COMP1204: Database Theory and Practice Coursework

Huw Jones
27618153

April 17, 2016

1 ERD and Normalisation

1.1 EX1 - Relation

```
HotelReview(
  ReviewID:Integer,
  Author:String,                Date:Date,
  HotelID:Integer,              URL:String,
  AveragePrice:Integer,         Content:String,
  Overall:Integer,              OverallRating:Integer,
  BusinessService:Integer,      CheckIn:Integer,
  Cleanliness:Integer,          Location:Integer,
  Rooms:Integer,                Service:Integer,
  Value:Integer,                NoReaders:Integer,
  NoHelpful:Integer
)
```

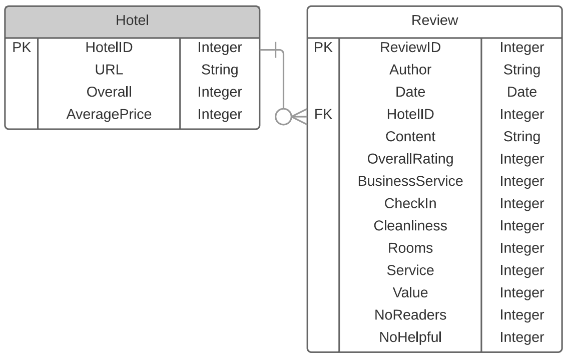
1.2 EX2 - Functional Dependencies

Author	Date	HotelName	→	Content	OverallRating	BusinessService	CheckIn
				Cleanliness	Location	Rooms	Service
				Value	NoReaders	NoHelpful	
HotelID			→	URL	Overall	AveragePrice	

1.3 EX3 - Normalised Relations

```
Hotel(
  HotelID:Integer,              URL:String,
  OverallRating:Integer,        AveragePrice:Integer
)
Review(
  ReviewID:Integer,             Author:String,
  Date:Date,                    HotelID:Integer,
  Content:String,               Overall:Integer,
  BusinessService:Integer,      CheckIn:Integer,
  Cleanliness:Integer,          Rooms:Integer,
  Service:Integer,              Value:Integer,
  NoReaders:Integer,            NoHelpful:Integer
)
```

1.4 EX4 - ERD Model



2 Relation Algebra

2.1 EX5 - Finding a user's reviews

$\sigma_{\text{author}=X}(\text{Review})$

2.2 EX6 - Finding users with more than two reviews

$\Pi_{\text{author,noReviews}}\sigma_{\text{noReviews}>2}\gamma\text{author; COUNT}(*) \rightarrow \text{noReviews}(\text{Review})$

2.3 EX7 - Finding all hotels with more than 10 reviews

$\Pi_{\text{hotelID,noReviews}}\sigma_{\text{noReviews}>10}\gamma\text{hotelID; COUNT}(*) \rightarrow \text{noReviews}(\text{Review})$

2.4 EX8 - Finding all hotels with overall rating and cleanliness

$\Pi_{\text{hotelID,avgOverall,avgCleanliness}}\sigma_{\text{avgOverall}>3 \text{ and avgCleanliness}\geq 5}\gamma\text{hotelID; AVG(overall)} \rightarrow \text{avgOverall, AVG(cleanliness)} \rightarrow \text{avgCleanliness(Review)}$

3 SQL

3.1 EX9 - Creating HotelReviews Table

```sql
CREATE TABLE HotelReviews (
   reviewID INTEGER PRIMARY KEY,
   author VARCHAR(256) NOT NULL,
   reviewDate DATE NOT NULL,
   hotelID INTEGER NOT NULL,
   URL VARCHAR(256),
   averagePrice INTEGER,
   content TEXT,
   overall INTEGER NOT NULL,
   overallRating INTEGER NOT NULL,
   businessService INTEGER,
   checkIn INTEGER,
   cleanliness INTEGER,
   location INTEGER,
   rooms INTEGER,
   service INTEGER,
   value INTEGER,
   noReaders INTEGER NOT NULL DEFAULT 0,
   noHelpful INTEGER NOT NULL DEFAULT 0
);
```

Please note, I have deliberately avoided using `AUTOINCREMENT` on the "reviewID" column. This is because the SQLite documentation specifically recommends not using this keyword as it "should be avoided if not strictly needed".

