SDDS

DATABASE DEVELOPMENT ASSIGNMENT

S1637337 | MMI322918 | 28.04.2017

Contents

SECTION 1: DESIGN

a. A relational data model or Document oriented data model for this data. Justify your design decisions.

The requirement for the treatment of the data was as follows:

> *A real estate developer wants to play around with some historical house prices for Glasgow after hearing about Glasgow Open Data at one of the DataFest17 workshops. They have also heard that data could be stored in an SQL or NoSQL database. They want this data in either SQL or NoSQL database so they come to you for help.*

It is clear that the developer is open to using either SQL or noSQL database storage, retrieval and analysis of the historical house price data. As a result, it is important to examine the merits of both systems in order be able to make a recommendation to the developer on how best to deal with the data.

The house price data files are stored on the Glasgow Open Data Portal website in a format known as “csv”, or comma separated values. The issue therefore was how to successfully import these files into either a relational or document oriented database and run queries on data that originated in this format.

One of the essential features of an implementation of a relational database model is that its transactions pass the ACID test – as outlined succinctly by John D Cook (Cook, 2009):

Atomic: Everything in a transaction succeeds or the entire transaction is rolled back.

Consistent: A transaction cannot leave the database in an inconsistent state.

Isolated: Transactions cannot interfere with each other.

Durable: Completed transactions persist, even when servers restart etc.

However, it is clear that the data is not going to be subject to regular transactions – the clue being found in the requirements description when the data was described as "historical house prices". As a result, the ACID test rules can be relaxed as there is not going to be a reliance on the consistency etc. that would be inherent in a relational design. Therefore, a document oriented data model, which does not have ACID as its primary focus – mongoDB, in this case - can now be under consideration.

Importing the data in its csv form into an implementation of a relational database, in this case the criteria being to use the RDBMS MySQL, is not a simple task. External scripts must be used to modify the data, and a knowledge of that scripting language is essential to produce the desired result of data that is ready to insert into a relation. Using a document oriented database model like mongoDB meant that it was possible, with a little user friendly modification of the csv file, to import the data with the "mongoimport.exe" command line (mongoDB, 2017).

However, in order to conduct meaningful queries on the data stored in the hpALL collection, it was easier to use mongoDB's recently introduced built in function "columnsHaveTypes" (mongoDB, 2017) to parse the string data of the csv file to appropriate data types so that it could be manipulated mathematically with the aggregation operator (mongoDB, 2017). This was achieved by reference to the data dictionary file (Stenico Data Services Ltd., 2017) available on the Glasgow Open Data Portal website. The existing top column was deleted manually from the csv files while they were opened in Libre Office Calc, and a superfluous column "omit or use" was itself omitted due to the data dictionary's contention that it was inapplicable to the data in its current state. The csv files were then saved with consistent, short names in the bin folder of the mongoDB installation in order to have them in the right folder for a simpler import statement.

Following import, the collection "hpAll" existed and was now available for querying using the appropriate syntax, as outlined in the mongoDB 3.4 manual.

b. Write appropriate commands to push the data into either MySQL or MongoDB databases.

Import command lines

```
mongoimport.exe --db hpDb --collection hpAll --type csv --fields
"class.string(),streetNo.string(),flatPos.string(),streetName.str
ing(),postCode.string(),monSale.int32(),calYear.int32(),busYear.s
tring(),monYear.string(),calQuarter.string(),salePrice.int64(),rp
i.decimal(),deflator.decimal(),constPrice.decimal(),origBuyer.str
ing(),omitUse.string(),newResale.string(),lhf.string()" --
columnsHaveTypes --file hp11.csv
```

```
mongoimport.exe --db hpDb --collection hpAll --type csv --fields
"class.string(),streetNo.string(),flatPos.string(),streetName.str
ing(),postCode.string(),monSale.int32(),calYear.int32(),busYear.s
tring(),monYear.string(),calQuarter.string(),salePrice.int64(),rp
i.decimal(),deflator.decimal(),constPrice.decimal(),origBuyer.str
ing(),omitUse.string(),newResale.string(),lhf.string()" --
columnsHaveTypes --file hp12.csv
```

```
mongoimport.exe --db hpDb --collection hpAll --type csv --fields
"class.string(),streetNo.string(),flatPos.string(),streetName.str
ing(),postCode.string(),monSale.int32(),calYear.int32(),busYear.s
tring(),monYear.string(),calQuarter.string(),salePrice.int64(),rp
i.decimal(),deflator.decimal(),constPrice.decimal(),origBuyer.str
ing(),omitUse.string(),newResale.string(),lhf.string()" --
columnsHaveTypes --file hp13.csv
```

