# Support for Semi-Structured Data

## DCS317 – Advanced Databases & Technology (2012)

Zygimantas RAUGAS – zr300

Artur KRZYNÓWEK – ak303

David James MAYS –djm30

**ABSTRACT**

An investigation of semi-structured data and the process of implementing semi-structured XML data into a computerised taxi firm scenario

# Table of Contents

# 1. Technology Background

## 1.1 What is XML?

XML (or Extensible Mark-up Language) is a semi-structured format for storing data. It uses tags denoted with <chevrons> to define different data values and is capable of breaking data into different tiers or a hierarchy. This is useful for storing semi-structured data about objects.XML is not designed to display data, but rather to carry it.

## 1.2 What is Semi-Structured Data?

A data structure defines the storage format and organization of specific data. Semi-structured data, therefore, is a form of data that does not necessarily conform to a formal structure. This is unlike structured data that, for example, might be held in relational or other type of database architecture. While a schema defines structured data where the order of attributes, size of elements and types of attributes are important, this is not the case in semi-structured data, where the actual number of attributes may not be known in advance. In some ways, this can be an advantage. This is why the following data structure is sometimes called "self-describing". Interestingly, the actual idea of semi-structured data actually predates XML itself.

The use of semi-structured data (like XML) has become very popular within large organisations that store large numbers of documents, due to its flexibility to move information around different software applications. Attempting the equivalent with structured data systems is usually much harder. As for XML, it is a platform-independent language; therefore moving documents is relatively easy. Another great thing about semi-structured data such as XML documents is that they can easily be modified - new attributes and elements can be added with ease in the documents.

## 1.3 Benefits of XML

The benefits of XML become immediately apparent from a software developer's perspective - many options exist for XML to be "read" and processed by various software packages and programming languages. Among other benefits, XML can be transformed into other formats, including JSON (JavaScript Object Notation), another data structure format that can be machine-read. Finally, XML is human-readable (unlike binary, serialised data and other storage methods).

From an actual data storage perspective, XML again proves to be very flexible. In addition to being a widely accessible format as already described, XML is not relational and not subject to various limitations and design restrictions when compared to relational SQL storage methods.

In an RDBMS (Relational Database Management System), a table must have a fixed number of columns, each one conforming to a certain data type. A fully normalised Database ensures minimal data redundancy and maximum data consistency. An XML document is not restricted in these ways, but can optionally be made to conform to a certain format or to respect certain restrictions by specifying an accompanying XSD file. It is even possible for XML to contain

repeating data. By contrast, the greatest flexibility RDBMS offers is the option to permit NULL data.

XML proves itself as a powerful and highly accessible mark-up language. Many people actually use XML on a daily basis without realising it – as the format used by some software applications to store configuration settings or to store user data. XML is even used by website designers and developers on a daily basis, in the form of HTML (Hypertext Mark-up Language).

## 1.4 XML Schema and Document Type Definitions

Both "Document Type Definition" (DTD) and "XML Schema" (XSD) can be used to define how a specific XML document should look like and used for document validation purposes. XSD, however, is considered to be more advanced and offer more options than DTD, such as to define data types (string, integer, date, etc.). XSD also does a better job of describing document elements and attributes, such as "complex types" and "choices" for different levels of data hierarchy and optional document contents. DTD is theoretically simple, but XSD promises to complete the same task as DTD, while also offering more options and for that reason we chose to use XML Schema (XSD) for our XML document definitions. **[5]**

## 1.5 Why The Taxi Company Might Benefit from XML

As discussed in previous sections, using semi-structured data such as XML can bring many benefits to a company. More and more companies are beginning to adopt partial or total non-relational alternatives to RDBMS set-ups, including but not limited to "NoSQL" solutions. Using XML semi-structured documents is one step towards this.

Using XML can make the data held by the company (which can go into millions of records) independent of any software package and thus allowing the company to manipulate data with different software technologies freely with no fear of that it may become incompatible within different or new environments. To some extent, this makes the data format "future proof". Thus, if the Taxi Company were to overhaul its system or make significant changes to its software environment–assuming the documents and records were stored in XML format, there would be no problems data migrating between the old and new systems. Another great thing about XML is that it requires no API in order to read the data (unlike MySQL, Oracle DB, etc.) which again can make it easy for the company to upgrade its software in the future as long as it can read simple files.

XML may also be particularly useful for the Taxi Company, if for example, it was to adopt the use of mobile or portable technology to allow drivers to process bookings, while at the same time allowing the company and customers themselves to track their taxi cars. Because there are many different brands and types of hand held devices available, many of them have their own implementation of storing data and thus ensuring compatibility with all of them may be nearly impossible. However if XML were to be adopted (being a self-describing data storage format), it would ensure compatibility with all the mobile devices.

Another benefit of using XML may become apparent if the company was to create an online booking system and/or to creating a mobile app which would allow multiple services for the customers such as booking, checking their taxi position and cancelling bookings. If the data was

to be stored in XML documents, it could easily be transformed and be represented in any type of format (website, application, etc.) without the need to recreate it for these separate purposes. This enables data and formatting rules how the data is displayed to be completely separated thus allowing different devices to have own rules of representation.

Another great thing is that there is no software required to store XML documents, thus could save cost of DB administration software if the company was to completely store everything in XML documents. XML is "not just free of charge but free of legal encumbrances"**[3]**

# 2. Data Design

## 2.1 Selecting Documents

It is essential to select the right document for conversion to semi-structured data as some may not work well being semi-structured where as other type of entries might be more suited for a semi-structured format.

It is also important to analyse the Taxi scenario in order to identify these documents and the following section discusses the reasons for converting or not converting the current tables into XML documents.

## 2.2 Considering Conversion of Taxi Schema Tables to XML

### 2.2.1 Table: Cars

This table would work rather well for conversion to semi-structured XML format document. However since most of the information about cars is practically the same and all of entries have the same number of attributes and there is no need for anything to be added to describe the car it seems as the following table is rather pointless to be converted to XML document type.

### 2.2.2 Table: CarStatus

The following table is not suitable for converting to XML document because it only consists of two attributes. It would only make sense to merge it with Cars table if that table was to be converted into XML document, however since all cars have the same number of attributes to describe them there is no need to convert it into an XML document.

### 2.2.3 Table: ShiftDays

This table would work very well in semi-structured XML format. Because currently the shift is described as days that are worked on and days that are not the table has to store an entry of "Y" meaning Yes and "N" meaning no for each day of the week. In a way this is data redundancy as we only need to know which days the employee works on, we don't care about the days he doesn't. Converting this table into an XML document would allow only storing the days that that employees work on that particular shift avoiding the need to store the days they do not work on.

### 2.2.4 Table: Employees

This is a very good candidate table to be converted into XML document. Converting this table into an XML document would add great share of flexibility of adding additional information about employees at any time (e.g. if Taxi company was to store nation insurance number, or if the company was to expand it could also store department number that employee works in).

### 2.2.5 Table: Rate

This table defines employees rate so the only time that it would make sense to convert this table into an XML document was if Employees table was to be converted into an XML document, then

the following table could be merged in that Employees XML document where each employee would have their own rate defined.

### 2.2.6 Table: Drivers& Operators

These tables only store ID's of operator or a driver, which then point to an employee record. There is no point if converting these tables to XML document.(Employee record could describe whether an employee is a driver or an operator however it's better to be kept as it is if the XML documents were to be integrated with current relational database, thus leaving referential integrity rules intact)

### 2.2.7 Table: Clients

This is a good candidate table to be converted into XML, due to the fact that if the company wanted to store more information on their Clients such as payment contracts and other information they could easily add this information without the need to change the database.

### 2.2.8 Table: Bookings, Normal Bookings, Regular Bookings & ClientBookings

At the moment to allow the ability of regular bookings, and normal bookings they have to be split into different tables to accompany the need of settings which days of week and time intervals that regular bookings are made on. It also meant that ClientBookings table was needed in order to ensure that only clients make regular bookings and it also was a way of linking particular bookings to a client. The merge of these tables into one XML document would be a great idea. Because XML is semi-structured it would allow a booking document to either represent a regular booking or a normal booking.

### 2.2.9 Table: Payments

Due to the obvious sensitive nature of payment data, it was decided that sacrificing data consistency in favour of a semi-structured data format would be unwise. Data fields concerning payments handled by a business is relatively static, so the requirement to add additional data elements in the future is unlikely.

### 2.2.10 Table: PaidSalaries

The PaidSalaries table design stores a limited amount of data – for each salary paid, it stores the employee's ID, the date, and the amount paid. Converting a table of this size into XML would only eliminate one field per record, so would make little sense.