3.2 EX10 - Creating a SQL insert script

Please see Appendix A.1 and A.2 for the Unix code (note the script is split into 2 sections. The first, generatesql.sh, the second generatesql.awk. This was done as lstlisting had trouble syntax highlighting the big awk script in a bash script.) I chose to implement my script mainly in awk. As awk supports record/field processing, it was just the case of getting it to correctly identify the records and fields. Once it could identity the records, processing the fields was as simple as looping through the fields, stripping the tags and building a 2D array. Finally, once the data was put into the 2D array, it then loops back over the data and creates the insert statement.

3.3 EX11 - Creating Normalised Tables

```sql
CREATE TABLE Hotels (
   hotelID INTEGER PRIMARY KEY,
   URL VARCHAR(256) NOT NULL,
   overallRating INTEGER NOT NULL,
   averagePrice INTEGER
);
CREATE TABLE Reviews (
   reviewID INTEGER PRIMARY KEY,
   author VARCHAR(256) NOT NULL,
   reviewDate DATE NOT NULL,
   hotelID INTEGER NOT NULL,
   content TEXT,
   overall INTEGER NOT NULL,
   businessService INTEGER,
   checkIn INTEGER,
   cleanliness INTEGER,
   location INTEGER,
   rooms INTEGER,
   service INTEGER,
   value INTEGER,
   noReaders INTEGER NOT NULL DEFAULT 0,
   noHelpful INTEGER NOT NULL DEFAULT 0,
   FOREIGN KEY (hotelID) REFERENCES Hotels(hotelID)
);
```

3.4 EX12 - Populating Normalised Tables

```
INSERT INTO Hotels
(hotelID, URL, overallRating, averagePrice)
SELECT hotelID, URL, overallRating, averagePrice
FROM HotelReviews
GROUP BY hotelID;
```

By grouping by "hotelID", this prevents duplicate inserts.

```
INSERT INTO Reviews
(author, reviewDate, hotelID, content, overall, businessService, checkIn, cleanliness, location, rooms
    , service, value, noReaders, noHelpful)
SELECT author, reviewDate, hotelID, content, overall, businessService, checkIn, cleanliness, location,
    rooms, service, value, noReaders, noHelpful
FROM HotelReviews;
```

3.5 EX13 - Creating Indexes

```
CREATE INDEX hotelID ON Reviews(hotelID);
CREATE INDEX author on Reviews(author);
```

Index "hotelID" was chosen because it is the Foreign Key constraint field. In order to speed up queries that use this constraint, this field needs to be indexed.

Index "author" was chosen because many queries will most likely use the "author" field. Indexing this column speeds to operations.

Using `.timer on`, I timed how long it took to execute each query with and without the index. With the index, operations were about 25% quicker.

4 Data Retrieval and Analysis

4.1 EX14 - Relational Algebra to SQL

4.1.1 EX5 - Finding a user's reviews

```
SELECT * FROM Reviews WHERE author=?;
```

4.1.2 EX6 - Finding users with more than two reviews

```
SELECT author, COUNT(*) as noReviews
FROM Reviews
GROUP BY author
HAVING noReviews > 2;
```

4.1.3 EX7 - Finding all hotels with more than 10 reviews

```
SELECT hotelID, COUNT(*) as noReviews
FROM Reviews
GROUP BY hotelID
HAVING noReviews > 10;
```

4.1.4 EX8 - Finding all hotels with overall rating and cleanliness

```
SELECT hotelID
FROM Reviews
GROUP BY hotelID
HAVING AVG(overall) > 3 AND AVG(cleanliness) >= 5;
```

5 Conclusions

Overall, the I felt I did not struggle much with the coursework. It took some reasonable thought as to how to make the `generatesql.sh` script as efficient as possible. I decided that I would try to limit the amount of programs I called and stick to as few environments as possible. Overall, this made my script execute very quickly (for processing $\approx$ 150MB of review data).

Due to my understanding of SQL vs Relational Algebra, I completed EX14 and then reverse engineered the queries back to relational algebra. I found this method of producing relational algebra much more efficient and pleasing as I understood more what the queries were doing.

