Generating MySQL Database from an ER Diagram - an XML-based Solution

Group 1 : Po-Chih Chang, Yen-Ting Chen, Shiou-Tong Lin

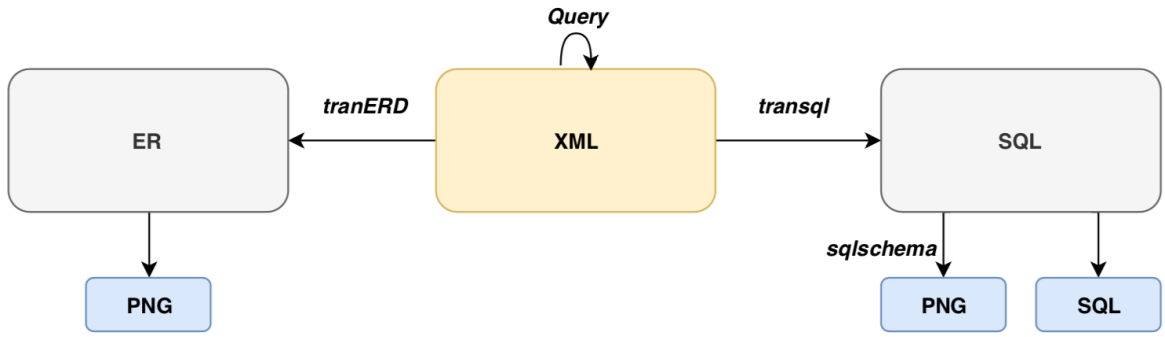
Abstract

The Entity-Relationship(ER) model and relational database are two popular methodologies for designing database. The ER model is famous for its simple concept and visual representation. On the other hand, the relational database is a well-known database with structured data. This work aims to map ER model and relational database to each other through the help of XML file.

1. Introduction

Our work is trying to make the workflow of transferring ER diagram to MySQL database more efficiently, flexibly, and easily. After researching [1][2], we discovered that xml is a good choice to describe an ER diagram, so we decided to create our own XML format, which can be parsed and converted in to ER diagram and .sql file.

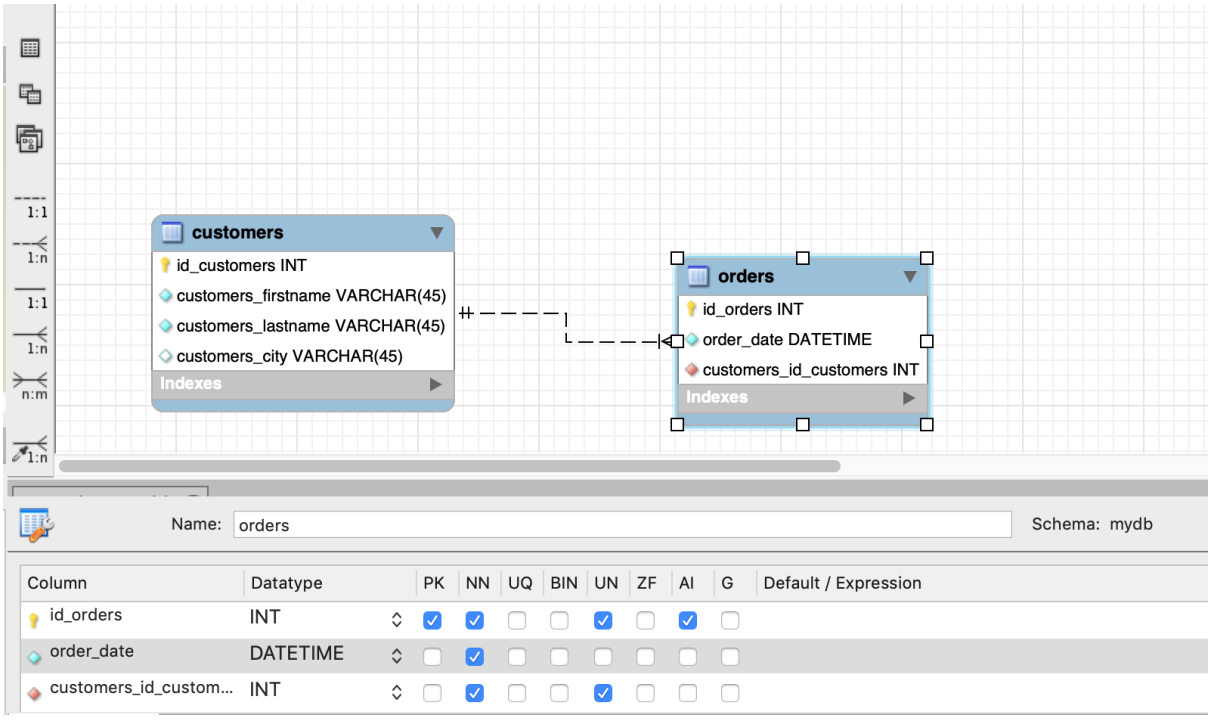
The following picture is our workflow. We can use ***query*** to determine which part of the XML that describes ER diagram we want, and can use ***tranERD***, ***transql*** to produce the corresponding ER diagram picture and .sql file. We also create the function ***sqlschema***, which shows the schema of database created by our queried .sql file.