### 2.2.11 Table: Shares

The following table represents the share that the company and a driver get from a specific payment if a driver is on a shared rate. This table only stores four attributes of which two are ID's and the other two are company and driver share therefore there is no need to convert this table into an XML document due to the fact that every recorded share will have a fixed number of attributes and there is no need to store them semi-structured data.

### 2.2.12 Table: Revenue

The Revenue table is a good example of a table that could be made semi-structured. While the data stored is important for the administration of the business, coherence is not mission-critical for day-to-day operations. Being able to add supplemental values in the future would prove beneficial for whatever records or data the company might wish to include in the future.

### 2.2.13 Table: Expenses

The Expenses table is very similar in function to the Revenue table – making the expenses table a semi-structured data type would not be detrimental to the software setup. From an accounting perspective, combining revenue and expenses data into a single XML type would also be a plausible option.

## 2.3 Tables Selected for Conversion to XML

After some review of the current tables and records that the Taxi company database holds we then selected the following candidate records to be converted into semi-structured XML data format as documents:

### 2.3.1 Employee Records

Converting this table into XML document is a good step. Employee record documents are one of the most common documents that you could find in any company thus storing them as documents is the right thing to do. This also enables to add additional information about an employee in the future without having to change the structure of the database due to the semi-structured properties that XML data storage can offer.

### 2.3.2 Bookings

Booking records were converted to XML; because of the multiple types of booking that exist. The taxi firm can book and complete one-off bookings for customers that only need their service once, or at irregular times. The form can also make agreements with clients for repeat or "regular" bookings, that may occur weekly or at some other interval. Because the two different types of booking require different data to be stored, bookings serve as a great demonstration for semi-structured data storage.

### 2.3.3 Expense Records

The Expenses was switched to semi-structured XML format, due to the variable amount of data that expense records may contain. Expenses may include optional or conditional elements such as those associated with car maintenance. As they are financial/economical records for the business, they are not required for the basic operation of the taxi service, but exist as a benefit for financial and statistical administration of the business. Therefore, a lack of coherence of relational consistency does not bring risk to the business.

### 2.3.4 Client Records

Client records were converted to document format, because of the flexibility that a semi-structured storage format can provide. The data held for different clients could vary considerably, such as a client or company that might have multiple forms of contact, or separate business and billing addresses. Miscellaneous data such as operator notes could also be stored.

## 2.4 How We Chose to Represent/Integrate the Records

Because our original design of the database was relational and stores no semi-structured type of data, a good integration of Semi-Structured XML document together with relational parts of the database was a key issue to consider here.

In order to integrate our XML documents into the database they had to be stored as XMLType. XMLType is a new data type introduced by oracle in order to allow "native handling of XML data in the database" [4].Now this data type allows to create new XMLType tables as well as XMlType columns (or attributed).

The approach we had chosen was to store our XML documents as columns (attributes) in the new updated tables where a table consists mainly of two columns altogether as so:

**Table (DocumentID, DocumentInfo(XML Document) )**

This way we could preserve the referential integrity other tables relied on by still keeping for e.g EmployeeID as a number column outside of XML document thus original Drivers and Operators relations are unaffected and thus other relations which referred to these tables in terms of primary keys are not affected either. This design allows minimum change in order to integrate both relation and the document based database.

Now if we were to integrate for e.g. the ID of employee within the XML document this would cause allot of cascading problems with referential integrity in other tables and thus would have to be modified.

Also following such design ensures referential integrity rules that are held as part of primary – foreign key relationship unlike if the ID was to be held within the XML document.

## 2.5 Revised ER Diagram



The following diagram below is the revised ER diagram of the Taxi Company scenario when the XML elements were implemented. As it can be seen the RegularBookings, NormalBookings and ClientBookings table do no longer exist. This is because all of these tables were merge due to the flexibility of XML allowing holding semi-structured data, meaning that a document can either hold a normal booking or a regular booking which can also list on which days to repeat the bookings.

As it can be seen our design nicely integrates together with previous tables without much of changes needed. The Drivers and Operators tables are still kept in order to easily distinguish drivers from operators without having to query the large XML documents and because Booking entry both requires Operators ID who made the booking and Drivers id who will take the booking, leaving both tables enabled a much better design where an Operator who made the booking or the driver that carried out the booking can be easily queried.

Expenses table was merged together with CarMaintenance which is a sub-class of Expenses. Again because XML allows semi-structured data thus an expense could easily be described as of type car maintenance by easily adding additional attributes and information within the XML document.

Rate table was removed too, this is because employee XML record can now describe the type of rate that employee is on. Even though issues could arise where Operators may be set on shared rate, but this could easily be fixed with a trigger before insert into Employees relation (we did not implement such a trigger, as it was not required). Even though the Shifts table could also be embedded within the Employees XML document we decided to keep it as relational because it seemed that allot of employees may be on the same shift and therefore there would be allot of repetition of data within XML documents (in some way this could also apply to Rate relation but we just wanted to demonstrate the power of semi-structured XML documents).

Clients table has also been changed. Instead of storing client information in columns now it's stored in one XML document where each document describes only one client. This way it enables the company to add more information about clients as required without having to change the structure of the database.

As it can be seen Addresses table is now removed completely. This is because the relations that were using a reference to an address have now been converted into XML documents where each document can now store its own address (e.g. Booking document now stores PickUpAddress and DesinationAddress as part of the document).

## 2.6 Designs of Converted XML Documents:

### 2.6.1 Booking Record:
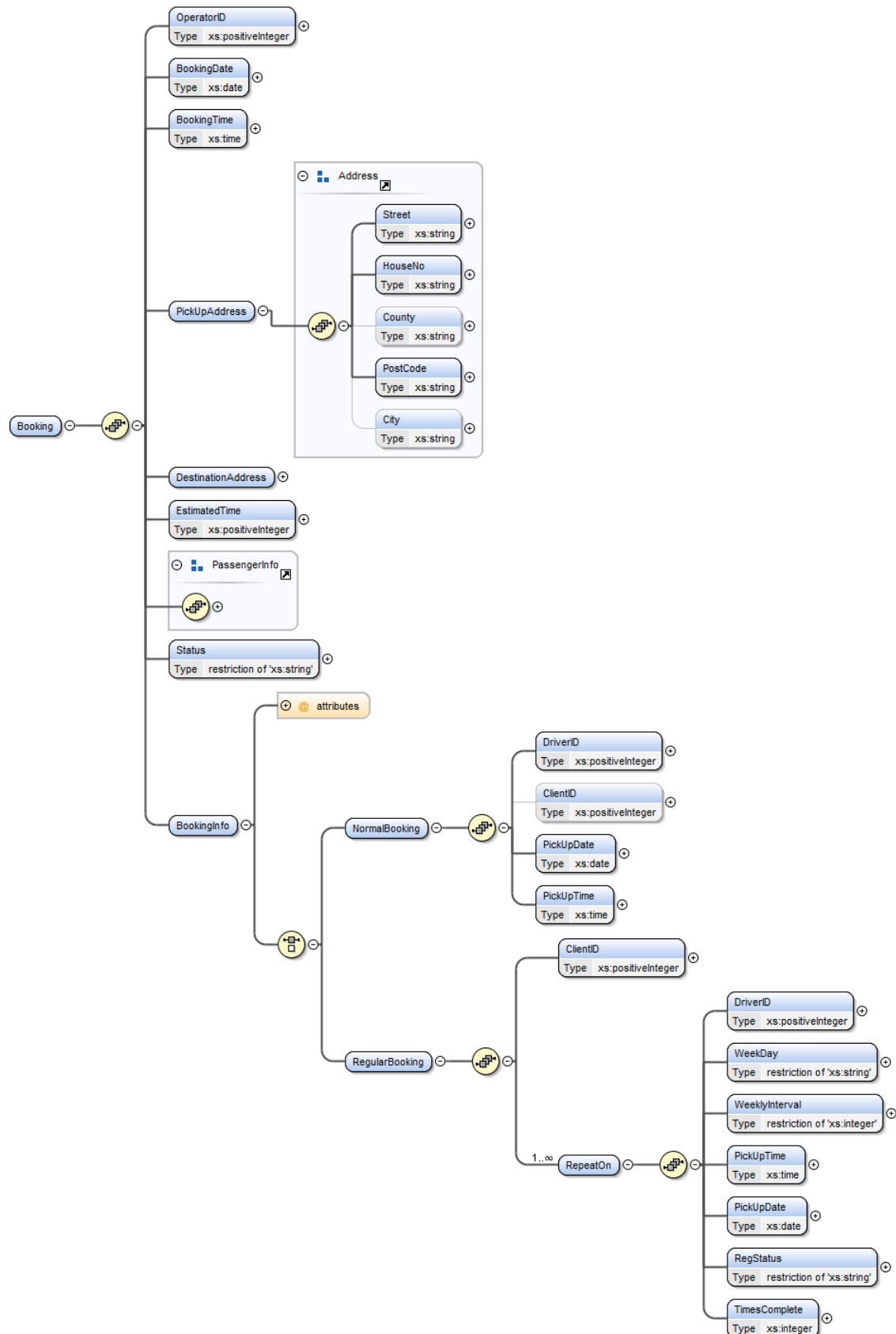
**Description:**

The following XML schema describes how a Booking document is now stored within the database. The new relation looks as following: ***Bookings (BookingID ,BookingInfo)***

As mentioned in previous sections of the report, the Booking document is now a result of four tables being merged together which were Bookings, RegularBookings, NormalBookings and ClientBookings. Because XML allows semi-structured data the information that was held in these tables can now be part of one document depending on the type of booking.