Appendices

A Unix Code

A.1 generatesql.sh

```
#!/bin/bash
if [ $# -ne 1 ]
then
  echo "No␣argument␣passed␣to␣script.";
  exit 1;
fi

# Extracts the HotelID from a hotel file name
# @param $1 Hotel File name
function getHotelID() {
  echo "$1" | sed -e 's:^.*\/::' -e 's:.dat::' -e 's:hotel_::'
}

# Returns the table schema
function createTable() {
  echo "PRAGMA␣encoding␣=␣\"UTF-8\";"
  echo "DROP␣TABLE␣IF␣EXISTS␣HotelReviews;"
  echo "CREATE␣TABLE␣HotelReviews␣("
  echo "␣␣reviewID␣INTEGER␣PRIMARY␣KEY,"
  echo "␣␣author␣VARCHAR(256)␣NOT␣NULL,"
  echo "␣␣reviewDate␣DATE␣NOT␣NULL,"
  echo "␣␣hotelID␣INTEGER␣NOT␣NULL,"
  echo "␣␣URL␣VARCHAR(256),"
  echo "␣␣averagePrice␣INTEGER,"
  echo "␣␣content␣TEXT,"
  echo "␣␣overall␣INTEGER␣NOT␣NULL,"
  echo "␣␣overallRating␣INTEGER␣NOT␣NULL,"
  echo "␣␣businessService␣INTEGER,"
  echo "␣␣checkIn␣INTEGER,"
  echo "␣␣cleanliness␣INTEGER,"
  echo "␣␣location␣INTEGER,"
  echo "␣␣rooms␣INTEGER,"
  echo "␣␣service␣INTEGER,"
  echo "␣␣value␣INTEGER,"
  echo "␣␣noReaders␣INTEGER␣NOT␣NULL␣DEFAULT␣0,"
  echo "␣␣noHelpful␣INTEGER␣NOT␣NULL␣DEFAULT␣0,"
  echo "␣␣FOREIGN␣KEY␣(hotelID)␣REFERENCES␣Hotels(hotelID)"
  echo ");"
}

# Processes an individual hotel file
# @param $1 Filename of hotel file
function processHotel() {
  hotelID=$(getHotelID $1)
  tr -d '\r' < $1 | awk \
    -v hotelID="$hotelID" --exec=generatesql.awk
}

# Prints out a progress bar
# @param $1 Current iteration number
# @param $2 Number of iterations
function printProgress() {
  awk '
  BEGIN {
    percentage = ( '$1' / '$2');
    numberHashes = ( percentage * 50 );
    hashString = "";
    for(i = 1; i < numberHashes; i++){
      hashString = hashString "#";
    }
    printf("\rProgress␣[%-50s]␣(%.2f%)", hashString, ( percentage * 100 ));
  }'
}

# Returns a field from a string
# @param $1 String
# @param $2 Field Name to filter
function getField() {
  grep "$2" $1 | sed -e "s:$2::"
}

# Create the table
```

```
echo "$(createTable)" > hotelreviews.sql

# If the file is a directory, then iterate over the directory
if [ -d $1 ]
then
  fileCount=$(ls -l $1 | wc -l)
  counter=0
  for f in $1/*
  do
    echo "$(processHotel␣$f)" >> hotelreviews.sql
    counter=$((counter+1))
    printProgress $counter $fileCount
  done
  printProgress $fileCount $fileCount
else
  # Otherwise process one file (useful for testing files that break the script)
  echo -e "$(processHotel␣$1)" >> hotelreviews.sql
fi

echo -ne "\n"
```