SECTION 2: QUERIES

a. What was the total house sale over the three years?

Query

```
db.hpAll.findAndModify({
      query: { class: "r" },
      update: { class : "R" } })
db.hpAll.aggregate([
      { $group : { _id : { "Class of housing":  "$class"},
      totalAmount: { $sum[1]: { $add: [ "$salePrice" ] } },
      count: { $sum: 1 } } }
] );
```

Result

```
{ "_id" : { "Class of housing" : "R" }, "Total_House_Sale" :
NumberLong(2025811245), "count" : 15354 }
```

The total house sale over the three years was £2025811245 for 15354 entries.

[1] https://docs.mongodb.com/manual/reference/operator/aggregation/sum/

b. Using the first part of the postcode rank the areas in order of number of sales over the three years.

Query

```
db.hpAll.aggregate([
 { $project : { post_part : { $split[2]: ["$postCode", " "] }} },
  { $unwind : "$post_part" },
  {  $match : { post_part : /\bG/g [3]} },
  { $group : { _id: { "code" : "$post_part" }, count: { $sum:
1 }}},
  { $sort : { count: -1 }}
 ]);
```

Result (Top 5 shown)

```
{ "_id" : { "code" : "G12" }, "count" : 1259 }
{ "_id" : { "code" : "G41" }, "count" : 1144 }
{ "_id" : { "code" : "G42" }, "count" : 1067 }
{ "_id" : { "code" : "G13" }, "count" : 900 }
{ "_id" : { "code" : "G11" }, "count" : 887 }
```

Using the first part of the postcode, **G12, G41, G42, G13** and **G11** are the top 5 results for number of sales over the 3 years in the collection.

[2] https://docs.mongodb.com/manual/reference/operator/aggregation/split/
[3] https://www.w3schools.com/jsref/jsref_regexp_begin.asp

c. Which postcode has the most expensive houses?

Query

```
db.hpAll.aggregate([

...   { $group : { _id: { "PostCode" :
"$postCode" },TotalSalesInPounds : {$sum : {$add :
[ "$salePrice"] }}, count : { $sum: 1 }}},

...  {$project: {TotalSalesInPounds:1, count: 1,
MostExpensiveHouses: { $divide[4]: [ "$TotalSalesInPounds",
"$count" ]}}},

...    { $sort : { MostExpensiveHouses: -1 }}

...  ]);
```

Result

```
{ "_id" : { "PostCode" : "G43 1TT" }, "TotalSalesInPounds" :
NumberLong(995000), "count" : 1, "MostExpensiveHouses" : 995000 }
```

Based on the data, G43 1TT has the most expensive house (count: 1). Reporting on this data reveals the truth of the entries in the database, but local knowledge of surrounding postcodes indicates that incorrect data for the sale price of this property may have been entered into the database. Queries on the database can only ever result in a filtered report of its contents, and is not designed to flag up anomalies.