Node Booking is the parent node of this document. All types of booking have the following elements in common that MUST be in the document:

- OperatorID (who took the booking)
- BookingDate (date when booking was made)
- BookingTime (time that booking was made)
- PickUp and Desinationaddresses (where from and where to)
- Estimated time (time how long the driving from A to B should take,
- Passenger information (such a name and phone number)
- Status (is the booking cancelled or not)

However this is where the differences between Normal Booking and Regular Booking come in. Previously, three additional tables represented these differences: NormalBookings, RegularBookings and ClientBookings, where NormalBookings held all the normal bookings and RegularBookings described regular bookings. In order to make sure that only clients could make regular bookings, we had ClientBookings where BookingID and ClientID were linked together and if there was no entry of Booking ID from RegularBookings in ClientBookings, then a RegularBooking could not be made as it is not linked to a client. Since XML is very good at storing semi-structured data, the previous solution of four different tables could be replaced by just one document entry in Bookings table represented as an XML document.

Now, additionally to the information that a booking document holds as outlined above, the booking has a node called <BookingInfo>. This node is responsible for either describing a Normal Booking or a Regular Booking, but not both at the same time (achieved by choice clause in XML schema XSD). The child nodes of this node can either be <RegularBooking> or <NormalBooking>, but as already mentioned, not both at the same time. This is allowed by an XSD <choice> element.

If the document describes a normal booking then it consist of the following elements within the <NormalBooking> node:

- ClientID (Optional)
- DriverID (who will take the booking)
- PickUpDate (When will the passenger be picked up)
- PickUpTime( The exact time when the passenger will be picked up)

(Please note that date and time had to be split into two elements because of limitations of how XSD can validate date and time)

If the document describes a RegularBooking, then the <RegularBooking> node must contain these child nodes:

- ClientID (Client who made the booking)
- RepeatOn (This node describes when the booking should be repeated)

The RepeatOn node is the node that describes when this booking is repeated (day, how many weeks should pass, what time of day etc.):

- DriverID (who will take the booking)
- WeekDay (weekday that booking is repeated on)
- WeeklyInterval (weekly interval that booking is repeated)
- PickUpTime (time that driver the will pick up the passenger)
- PickUpDate (date that the driver will pick up the passenger)
- RegStatus (status of the regular booking)
- TimesComplete (how many times has it been completed)

Note that the <RegularBooking> node can have many <RepeatOn> child nodes describing on which days and weekly intervals the booking should be made, thus allowing a great deal of regular booking flexibility.

## 2.6.2 Employee Record:
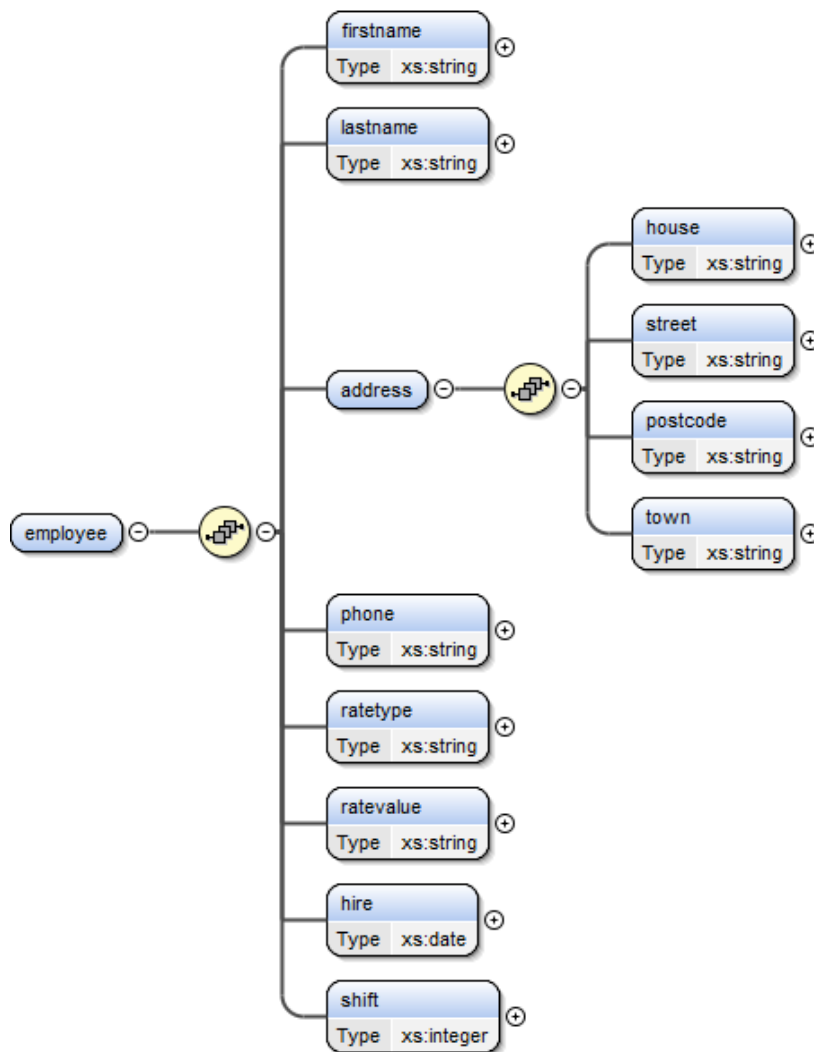


**Description:**

The Employees data structure contains two distinct hierarchy levels. The top level contains specific information for each employee - their name, phone, hire date and their shift and salary information, each represented by appropriately named nodes.

There is also an address node (denoted by the <address> node) that contains address information in a second level - the employee's house name or number, street, town and postcode. All but two values store string data types. A date type represents the employee's hire date, and an integer type represents their shift, because this is relational to a shift record elsewhere in the database.

These documents are now stored in the database as **Employees (EmployeeID, EmployeeInfo)**, with the "EmployeeInfo" being the XML document as described above.

### 2.6.3 Client Record:
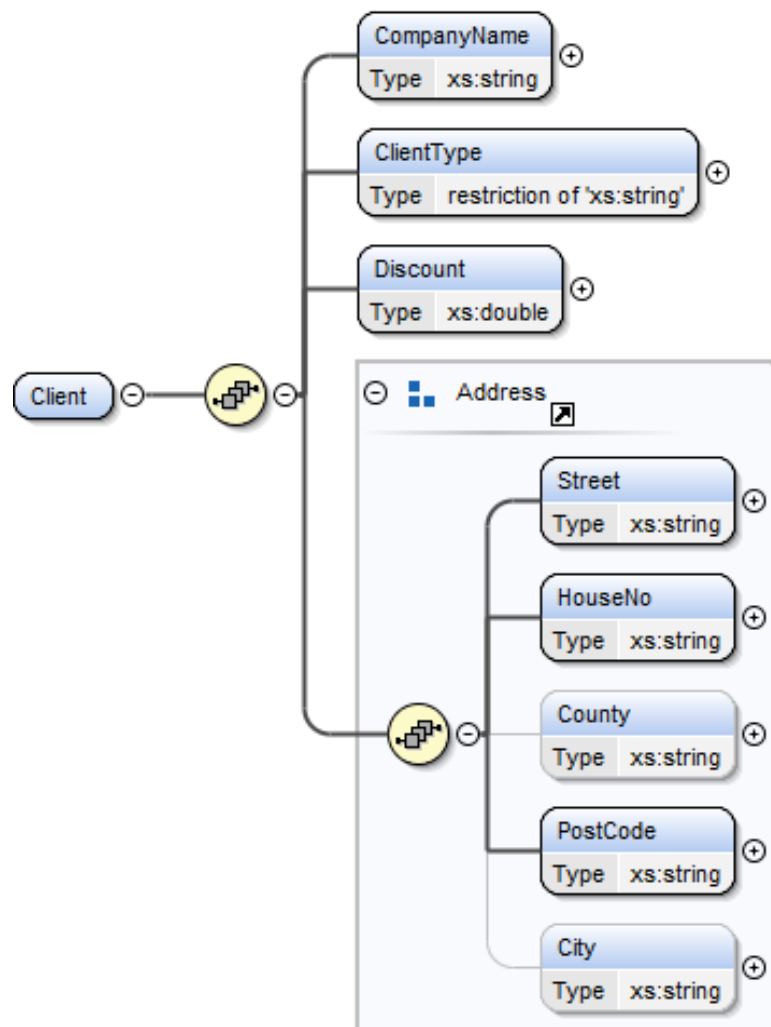


**Description:**

The following XML schema describes how a Client document is now stored within the database. The new relation looks as following: ***Clients (ClientID, ClientInfo)***

ClientID represents a primary key, which represents an ID of a client. ClientInfo is the XMLType attribute, which holds the XML document that describes a Client.

