unknown.

The predicate for Document would normally be:

> A document with the number DocNo, in the class DocClass, the version Version and the contents XMLContent.

Because of the null value there are several predicates for the same table:

> A document with the number DocNo, in the class DocClass, the version Version and the contents XMLContent.
>
> or
>
> A document with the number DocNo, in the class DocClass, the version Version and *unknown content*
>
> or
>
> A document with the number DocNo, in the class DocClass, the version Version and *no content*
>
> or
>
> ...

A better design instead of using nulls would be to decompose the Document table to one table that contains the document header, and two other tables, one for the XML content when it exists, and one to represent an empty document as shown in Figure 3. The XMLContent column in the DocumentContent is now not nullable. We assume that either for each row in the Document table ther exists

a row in either DocumentContent or in EmptyDocument, but not both. This should be defined in a constraint.

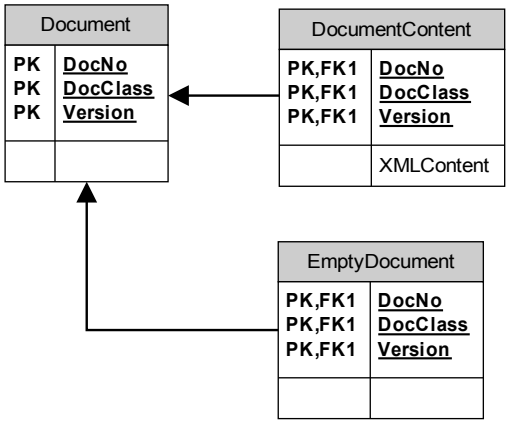

Figure 3 Decomposed Document table

With this design we get only two different predicates:

> A document with the number DocNo, in the class DocClass, the version Version and the contents XMLContent.
>
> or
>
> A document with the number DocNo, in the class DocClass, the version Version and *no content*

To sum up, we now have a much better definition of the semantics of a non existing value, by explicitly stating that the document is empty instead of using a null value that can be interpreted in many different ways.

Transactions

Often modification of the database spans multiple tables. For instance when a user inserts a document, first a record is inserted into the Document table, then a record is inserted into the Entitlement table with the key from the newly inserted document record and with the key from the company which the user works for.

Say a user who works for a company with a GLN = 33366699 inserts first version of a document with docno = 1711 and docclass = ExportDocument.

The sequence of operations would be like this:

```
INSERT INTO document (docno, docclass, version, xmlcontent)
VALUES (1711, 'ExportDocument', 1, '<Document> ...
                                   </Document>')
```

Code snippet 8 First insert into the Document table

```
INSERT INTO entitlement (gln, docno, docclass, version, type)
VALUES (33366699, 1711, 'ExportDocument', 1, 'Owner')
```

Code snippet 9 Then insert into the relationship table Entitlement

Here it is important that either both statements complete or that none of them does. Because if only the first insert statement completes we get a dangling document which has no owner. The second statement couldn't complete without the existence of the first row because the Entitlement table has a referential integrity constraint on the primary key of the Document table (foreign key). The same goes for the gln, if we provide the insert statement for the Entitlement table with a gln that does not exist in the Company table we get an error.

In order to ensure that both statements complete or that both statements fail, we use transactions.

References

[Date 2005] Date, C. J.; *Database in depth, relational theory for practitioners*. O'Reilly, 2005.

[Silberschatz et al. 2006] Silberschatz, A; Korth, Henry F.; Sudarshan, S.; *Database system concepts*, fifth edition. McGraw-Hill, 2006.