[4] http://stackoverflow.com/questions/22819303/mongodb-aggregation-divide-computed-fields

SECTION 3: COMPARISON

a. Describe the main differences between a documented oriented and relational database design

Relational database design is an implementation of the principles of relational algebra, as defined and then refined by mathematician E.F. Codd from the 1970s on. As a result, while it may be a misapprehension and a simplification, SQL and its ilk are seen as an attempt to map Codd's logical "relations" to the perception of tables, and to map a theoretical attribute and tuple to an actual representation of a column and row in this representational table. Having accepted this simplification, we can then describe a typical relational database design as being one that is made up of a large clump of data that is then subject to a schema. This schema is based on the requirements gathering for what type of data the database will store and what constraints are to be placed on the insertion of the data, to guarantee integrity and to ensure that the data reflects real life conditions. The schema also delineates how the data shall be split up before insertion and the keys - e.g. the concept of a primary key as spawned from the relational theory of a candidate key that will enable the tables to be joined. Once the data has been successfully inserted, queries can be run on the data.

In contrast, document oriented design often claims to be schema-less, a facet of their ad-hoc nature that is trumpeted as a positive. C.J Date (Date, 2009) describes one of the features of relational database design that does not figure in document oriented design:

"Physical data independence...means we have the freedom to make changes in the way the data is physically stored and accessed without having to make corresponding changes in the way the data is perceived by the user. The reason we might want to change those storage and access specifics is, typically, performance; and the fact that we can make such changes without having to change the way the data looks to the user means that existing programs, queries, and the like can all still work after the change."

In the document oriented design, the perceived need for a great deal of physical storage often informs the choice of this design over a relational database design, so the design

can actually be seen as physically dependent, without the abstraction that a relational design engenders. A document oriented design - one of the database designs that comes under the heading of noSQL - uses what are known as "documents" to represent data. In the case of the document oriented mongoDB database, a "document" consists of 'field:value' pairs . The collective noun for a set of these "documents" is a "collection". Using Brewer's CAP theorem (Cook, 2009), many implementations of a relational design are **C**a**P**, the adherence to ACID resulting in a trade-off of Availability for Consistency and Partition Tolerance. The document oriented response is c**AP**, with Consistency traded off for Availability and Partition Tolerance - BASE not ACID.

b. Describe the main differences between a documented oriented and relational database queries.

In a purportedly relational database design, relational algebra is transmuted to user friendly languages like the SQL flavour of MySQL. As a result, these database specific languages can be used to perform operations on data with the backing of the logic from the relational algebra. As a structured query language, the language was specifically designed to be used for the purpose of manipulating and querying a relational database implementation.

For a document oriented design, queries must be more powerful as the lack of a schema means that the application level has the responsibility to guarantee accurate and meaningful query results. Unlike SQL flavours, the document oriented design has the potential to have different application front-ends, meaning that it is not restricted to one query language, with the possibility that SQL itself could be used through a translator/converter to produce queries for a document oriented design like mongoDB. This means that the queries are not limited to those with a knowledge of JavaScript functions etc. but instead admit of the fact that it truly is **Not Only** SQL and not **NO** SQL.

References

Cook, J. D., 2009. *John D. Cook: Applied Mathematical Consulting.* [Online]
Available at: https://www.johndcook.com/blog/2009/07/06/brewer-cap-theorem-base/
[Accessed 25 4 2017].

Date, C., 2009. Model vs. Implementation. In: *SQL and relational theory: how to write accurate SQL code.* 1st ed. Sebastopol, California: O'Reilly Media, Inc.

mongoDB, 2017. *Aggregation Pipeline Operators.* [Online]
Available at: https://docs.mongodb.com/manual/reference/operator/aggregation/
[Accessed 25 4 2017].

mongoDB, 2017. *mongoimport - mongoDB Manual 3.4.* [Online]
Available at:
https://docs.mongodb.com/manual/reference/program/mongoimport/#cmdoption-columnshavetypes
[Accessed 25 4 2017].

Selko, J., 2013. *Joe Celko's Complete Guide to NoSQL.* s.l.:Elsevier.

Stenico Data Services Ltd., 2017. *Glasgow House Sales - 1991 - 2013 - House Price Data Dictionary.csv.* [Online]
Available at: https://data.glasgow.gov.uk/dataset/glasgow-house-sales-1991-2013/resource/4d2baac2-c786-4343-bf60-e3194ac9e237
[Accessed 26 4 2017].