As it can be seen in the schema, <Client> is the parent node of the document. The node <CompanyName> holds the information on the client's company name. The node <ClientType> describes the type of a client it is and is restricted (using XML schema restrictions) to only hold either "Private" or "Corporate" as the value of the node. Node <Discount> holds a number of type "double", which represents the discount that a client gets over their bookings.

Address node is a group of elements which describe the address of a client with node <City> being an optional node (does not have to be within the document for a document to be valid).

## 2.6.4 Expense Record:
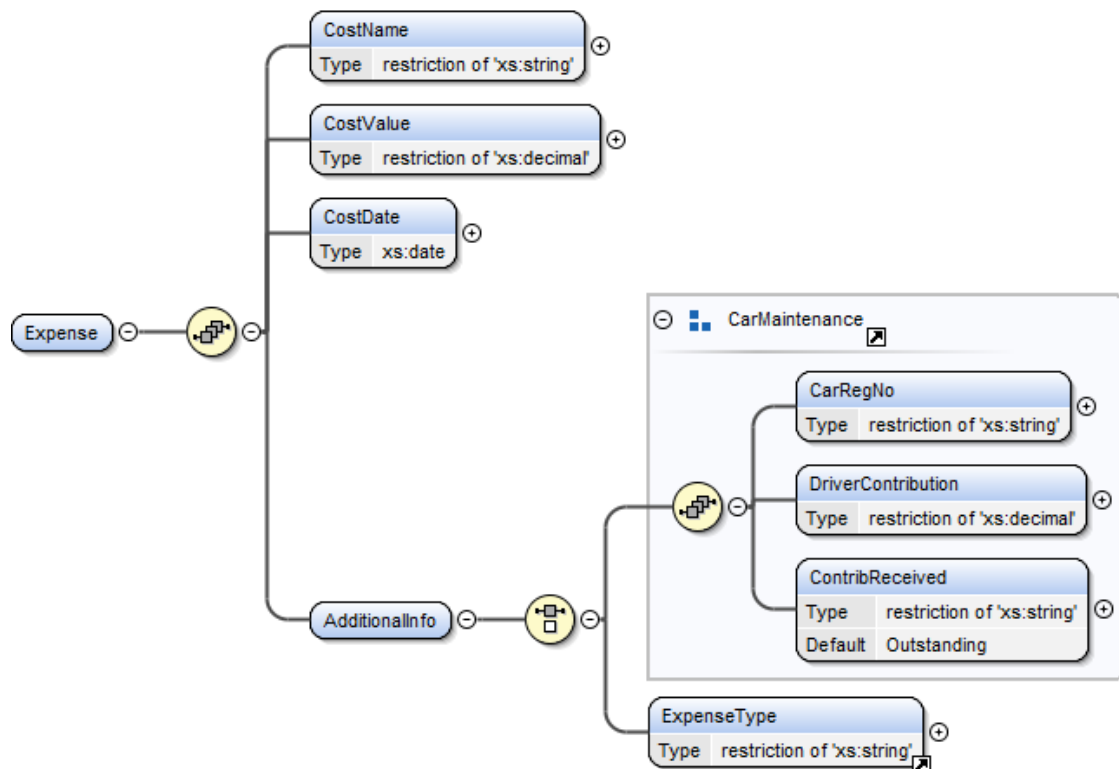


**Description:**

Previously there were two tables, one for general expense and one for car maintenance which contained additional info for when an expense was related to a car. Since XML is really good at storing semi-structured data, the two tables are now combined into one document called **ExpenseInfo.** The table that contains the document is called Expenses and holds an ID of a document and its corresponding document - **Expenses (ExpenseID, ExpenseInfo).**

The document has few elements that MUST always be present in any expense document and also few elements that are choices. The elements that must always be present are:

- Expense - the root node of the document
- CostName - name of an expense
- CostVaue - cost of an expense
- CostDate - date of an expense
- AdditionalInfo - additional information about an expense which can vary depending on the expense's type.

The "AdditionalInfo" node contains a choice for either car maintenance or other expense types such as utility. This is because whenever it is car maintenance, it should be clearly distinguished among any other expenses - it is important for the company to make sure that a driver has paid his contribution towards any maintenance of a car that he is driving.

The additional info can contain the following elements:

- ExpenseType - this describes any other expense type e.g. utility

...and for car maintenance expenses:

- CarRegNo - it is necessary to identify the driver responsible for the contributing towards the expense.
- DriverContribution - this shows how much the driver needs to pay towards the expense
- Contribreceived - this indicates whether a driver has paid his contribution or if it is outstanding.

## 2.7 Sample XML Documents

Please note that sample documents are provided in separate files (they were not included into this report in order to make it more readable). These documents can be found at the following directories within the zip file:

| XML/Employees | Contains sample XML files for employee inserts |
|---|---|
| XML/Bookings | Contains sample XML files for booking inserts |
| XML/Clients | Contains sample XML files for clients inserts |
| XML/Expenses | Contains sample XML files for expenses inserts |

Also please note that the same documents can be found as prepared SQL insert tables in the following files:

| Inserts/EmployeeInserts.sql | Contains prepared SQL inserts for XML employee records |
|---|---|
| Inserts/BookingInserts.sql | Contains prepared SQL inserts for XML booking records |
| Inserts/ClientInserts.sql | Contains prepared SQL inserts for XML client records |
| Inserts/ExpensesInsert.sql | Contains prepared SQL inserts for XML expenserecords |

### 2.7.1 XSD Schemas for Record Validation

Because checking that an XML document complies with the specification, we created XSD's (XML Schemas) for each document in order to validate documents before insertion. Each document refers to its XML schema in its header file like so:

**<employee                    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instancexsi:noNamespaceSchemaLocation="http://group17.99k.org/Employee.xsd">**

This says that the whole XML document has to conform to XML schema set by W3, as well as having to conform to the XSD document, which is in this case uploaded onto a website.

These schemas themselves can be found within the zip file:

| XSD/Employee.xsd | Contains XSD schema for used to validate employee records |
|---|---|
| XSD/Booking.xsd | Contains XSD schema for used to validate booking records |
| XSD/Client.xsd | Contains XSD schema for used to validate client records |
| XSD/Expenses.xsd | Contains XSD schema for used to validate expenses records |

## 2.7.2 Revised Tables and Sample Inserts

Please note that we have also revised the tables for this coursework in order to comply with this new design. The script for table creation can be found in the following file:

**Tables.sql**

We have also revised the sample inserts which could also be found in the following files:

| Inserts/EmployeeInserts.sql | Contains prepared SQL inserts for XML employee records |
|---|---|
| Inserts/BookingInserts.sql | Contains prepared SQL inserts for XML booking records |
| Inserts/ClientInserts.sql | Contains prepared SQL inserts for XML client records |
| Inserts/ExpensesInsert.sql | Contains prepared SQL inserts for XML expense records |
| Inserts/TestInserts.sql | Contains all other inserts for relational tables |

## 2.7.3 Sample Queries for XML Documents

Sample queries that test out the structure of the new database with document integration can be found in the following files:

| Queries/EmployeeQueries.sql | Contains all the queries that focus on employee records. |
|---|---|
| Queries/BookingQueries.sql | Contains all the queries that focus on booking records. |
| Queries/ClientQueries.sql | Contains all the queries that focus on client records. |
| Queries/ExpensesQueries.sql | Contains all the queries that focus on expenses records. |
| Queries/ GeneralQueries.sql | Other general queries, mainly joins of many tables. |

# 3. Test Queries, Updates & Results

## 3.1 Queries on Expense Records

Do note that the scripts for these queries can be found in ExpensesQueries.sql file.