2. Prior work

ER-Draw [1] is an educational prototype that developed an XML-based ER-diagram and translation tool. This tool support object oriented techniques, and is able to get the relational database schemas from ER-diagram. μE Framework [2] can map ER-diagram to Relational database schema and vice versa. In [2], two types of XML document were shown to represent ER-diagram and relational database, which inspire us to combine the concepts and develop a new type of XML document.

SQL Workbench supports the functions of creating ER diagram and converting it into sql schema, but it can't record details about ER diagram such as weak entity, relationship name, composite attributes, multi-value attributes, total/partial participation, etc. Also, if the size of ER diagram is huge, it's very difficult to draw the whole ER diagram from MySQL Workbench.



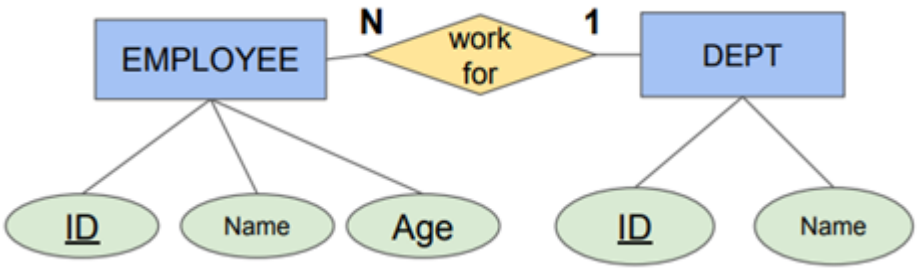
3. Our work

3.1. XML Document

Compared to [2], the XML document in this work not only represents the ER-diagram, but also has efficient information to transform the ER-diagram into RDB. Figure shows an example of ER-diagram, and the XML for this ER-diagram is shown in Figure . We extend the ER-diagram XML document in [2] by adding several ATT(to distinguish two attribute in XML and ER-diagram, we call the one in XML "ATT" in this paper) into different elements. The following lists all ATT in all elements.

1) Entity
    a) name : Represent the name of the entity. Must exist.
    b) weak : Has the value "True" if the entity is a weak entity.
2) Attribute
    a) name : Represent the name of the attribute.
    b) type : Record MySQL data type of the attribute. Must exist.
    c) bit : Record the bits for type "char" and "varchar"
    d) notNull : Has the value "True" if the attribute cannot be Null.
    e) PK : Has the value "True" if the attribute belongs to primary key.
    f) multi : Has the value "True" if the attribute is multi-valued.
    g) Part :
    h) default : Record the default value of the attribute if exists.