A.2 generatesql.awk

```
BEGIN {
  # This allows us to read the file as a series of records separated by blank lines.
  # The fields are deliminated by newlines (\n)
  RS = "";       # Set record separator to ""
  FS = "\n";       # Set field separator to "\n"
  recordNum = 0;     # Set record counter to 0
}

# Checks if a field has a value of -1, and if so, returns 0.
# @param field Field to zero check
function zeroCheck(field){
  if(field == -1){
    return 0;
  } else {
    return field;
  }
}

# Checks if a field has a value of -1, and if so, returns null.
# @param field Field to null check
function nullCheck(field){
  if(field == -1){
    return "NULL";
  } else {
    return field;
  }
}

# Escapes a field to prevent SQL errors
function escapeField(field){
  gsub(/"/,␣"\"\"", field);
  return field;
}

# Formats the record into a SQL insert statement
# @param rowNumber Row (record) number of the row to format
function formatRow(rowNumber){
  insert = "INSERT␣INTO␣HotelReviews␣(author,␣reviewDate,␣hotelID,␣URL,␣averagePrice,␣overallRating,␣
      content,␣overall,␣businessService,␣checkIn,␣cleanliness,␣location,␣rooms,␣service,␣value,␣
      noReaders,␣noHelpful)␣VALUES␣(";
  insert = insert "\"" data[rowNumber "@author"]"\",␣";
  insert = insert data[rowNumber "@date"] ",␣";
  insert = insert hotelID ",␣";
  insert = insert "\"" URL "\",␣";
  insert = insert nullCheck(avgPrice) ",␣";
  insert = insert overallRating ",␣";
  insert = insert "\"" escapeField(data[rowNumber "@content"]) "\",␣";
  insert = insert data[rowNumber "@overall"] ",␣";
  insert = insert nullCheck(data[rowNumber "@business"]) ",␣";
  insert = insert nullCheck(data[rowNumber "@checkin"]) ",␣";
  insert = insert nullCheck(data[rowNumber "@cleanliness"]) ",␣";
  insert = insert nullCheck(data[rowNumber "@location"]) ",␣";
  insert = insert nullCheck(data[rowNumber "@rooms"]) ",␣";
  insert = insert nullCheck(data[rowNumber "@service"]) ",␣";
  insert = insert nullCheck(data[rowNumber "@value"]) ",␣";
  insert = insert zeroCheck(data[rowNumber "@readers"]) ",␣";
  insert = insert zeroCheck(data[rowNumber "@helpful"]);
  insert = insert ");";
  return insert;
}

{
  # Loop through all fields in record. NF is number of fields
  for (i = 1; i <= NF; i++){
    if(recordNum == 0){              # Get hotel properties (first, or 0th record)
      if(match($i, "<Overall␣Rating>")){
        gsub(/<Overall Rating>/, "", $i);
        overallRating = $i;
      } else if(match($i, "<Avg.␣Price>")){
        gsub(/<Avg. Price>\$/, "", $i);
        # Remove thousand separator
        gsub(/,/, "", $i);
        # Check to see if avg price is not "Unknown" (note, it is spelt correctly here, but the if
            will check for strings)
        if( $i + 0 != $i ){
          avgPrice = -1;
        } else {
          avgPrice = $i;
        }
      } else if(match($i, "<URL>")){
        gsub(/<URL>/, "", $i);
        URL = $i;
      }
    }
    # Get the record fields
    if(match($i, "<Author>")){          # Author
      gsub("<Author>", "", $i);
      data[recordNum "@author"] = $i;
    } else if(match($i, "<Date>")){       # Date (format to SQL yyyy-mm-dd)
      gsub(/<Date>/, "", $i);
      cmd = "date␣\"+%Y-%m-%d\"␣-d␣\"$i\"";
      cmd | getline date;
      data[recordNum "@date"] = date;
      close(cmd);
    } else if(match($i, "<Overall>")){      # Overall Score
      gsub("<Overall>", "", $i);
      data[recordNum "@overall"] = $i;
    } else if(match($i, "<Business␣service>")){ # Business Service
      gsub(/<Business service>/, "", $i);
      data[recordNum "@business"] = $i;
    }  else if(match($i, "<Content>")){     # Content
      gsub(/<Content>/, "", $i);
      data[recordNum "@content"] = $i;
    } else if(match($i, "<Check␣in␣/␣front␣desk>")){  # Check In
      gsub(/<Check in \/ front desk>/, "", $i);
      data[recordNum "@checkin"] = $i;
    } else if(match($i, "<Cleanliness>")){    # Cleanliness
      gsub(/<Cleanliness>/, "", $i);
      data[recordNum "@cleanliness"] = $i;
    } else if(match($i, "<Location>")){     # Location
      gsub(/<Location>/, "", $i);
      data[recordNum "@location"] = $i;
    } else if(match($i, "<Rooms>")){      # Rooms
      gsub(/<Rooms>/, "", $i);
      data[recordNum "@rooms"] = $i;
    } else if(match($i, "<Service>")){      # Service
      gsub(/<Service>/, "", $i);
      data[recordNum "@service"] = $i;
    } else if(match($i, "<Value>")){      # Value
      gsub(/<Value>/, "", $i);
      data[recordNum "@value"] = $i;
    } else if(match($i, "<No.␣Reader>")){   # Number of Readers
      gsub(/<No. Reader>/, "", $i);
      data[recordNum "@readers"] = $i;
    } else if(match($i, "<No.␣Helpful>")){    # Number of Helpful
      gsub(/<No. Helpful>/, "", $i);
      data[recordNum "@helpful"] = $i;
    }
  }
  if(recordNum != 0){
    print formatRow(recordNum);
  }
  recordNum++;
}
```