**Query:** Display all the expenses that are car maintenance related

```
SELECT
ExpenseId as ExpenseID,
extractValue(expenseInfo, '/Expense/CostName') as CostName,
extractValue(expenseInfo, '/Expense/CostValue') as CostValue,
extractValue(expenseInfo, '/Expense/CostDate') as CostDate,
extractValue(expenseInfo, '/Expense/AdditionalInfo/CarRegNo') as CarRegNo,
extractValue(expenseInfo, '/Expense/AdditionalInfo/DriverContribution') as
DriverContribution,
extractValue(expenseInfo, '/Expense/AdditionalInfo/ContribReceived') as
ContribReceived
FROM Expenses
WHERE existsNode(expenseInfo, '/Expense/AdditionalInfo/CarRegNo')>0;
```

**Result:**

|   | EXPENSEID | COSTNAME | COSTVALUE | COSTDATE | CARREGNO | DRIVERCONTRIBUTION | CONTRIBRECEIVED |
|---|---|---|---|---|---|---|---|
| 1 | 0 | Replacement Bulb | 25 | 2012-12-10 | RF58MUY | 5 | Paid |
| 2 | 1 | Replacement Bulb | 25 | 2012-12-10 | RF58MUY | 5 | Paid |
| 3 | 3 | Tyres | 300 | 2012-12-12 | BD51SMR | 60 | Outstanding |
| 4 | 7 | Screen Wash | 20 | 2012-12-14 | LO57FRT | 4 | Outstanding |
| 5 | 8 | MOT | 130 | 2012-12-14 | BD51SMR | 26 | Paid |

**Query:** Display all the other expenses

```
SELECT
ExpenseId as ExpenseID,
extractValue(expenseInfo, '/Expense/CostName') as CostName,
extractValue(expenseInfo, '/Expense/CostValue') as CostValue,
extractValue(expenseInfo, '/Expense/CostDate') as CostDate,
extractValue(expenseInfo, '/Expense/AdditionalInfo/ExpenseType') as ExpenseType
FROM Expenses
WHERE existsNode(expenseInfo, '/Expense/AdditionalInfo/ExpenseType')>0;
```

**Result:**

|   | EXPENSEID | COSTNAME | COSTVALUE | COSTDATE | EXPENSETYPE |
|---|---|---|---|---|---|
| 1 | 2 | Stationary | 30 | 2012-12-11 | Other |
| 2 | 4 | Gas Bill | 275 | 2012-12-13 | Utilities |
| 3 | 5 | Electricity Bill | 460 | 2012-12-13 | Utilities |
| 4 | 6 | Catering | 120 | 2012-12-14 | Other |

**Query:** Display all expenses for cars that cost was 25

```
SELECT
ExpenseId as ExpenseID,
extractValue(expenseInfo, '/Expense/CostName') as CostName,
extractValue(expenseInfo, '/Expense/CostValue') as CostValue,
extractValue(expenseInfo, '/Expense/AdditionalInfo/CarRegNo') as CarReg
FROM Expenses
WHERE existsNode(expenseInfo, '/Expense/AdditionalInfo/CarRegNo')>0
AND extractValue(expenseInfo, '/Expense/CostValue') = 25;
```

**Result:**

|   | EXPENSEID | COSTNAME | COSTVALUE | CARREG |
|---|-----------|----------|-----------|--------|
| 1 | 0 | Replacement Bulb | 25 | RF58MUY |
| 2 | 1 | Replacement Bulb | 25 | RF58MUY |

**Query:** Displayall the car expenses that are outstanding and print all the drivers that are driving the cars

```
SELECT ExpenseId as ExpenseID,DriverID, Employees.EmployeeID,
extractValue(expenseInfo, '/Expense/CostName') as CostName,
extractValue(expenseInfo, '/Expense/CostValue') as CostValue,
extractValue(expenseInfo, '/Expense/AdditionalInfo/CarRegNo') as CarRegNo,
extractValue(expenseInfo, '/Expense/AdditionalInfo/DriverContribution') as
DriverContribution,
extractValue(expenseInfo, '/Expense/AdditionalInfo/ContribReceived') as
ContribReceived,
extractValue(employeeInfo,'/employee/firstname') as DriverName,
extractValue(employeeInfo,'/employee/lastname') as DriverSurname
FROM Expenses, Drivers ,Employees
WHERE extractValue(expenseInfo, '/Expense/AdditionalInfo/CarRegNo') =
Drivers.CabRegNo
AND Employees.EmployeeID = Drivers.EmployeeID
AND existsNode(expenseInfo, '/Expense/AdditionalInfo/CarRegNo')>0
AND extractValue(expenseInfo, '/Expense/AdditionalInfo/ContribReceived') =
'Outstanding';
```

**Result:**

|   | EXPENSEID | DRIVERID | EMPLOYEEID | COSTNAME | COSTVALUE | CARREGNO | DRIVERCONTRIBUTION | CONTRIBRECEIVED | DRIVERNAME | DRIVERSURNAME |
|---|-----------|----------|------------|----------|-----------|----------|---------------------|-----------------|------------|---------------|
| 1 | 3 | 5 | 7 | Tyres | 300 | BD51SMR | 60 | Outstanding | Saundra | Heft |
| 2 | 7 | 15 | 23 | Screen Wash | 20 | L057FRT | 4 | Outstanding | Mindy | Buntler |

| **Query:** Update contribution received of expense to "Paid" |
|---|
| **UPDATE Expenses SET ExpenseInfo = UPDATEXML(ExpenseInfo,'/Expense/AdditionalInfo/ContribReceived/text()','Paid') WHERE ExpenseId = 3;** |
| **Result:** |

**BEFORE:**

| | EXPENSEID | STATUS |
|---|---|---|
| 1 | 3 | Outstanding |

**AFTER:**

| | EXPENSEID | STATUS |
|---|---|---|
| 1 | 3 | Paid |

| **Query:** Update expense name to "New Toilet" |
|---|
| **UPDATE Expenses SET ExpenseInfo = UPDATEXML(ExpenseInfo,'/Expense/CostName/text()','New Toilet') WHERE ExpenseId = 2;** |
| **Result:** |

**BEFORE:**

| | EXPENSEID | COSTNAME |
|---|---|---|
| 1 | 2 | Stationary |

**AFTER:**

| | EXPENSEID | COSTNAME |
|---|---|---|
| 1 | 2 | New Toilet |

## 3.2 Queries on Employee Records

Do note that the scripts for these queries can be found in EmployeeQueries.sql file.

| Query: Display all employee information |
| --- |

```
SELECT
employeeid as EmployeeID,
extractValue(employeeinfo,'/employee/firstname') as FirstName,
extractValue(employeeinfo,'/employee/lastname') as LastName,
extractValue(employeeinfo,'/employee/address/house') as HouseNum,
extractValue(employeeinfo,'/employee/address/street') as Street,
extractValue(employeeinfo,'/employee/address/postcode') as Postcode,
extractValue(employeeinfo,'/employee/address/town') as Town,
extractValue(employeeinfo,'/employee/phone') as Phone,
extractValue(employeeinfo,'/employee/ratetype') as RateType,
extractValue(employeeinfo,'/employee/ratevalue') as RateValue,
extractValue(employeeinfo,'/employee/hire') as HireDate,
extractValue(employeeinfo,'/employee/shift') as Shift
FROM EMPLOYEES;
```