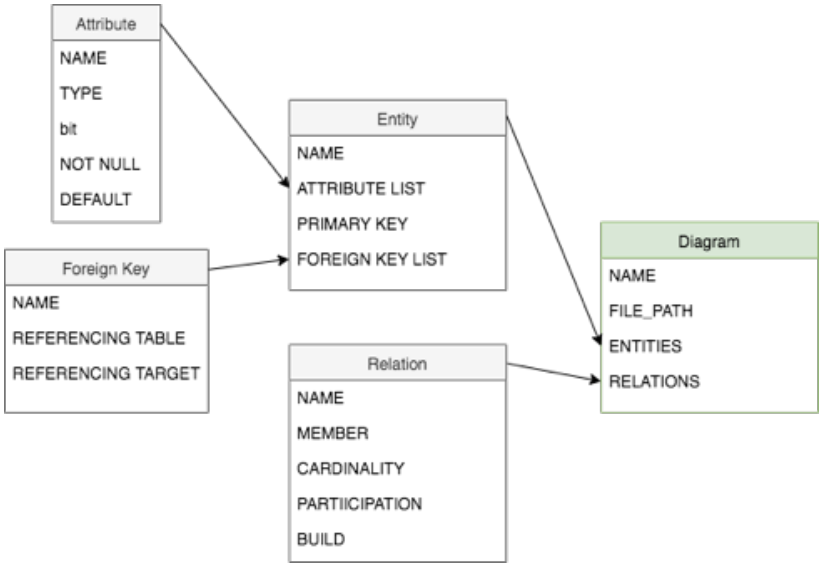
3) Relation
    a) name : Represent the name of the Relation.
4) Member
    a) name : Represent the name of the entity in this relation.
    b) value : Has the value “total” or “partial” to represent total participation or partial participation
    c) Cardinality : Has the value “one” or “many” to show the cardinality ratio of the entity in this relation
    d) ref :




```xml
<?xml version="1.0"?>
<ER name = "Company">
<!-- Entity -->
    <Entity name = "Employee">
        <Attribute name = "ID" type = "int" notNull = "True" PK = "True"/>
        <Attribute name = "Name" type = "varchar" bit = "10" notNull = "True"/>
        <Attribute name = "Age" type = "int" default = "20"/>
        <Attribute name = "Depart_ID" type = "int" notNull = "True"/>
    </Entity>
    <Entity name = "Department">
        <Attribute name = "ID" type = "int" notNull = "True" PK = "True" />
        <Attribute name = "Name" type = "varchar" bit = "10" notNull = "True"/>
    </Entity>
<!-- Relation -->
    <Relation name = "work_for">
        <member name = "Employee"   value = "total"   Cardinality = "many" ref = "Depart_ID"/>
        <member name = "Department" value = "partial" Cardinality = "one"  ref = "ID"/>
    </Relation>
</ER>
```

3.2. Data Structure for XML

We design a data structure for parsing XML document. The figure shows that we store the information of “Entity” and “Relation” into “Diagram”. After scanning through XML document, we can easily parse necessary information for “Diagram” thanks to clear definition of schema.



According to the workflow on page 1, we can now perform “Query” command or directly output the data in the form of a .sql file or a picture with corresponding ER-Diagram.

3.3. Query

The “Query” command aims to simplify the database by pruning unrelated entities and relations. This work provides three types of query function. User can query by attribute name, by entity name, or by relation name.

1) Query by attribute name

   After the query, entities that contain target attribute will remain and those don’t contain target attribute will be deleted. All the relations will be deleted.

2) Query by entity name

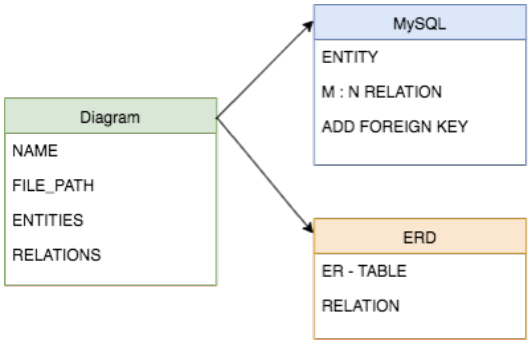
   After the query, the target entity and the related entities, which are away from target entity by one relation, will remain. Other entities will be deleted.

3) Query by relation name

   After the query, the target relation and its member entities will remain. Other relations and entities will be deleted.

3.4. transql & tranERD

By the command “transql” and “trainERD”, user can transform the data in the database into sql file and the picture of ER diagram. Figure shows the workflow.



1) SQL file

We can simply create a table for each entity. However, the way we deal with relations falls into two categories. For a many-to-many relation, we create an additional table to store the information, and use foreign keys to connect the additional table and original tables. For other types of relation, we add foreign keys into the table to connect tables that are related.

2) ER Diagram

In 3.2, we show that the data structure we design for XML parsing is conceptually based on ER model. Therefore, transforming into ER diagram is straightforward.

4. Results

We create a MLB.xml contains 10 entities and 13 relationships to demo our functionality of generating the corresponding sql file, ER diagram picture and the schema of the MySQL database which is created by taking .sql file as input data and querying by attribute name, entity name and relation name.