**Result:**

| | EMPLOYEEID | FIRSTNAME | LASTNAME | HOUSENUM | STREET | POSTCODE | TOWN | PHONE | RATETYPE | RATEVALUE | HIREDATE | SHIFT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 7 | Saundra | Heft | 54 | Quinse Road | W1F7PL | London | 07549665593 | F | 6 | 2012-12-10 | 2 |
| 2 | 8 | Neil | Havis | 1 | Olah Street | W1T4JR | London | 07289441295 | S | 30 | 2012-12-10 | 5 |
| 3 | 9 | Clinton | Beitz | 23 | Queensbury Street | W139TY | London | 07433127445 | S | 30 | 2012-12-10 | 20 |
| 4 | 10 | Ted | Kinnison | 46 | Anadel Road | W42RA | London | 07549223495 | F | 6 | 2012-12-10 | 11 |
| 5 | 11 | Odessa | Perugini | 75 | Roman Street | W41LU | London | 07884151895 | F | 6 | 2012-12-10 | 32 |
| 6 | 12 | Pengwei | Yun | 32 | Omah Street | W1D5PH | London | 0784425189 | F | 6 | 2012-12-10 | 43 |
| 7 | 13 | Marly | Hunt | 66 | Bishop Street | W1A3WZ | London | 07885358895 | F | 6 | 2012-12-10 | 54 |
| 8 | 14 | Mayan | Dang | 12 | Canel Woods Road | W1D4HZ | London | 07845455139 | F | 7.5 | 2012-12-10 | 75 |
| 9 | 15 | Richar | Huntler | 54 | Manchester Street | W1K4JU | London | 07834359895 | F | 7.5 | 2012-12-10 | 76 |
| 10 | 16 | Ben | Hunt | 17 | River Syf | W130NQ | London | 07834359895 | F | 7.5 | 2012-12-10 | 17 |
| 11 | 17 | Charles | Huntler | 5 | Mayhem Road | W144DJ | London | 07834359895 | F | 8 | 2012-12-10 | 28 |
| 12 | 18 | Tom | Maxez | 44 | Granville Place | W84QX | London | 07834359895 | F | 7.5 | 2012-12-10 | 39 |
| 13 | 19 | Tony | Huxaa | 15 | Turndown Road | W140NL | London | 07834359895 | F | 8 | 2012-12-10 | 40 |
| 14 | 20 | Mark | Hunae | 66 | Eten Street | W1K1AN | London | 07834359895 | F | 7.5 | 2012-12-10 | 51 |
| 15 | 21 | Marcus | Muntler | 44 | Shenley Road | W1H1BX | London | 07834359895 | F | 8 | 2012-12-10 | 62 |

| Query: Display all information of driver employees |
|---|

```
SELECT
EMPLOYEES.EMPLOYEEID as EmployeeNum,
extractValue(employeeinfo,'/employee/firstname') as FirstName,
extractValue(employeeinfo,'/employee/lastname') as LastName,
extractValue(employeeinfo,'/employee/address/house') as HouseNum,
extractValue(employeeinfo,'/employee/address/street') as Street,
extractValue(employeeinfo,'/employee/address/postcode') as Postcode,
extractValue(employeeinfo,'/employee/address/town') as Town,
extractValue(employeeinfo,'/employee/phone') as Phone,
extractValue(employeeinfo,'/employee/ratetype') as RateType,
extractValue(employeeinfo,'/employee/ratevalue') as RateValue,
extractValue(employeeinfo,'/employee/hire') as HireDate,
extractValue(employeeinfo,'/employee/shift') as Shift
FROM EMPLOYEES, DRIVERS
WHERE (DRIVERS.EMPLOYEEID = EMPLOYEES.EMPLOYEEID);
```

**Result:**

| | EMPLOYEENUM | FIRSTNAME | LASTNAME | HOUSENUM | STREET | POSTCODE | TOWN | PHONE | RATETYPE | RATEVALUE | HIREDATE | SHIFT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 7 | Saundra | Heft | 54 | Quinse Road | W1F7PL | London | 07549665593 | F | 6 | 2012-12-10 | 2 |
| 2 | 9 | Clinton | Beitz | 23 | Queensbury Street | W139TY | London | 07433127445 | S | 30 | 2012-12-10 | 20 |
| 3 | 15 | Richar | Huntler | 54 | Manchester Street | W1K4JU | London | 07834359895 | F | 7.5 | 2012-12-10 | 76 |
| 4 | 16 | Ben | Hunt | 17 | River Syf | W130NQ | London | 07834359895 | F | 7.5 | 2012-12-10 | 17 |
| 5 | 17 | Charles | Huntler | 5 | Mayhem Road | W144DJ | London | 07834359895 | F | 8 | 2012-12-10 | 28 |
| 6 | 18 | Tom | Maxez | 44 | Granville Place | W84QX | London | 07834359895 | F | 7.5 | 2012-12-10 | 39 |
| 7 | 19 | Tony | Huxaa | 15 | Turndown Road | W140NL | London | 07834359895 | F | 8 | 2012-12-10 | 40 |
| 8 | 20 | Mark | Hunae | 66 | Eten Street | W1K1AN | London | 07834359895 | F | 7.5 | 2012-12-10 | 51 |
| 9 | 21 | Marcus | Muntler | 44 | Shenley Road | W1H1BX | London | 07834359895 | F | 8 | 2012-12-10 | 62 |
| 10 | 22 | Mike | Sadur | 54 | Quinse Road | W1F7PL | London | 07834359895 | F | 7.5 | 2012-12-10 | 13 |
| 11 | 23 | Mindy | Buntler | 1 | Olah Street | W1T4JR | London | 07834359895 | F | 8 | 2012-12-10 | 24 |
| 12 | 1 | Mike | Dan | 17 | River Syf | W130NQ | London | 07589625493 | F | 7 | 2012-12-10 | 2 |
| 13 | 2 | Pan | Swift | 5 | Mayhem Road | W144DJ | London | 07289141196 | S | 15 | 2012-12-10 | 3 |
| 14 | 3 | Louis | Bend | 44 | Granville Place | W84QX | London | 07539121495 | F | 8 | 2012-12-10 | 1 |
| 15 | 6 | Rae | Freda | 44 | Shenley Road | W1H1BX | London | 02089721494 | S | 15 | 2012-12-10 | 4 |

| Query: Display employees shifts |
|---|

```
SELECT      extractValue(employeeinfo,'/employee/firstname') as FirstName,
extractValue(employeeinfo,'/employee/lastname') as LastName, SHIFTS.STARTHOURS,
SHIFTS.ENDHOURS, SHIFTDAYS.MONDAY, SHIFTDAYS.TUESDAY, SHIFTDAYS.WEDNESDAY,
SHIFTDAYS.THURSDAY, SHIFTDAYS.FRIDAY, SHIFTDAYS.SATURDAY, SHIFTDAYS.SUNDAY
FROM  EMPLOYEES, SHIFTS, SHIFTDAYS
WHERE      (SHIFTDAYS.SHIFTDAYSID = SHIFTS.SHIFTDAYSID) AND
           (SHIFTS.SHIFTID = extractValue(employeeinfo,'/employee/shift'));
```

**Result:**

| | FIRSTNAME | LASTNAME | STARTHOURS | ENDHOURS | MONDAY | TUESDAY | WEDNESDAY | THURSDAY | FRIDAY | SATURDAY | SUNDAY |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Saundra | Heft | 16 | 0 | Y | Y | Y | Y | Y | N | N |
| 2 | Neil | Havis | 16 | 0 | N | N | N | N | N | Y | Y |
| 3 | Jorge | Moreno | 8 | 16 | Y | Y | Y | Y | Y | N | N |
| 4 | Mike | Dan | 16 | 0 | Y | Y | Y | Y | Y | N | N |
| 5 | Pan | Swift | 0 | 8 | Y | Y | Y | Y | Y | N | N |
| 6 | Louis | Bend | 8 | 16 | Y | Y | Y | Y | Y | N | N |
| 7 | Miro | Karakol | 16 | 0 | Y | Y | Y | Y | Y | N | N |
| 8 | Jonas | Brothers | 0 | 8 | Y | Y | Y | Y | Y | N | N |
| 9 | Rae | Freda | 8 | 16 | N | N | N | N | N | Y | Y |

| **Query:** Change phone number of employee 7 |
|---|
| ```
UPDATE ZR300.Employees SET EmployeeInfo =
        UPDATEXML(EmployeeInfo,'/employee/phone/text()', '0205631234')
        WHERE EmployeeID = 7;
``` |
| **Result:** |
| BEFORE:<br><br>| | EMPLOYEEID | PHONE |<br>|---|---|---|<br>| 1 | 7 | 07646573211 |<br><br>AFTER:<br><br>| | EMPLOYEEID | PHONE |<br>|---|---|---|<br>| 1 | 7 | 0205631234 | |

| **Query:** Change the rate of an employee |
|---|
| ```
UPDATE ZR300.Employees SET EmployeeInfo =
        UPDATEXML(EmployeeInfo,'/employee/ratevalue/text()', '9')
        WHERE EmployeeID = 7;
``` |
| **Result:** |
| BEFORE:<br><br>| | EMPLOYEEID | RATE |<br>|---|---|---|<br>| 1 | 7 | 6 |<br><br>AFTER:<br><br>| | EMPLOYEEID | RATE |<br>|---|---|---|<br>| 1 | 7 | 9 | |

## 3.3 Queries on Client Records

Do note that same queries can be found within ClientQueries.sql file.

| **Query:** Display client information |
|---|
| `SELECT ClientID as ID, extractValue(clientinfo,'/Client/CompanyName') as CompanyName, extractValue(clientinfo,'/Client/ClientType') as ClientType, extractValue(clientinfo,'/Client/Discount') as Discount, extractValue(clientinfo,'/Client/Street') as Street, extractValue(clientinfo,'/Client/HouseNo') as BuildingNo, extractValue(clientinfo,'/Client/PostCode') as PostCOde, extractValue(clientinfo,'/Client/City') as City FROM Clients;` |
| **Result:** |

| | ID | COMPANYNAME | CLIENTTYPE | DISCOUNT | STREET | BUILDINGNO | POSTCODE | CITY |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | Maddison Engineering | Private | 2.5 | Manchester Street | 54 | W1K 4JU | London |
| 2 | 1 | Big Baboon Web Design | Corporate | 1.5 | Sunny Run Street | 61 | W1K 3EP | London |
| 3 | 2 | The Peaceful Zebra Devs | Private | 6.5 | Emerald Lake Orchard Road | 14 | W11 9AQ | London |
| 4 | 3 | Tall Cow Devs | Corporate | 10 | Orange Snake Road | 53 | W14 0ZN | London |
| 5 | 4 | The Small Tree Engineers | Private | 11 | Big Vale Street | 12 | W52 SZ | London |
| 6 | 5 | Beta Tomato Marketing | Corporate | 2.5 | Lone Castle Road | 65 | W41 BH | London |
| 7 | 6 | The Content Panda Company | Private | 3.3 | Sleepy Point Street | 43 | W36 JH | London |
| 8 | 7 | White Box Builders | Corporate | 2 | Rustic Anchor Street | 41 | W1D 2DS | London |
| 9 | 8 | Black Builders Co | Private | 5 | Bishop Street | 66 | W1A 3WZ | London |

| **Query:** List clients and their bookings (Uses FLOWR Expression) |
|---|
| `Select BookingID, extractValue(bookinginfo ,'/Booking//ClientID') as ClientID, extractValue(clientinfo,'/Client/CompanyName') as CompanyName, XMLQuery('for $i in /Client let $street := $i/Street let $houseno := $i/HouseNo let $postcode := $i/PostCode let $city := $i/City return concat($street," ",$houseno," ",$postcode," ",$city)' passing by value clientinfo returning content ) as Address from Bookings,Clients where existsNode(bookinginfo,'/Booking//ClientID') > 0 AND extractValue(bookinginfo ,'/Booking//ClientID') = ClientID;` |
| **Result:** |

| | BOOKINGID | CLIENTID | COMPANYNAME | ADDRESS |
|---|---|---|---|---|
| 1 | 7 | 1 | Big Baboon Web Design | Sunny Run Street 61 W1K 3EP London |
| 2 | 8 | 3 | Tall Cow Devs | Orange Snake Road 53 W14 0ZN London |
| 3 | 9 | 4 | The Small Tree Engineers | Big Vale Street 12 W52 SZ London |
| 4 | 10 | 5 | Beta Tomato Marketing | Lone Castle Road 65 W41 BH London |
| 5 | 11 | 7 | White Box Builders | Rustic Anchor Street 41 W1D 2DS London |
| 6 | 12 | 7 | White Box Builders | Rustic Anchor Street 41 W1D 2DS London |
| 7 | 13 | 6 | The Content Panda Company | Sleepy Point Street 43 W36 JH London |
| 8 | 5 | 1 | Big Baboon Web Design | Sunny Run Street 61 W1K 3EP London |
| 9 | 6 | 1 | Big Baboon Web Design | Sunny Run Street 61 W1K 3EP London |
| 10 | 14 | 7 | White Box Builders | Rustic Anchor Street 41 W1D 2DS London |
| 11 | 15 | 7 | White Box Builders | Rustic Anchor Street 41 W1D 2DS London |
| 12 | 17 | 6 | The Content Panda Company | Sleepy Point Street 43 W36 JH London |

| **Query:** Using XPath display clients discount |
|---|
| **Select extractValue(clientinfo,'Client/Discount') as Discount  FROM Clients WHERE clientid = 7;** |
| **Result:** |
| DISCOUNT<br>1  2 |

| **Query:** Get how many bookings a client has made in total |
|---|
| **select count(*) as Bookings**<br>**from Bookings**<br>**where existsNode(bookingInfo, '/Booking//ClientID')>0**<br>**AND extractValue(bookingInfo , 'Booking//ClientID') = 7;** |
| **Result:** |
| BOOKINGS<br>1         4 |

| **Query:** Get the next regular booking of a client |
|---|
| **Select BookingID,extractValue(bookinginfo,'/Booking/BookingInfo/RegularBooking/RepeatOn/PickUpDate') as PickUpDate FROM Bookings WHERE extractValue(bookinginfo,'/Booking/BookingInfo/@Type') = 'Regular' AND rownum = 1 AND extractValue(bookinginfo,'/Booking//ClientID') = 7 ORDER BY PickUpDate ASC;** |
| **Result:** |
| BOOKINGID  PICKUPDATE<br>1        14 2012-12-24 |

| **Query:** Update query to change the company name of client 7. |
|---|
| **UPDATE Clients SET ClientInfo**<br>**= UPDATEXML(ClientInfo,'/Client/CompanyName/text()','Johns Paper Company') WHERE ClientID = 7;** |
| **Result:** |
| Before:<br>CLIENTID  COMPANYNAME<br>1       7 White Box Builders<br>After:<br>CLIENTID  COMPANYNAME<br>1       7 Johns Paper Company |

| **Query:** Update query to change the clients discount. |
|---|

```
UPDATE Clients SET ClientInfo
= UPDATEXML(ClientInfo,'/Client/Discount/text()',"7.8") WHERE ClientID = 2;
```

**Result:**

BEFORE:

| | CLIENTID | DISCOUNT |
|---|---|---|
| 1 | 2 | 6.5 |

AFTER:

| | CLIENTID | DISCOUNT |
|---|---|---|
| 1 | 2 | 7.8 |

| **Query:** Update query to change client type. |
|---|

```
UPDATE Clients SET ClientInfo
= UPDATEXML(ClientInfo,'/Client/ClientType/text()','Private') WHERE ClientID = 3;
```

**Result:**

Before:

| | CLIENTID | CLIENTTYPE |
|---|---|---|
| 1 | 3 | Corporate |

After:

| | CLIENTID | CLIENTTYPE |
|---|---|---|
| 1 | 3 | Private |

## 3.4 Queries on Booking Records

Please note that same booking queries can be found within BookingQueries.sql file.

| Query: Get all bookings made by client 7. |
|---|
| **SELECT BookingID from Bookings where extractValue(bookinginfo,'/Booking//ClientID') = 7;** |
| **Result:** |

| | BOOKINGID |
|---|---|
| 1 | 11 |
| 2 | 12 |
| 3 | 14 |
| 4 | 15 |

| Query: Using XPath showClient details of regular bookings |
|---|
| **SELECT BOOKINGID,** **extractValue(clientinfo,'Client/CompanyName') as ClientName,** **extractValue(clientinfo,'Client/ClientType') as ClientType,** **extractValue(clientinfo,'Client/Discount') as Discount,** **extractValue(clientinfo,'Client/Street') as Street,** **extractValue(clientinfo,'Client/HouseNo') as HouseNo,** **extractValue(clientinfo,'Client/PostCode') as PostCode,** **extractValue(clientinfo,'Client/City') as City** **from Bookings,Clients where** **extractValue(bookinginfo,'/Booking/BookingInfo/@Type') = 'Regular' AND** **extractValue(bookinginfo,'/Booking/BookingInfo/RegularBooking/ClientID') = 3;** |
| **Result:** |

| | BOOKINGID | CLIENTNAME | CLIENTTYPE | DISCOUNT | STREET | HOUSENO | POSTCODE | CITY |
|---|---|---|---|---|---|---|---|---|
| 1 | 8 | Maddison Engineering | Private | 2.5 | Manchester Street | 54 | W1K 4JU | London |
| 2 | 8 | Big Baboon Web Design | Corporate | 1.5 | Sunny Run Street | 61 | W1K 3EP | London |
| 3 | 8 | The Peaceful Zebra Devs | Private | 7.8 | Emerald Lake Orchard Road | 14 | W11 9AQ | London |
| 4 | 8 | Tall Cow Devs | Private | 10 | Orange Snake Road | 53 | W14 0ZN | London |
| 5 | 8 | The Small Tree Engineers | Private | 11 | Big Vale Street | 12 | W52 SZ | London |
| 6 | 8 | Beta Tomato Marketing | Corporate | 2.5 | Lone Castle Road | 65 | W41 BH | London |
| 7 | 8 | The Content Panda Company | Private | 3.3 | Sleepy Point Street | 43 | W36 JH | London |
| 8 | 8 | Johns Paper Company | Corporate | 2 | Rustic Anchor Street | 41 | W1D 2DS | London |
| 9 | 8 | Black Builders Co | Private | 5 | Bishop Street | 66 | W1A 3WZ | London |