The final result shows that our ER diagram can preserve all the information of MLB.xml and can generate corresponding sql file correctly. Besides, the results of our query commands are also correct.

0) MLB.xml

```
<Entity name = "Game"> ...
</Entity>
<Entity name = "Statistics">
    <Attribute name = "ID" type = "int" notNull = "True" PK = "True" />
    <Attribute name = "Player" type = "int" notNull = "True" />
    <Attribute name = "Team" type = "int" notNull = "True" />
    <Attribute name = "Pos" type = "varchar" bit = "20" notNull = "True"/>
    <Attribute name = "AVG" type = "float" notNull = "True" />
</Entity>
<Entity name = "FreeAgent">
    <Attribute name = "Player" type = "int" notNull = "True" PK = "True" />
</Entity>

<!-- Relation -->
<Relation name = "Challenge">
    <Member name = "Team" value = "partial" Cardinality = "one" ref = "NextAgainst"/>
    <Member name = "Team" value = "partial" Cardinality = "one" ref = "ID"/>
</Relation>
<Relation name = "Own"> ...
</Relation>
```

1) Parse MLB.xml

Parse MLB.xml into our data structure.

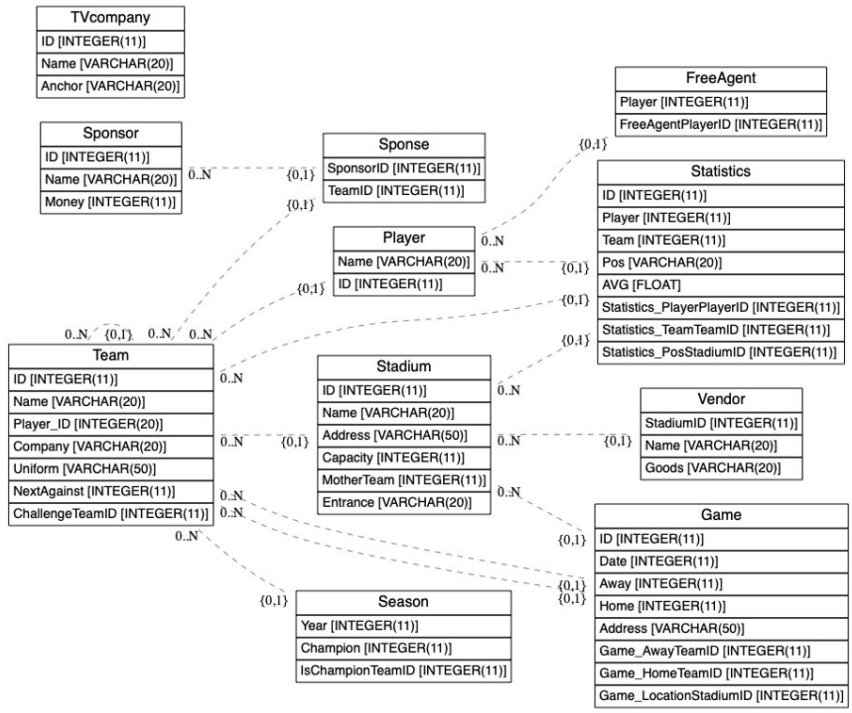
2) **transql demo.sql**

Output the sql file named demo.sql corresponding to MLB.xml.

```
CREATE TABLE Statistics(
    ID int NOT NULL,
    Player int NOT NULL,
    Team int NOT NULL,
    Pos varchar(20) NOT NULL,
    AVG float NOT NULL,
    Statistics_PlayerPlayerID int NOT NULL,
    Statistics_TeamTeamID int NOT NULL,
    Statistics_PosStadiumID int NOT NULL,
    PRIMARY KEY (ID)

);

CREATE TABLE FreeAgent(
    Player int NOT NULL,
    FreeAgentPlayerID int NOT NULL,
    PRIMARY KEY (Player)

);

CREATE TABLE Sponse(
    SponsorID int NOT NULL,
    TeamID int NOT NULL,
    PRIMARY KEY (SponsorID, TeamID)

);

ALTER TABLE Team ADD FOREIGN KEY (ChallengeTeamID) REFERENCES Team(ID);

ALTER TABLE Stadium ADD FOREIGN KEY (MotherTeam) REFERENCES Team(ID);
```

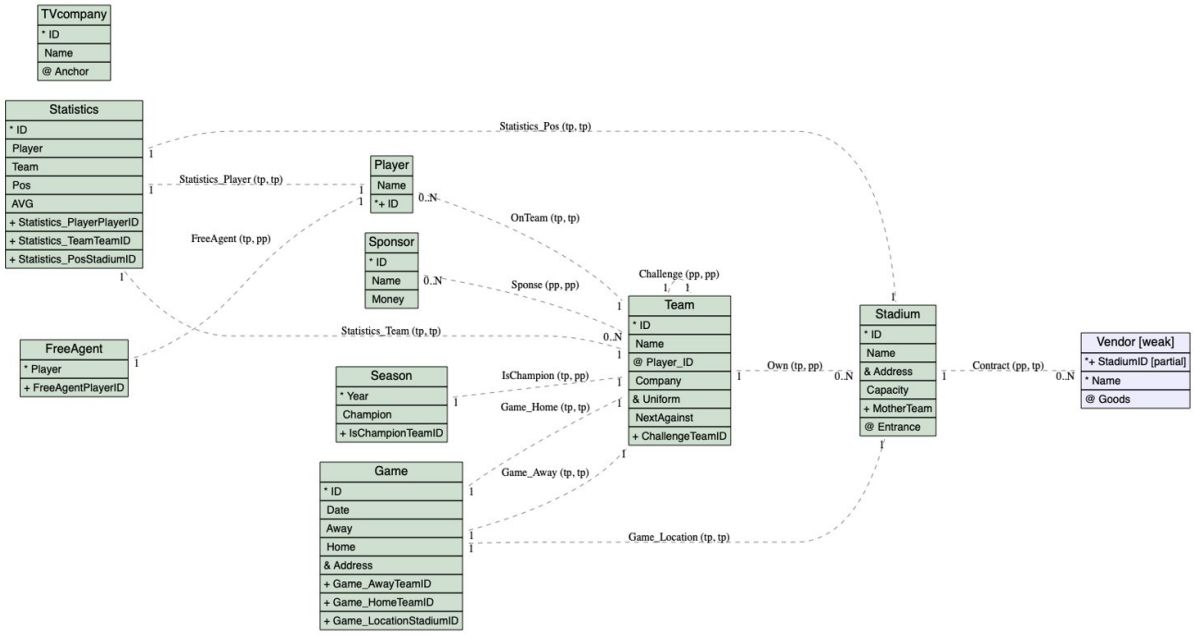
3) **sqlschema demo.sql**

Output the database schema corresponding to demo.sql.



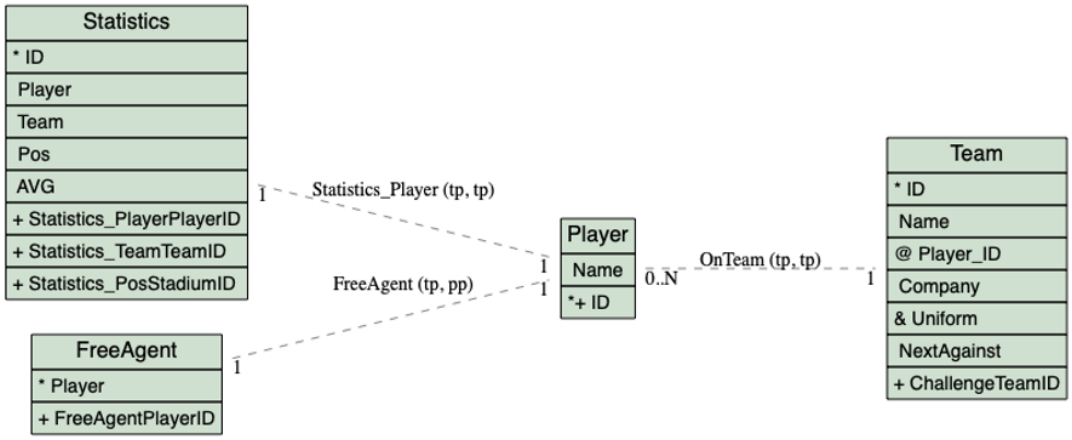
4) **tranERD**

Output the ER diagram picture corresponding to MLB.xml. Note that the special character before attribute name stands for different meaning. For example, '*' means primary key, '&' means composite data type, '@' means multi-valued data type, and '+' means foreign key.