**Query:** Using FLOWR expression mixed with XPath collect information on regular bookings

```
SELECT bookingid ,XMLQuery('for $i in /Booking/BookingInfo
for $j in $i/RegularBooking/RepeatOn
let $PickUpDate := $j/PickUpDate
let $Driver := $j/DriverID
let $interval := $j/WeeklyInterval
where $i/@Type="Regular"
return concat(" Driver:",$Driver," PickUpDate:",$PickUpDate,"
WeeklyInterval:",$interval/text(),)'
passing by value bookinginfo
returning content ) AS Information
FROM bookings
WHERE extractValue(bookinginfo,'/Booking/BookingInfo/@Type') = 'Regular';
```

**Result:**

| | BOOKINGID | INFORMATION |
|---|---|---|
| 1 | 5 | Driver:15 PickUpDate:2012-12-11 WeeklyInterval:2 |
| 2 | 6 | Driver:15 PickUpDate:2012-12-10 WeeklyInterval:3 |
| 3 | 7 | Driver:2 PickUpDate:2012-12-12 WeeklyInterval:1 |
| 4 | 8 | Driver:15 PickUpDate:2012-12-14 WeeklyInterval:2 |
| 5 | 10 | Driver:16 PickUpDate:2012-12-19 WeeklyInterval:3 |
| 6 | 9 | Driver:13 PickUpDate:2012-12-15 WeeklyInterval:4 |
| 7 | 13 | Driver:11 PickUpDate:2012-12-20 WeeklyInterval:4 |
| 8 | 14 | Driver:16 PickUpDate:2012-12-24 WeeklyInterval:2 |
| 9 | 15 | Driver:15 PickUpDate:2012-12-26 WeeklyInterval:1 |
| 10 | 17 | Driver:11 PickUpDate:2012-12-19 WeeklyInterval:4 |

**Query:** Using XPath  get ID's of all normal bookings

```
SELECT BOOKINGID from Bookings where
extractValue(bookinginfo,'/Booking/BookingInfo/@Type') = 'Normal';
```

**Result:**

| | BOOKINGID |
|---|---|
| 1 | 4 |
| 2 | 11 |
| 3 | 12 |
| 4 | 16 |
| 5 | 1 |
| 6 | 2 |
| 7 | 3 |

| **Query:** Check which regular booking(s) fall on the given date |
|---|
| ```
SELECT BookingID ,
extractValue(bookinginfo,'/Booking/BookingInfo/RegularBooking/ClientID') as Client,
extractValue(bookinginfo,'/Booking/BookingInfo/RegularBooking/RepeatOn/PickUpDate')
as PickUpDate
FROM Bookings
WHERE
extractValue(bookinginfo,'/Booking/BookingInfo/RegularBooking/RepeatOn/PickUpDate')
= '2012-12-24';
``` |
| **Result:** |

| | BOOKINGID | CLIENT | PICKUPDATE |
|---|---|---|---|
| 1 | 14 | 7 | 2012-12-24 |

| **Query:** Get all bookings for a given date as well as their drivers and pickup/drop addresses and drivers responsible.(Date chosen in this query is : 2012-12-05) |
|---|
| ```
Select BookingID,
extractValue(bookinginfo,'/Booking/Name') as PassengerName,
extractValue(bookinginfo,'/Booking/PhoneNumber') as PassengerPhone,
extractValue(bookinginfo,'/Booking/PickUpAddress/PostCode') as PickUpPostCode,
extractValue(bookinginfo,'/Booking/DestinationAddress/PostCode') as
DestinationPostCode,
extractValue(bookinginfo,'/Booking//PickUpDate') AS PickUpDate,
extractValue(bookinginfo,'/Booking//DriverID') AS DriverID,
extractValue(employeeinfo,'/employee/firstname') AS DriverName,
extractValue(employeeinfo,'/employee/lastname') AS DriverSurname,
Drivers.CabRegNo as Car
FROM Bookings,Drivers,Employees
WHERE to_date(extractValue(bookinginfo,'/Booking//PickUpDate'),'YYYY-MM-DD') =
to_date('2012-12-05','YYYY-MM-DD')
AND Drivers.DriverID =  extractValue(bookinginfo,'/Booking//DriverID')
AND Employees.EmployeeID = Drivers.DriverID;
``` |
| **Result:** |

| | BOOKINGID | PASSENGERNAME | PASSENGERPHONE | PICKUPPOSTCODE | DESTINATIONPOSTCODE | PICKUPDATE | DRIVERID | DRIVERNAME | DRIVERSURNAME | CAR |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Thomas | 07555451275 | W71 DA | W139RR | 2012-12-05 | 1 | Mike | Dan | MF51SQL |
| 2 | 2 | James | 07555431275 | W120TE | W71 DA | 2012-12-05 | 2 | Pan | Swift | FE54SQA |

| |
|---|
| **Query:** Cancel a regular booking which was on a given date booked by client with id 3. |
| **UPDATE Bookings SET BookingInfo = UPDATEXML(BookingInfo,'/Booking/BookingInfo/RegularBooking/RepeatOn/RegStatus/text()','CANCELLED') WHERE extractValue(bookinginfo,'/Booking/BookingInfo/RegularBooking/RepeatOn/PickUpDate') = '2012-12-14' AND extractValue(bookinginfo,'/Booking/BookingInfo/RegularBooking/ClientID') = 3;** |
| **Result:** |

<div align="center">

**BEFORE:**

| | BOOKINGID | REGULARBOOKINGSTATUS |
|---|---|---|
| 1 | 8 | ONGOING |

**AFTER:**

| | BOOKINGID | REGULARBOOKINGSTATUS |
|---|---|---|
| 1 | 8 | CANCELLED |

</div>

| |
|---|
| **Query:** Defer a booking to next repeat (add the weekly interval to the next pick up date) |
| **UPDATE Bookings SET BookingInfo = UPDATEXML(BookingInfo,'/Booking/BookingInfo/RegularBooking/RepeatOn/PickUpDate/text()', to_char((to_date(extractValue(bookinginfo,'/Booking/BookingInfo/RegularBooking/RepeatOn/PickUpDate'),'YYYY-MM-DD')+ (extractValue(bookinginfo,'/Booking/BookingInfo/RegularBooking/RepeatOn/WeeklyInterval')*7)),'YYYY-MM-DD')) WHERE extractValue(bookinginfo,'/Booking/BookingInfo/RegularBooking/RepeatOn/PickUpDate') = '2012-12-20' AND extractValue(bookinginfo,'/Booking/BookingInfo/RegularBooking/ClientID') = 6;** |
| **Result:** |

<div align="center">

**BEFORE:**

| | BOOKINGID | PICKUPDATE |
|---|---|---|
| 1 | 13 | 2012-12-20 |

**AFTER:**

| | BOOKINGID | PICKUPDATE |
|---|---|---|
| 1 | 13 | 2013-01-17 |

</div>

# 4. References

[1] **What is XML and Why Should Companies Use It?** (Alan Pelz-Sharpe, *April 2010*)
This web article by Inc. explains why companies should use XML semi-structured data.
http://www.inc.com/guides/2010/04/why-companies-should-use-xml.html
**Accessed:** 5th December 2012

[2] **Oracle Database 11g The Complete Reference** (Kevin Loney, *January 2009*)
*Chapter 52: "The Hitchhiker's Guide to XML in Oracle"* - **ISBN:** 0071598758
This book chapter explained the general concepts of how XML is used within Oracle and how to store and retrieve XML data. It also explained how to validate XML documents using XSD and/or DTD schemas.

[3] **Why should I use XML?** (Peter Flynn, *January 2011*)
Another web article discussing ideas behind usage of semi-structured data, such as XML.
http://xml.silmaril.ie/whyxml.html
**Accessed:** 9th December 2012

[4] **Using XMLType** (Oracle Corporation, *October 2002*)
A great resource on explaining how to use XMLType and query XMLType documents written by Oracle itself.
http://docs.oracle.com/cd/B10500_01/appdev.920/a96620/xdb04cre.htm
**Accessed:** 3rd December 2012

[5] **XML DTDs Vs XML Schema** (Michael Jervis, *November 2002*)
Good article written on explaining the differences between XSD and DTD schemas to validate XML Documents.
http://www.sitepoint.com/xml-dtds-xml-schema/
**Accessed:** 5th December 2012