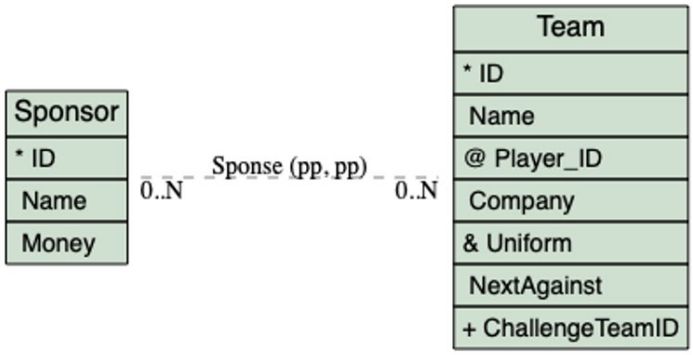
5) **query -e Player**

Query related entities and relationships of the specific entity named Player. After querying, you can get the corresponding ER diagram picture and sql file by calling ***tranERD*** and ***transql <output filename>*** respectively.



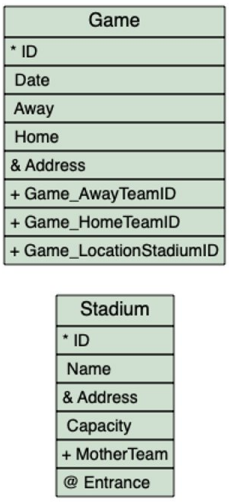
6) **query -r Sponse**

Query entities that have the specific relationship named Sponse.



7) **query -a Address**

Query entities that contain the specific attribute named Address.



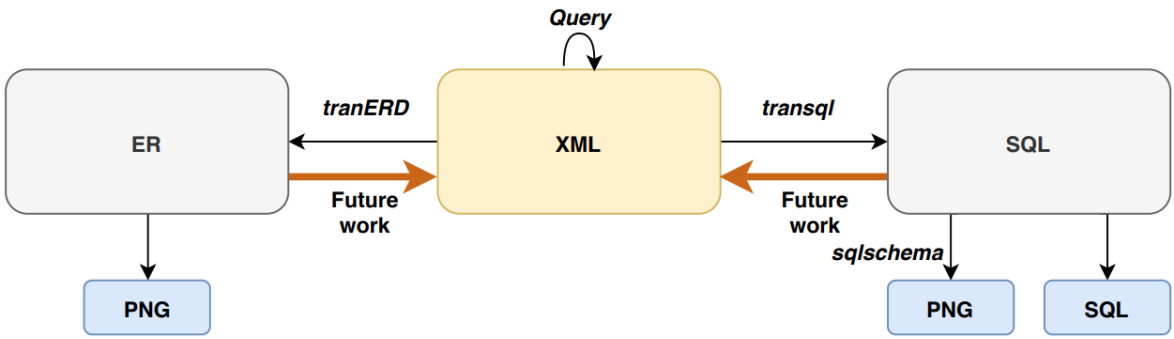
5. Conclusion

In our work, we create an XML schema that can comprehensively describe an ER diagram, and this XML schema is clear and simple for user to create their own XML file. Also we have the functionality of showing the schema of the MySQL database created by taking sql file as input data.

We can generate both the ER diagram and the sql file which are corresponding to the specific XML fitting our XML schema by calling function ***tranERD*** and function ***transql***. We create three query functions for our XML file, including querying related entities of the specific entity, querying entities that have the specific relationship and querying entities that contain the specific attribute. After querying, we can generate both the sub-ER diagram and sql file corresponding to the queried XML file by calling functions we mentioned above.

6. Future work

In the future, we can add more services like transforming sql file into XML since users may want to know the ER diagram of an existing database, and creating an graph user interface that can help users create XML file with less effort. By adding these services, we can make both the transformation between ER diagram and XML and the transformation between XML and sql file bilateral.



7. Reference

[1] S. Xu, L. Yu and S. Lu, "ERDraw: An XML-based ER-diagram Drawing and Translation Tool", Computers and Their Applications, (2003), pp. 143-146.
[2] Mohammad Almseidin, Khaled Alrfou, "μE – Automation Framework", International Journal of Database Theory and Application Vol.8, No.3 (2015), pp.249-258