COMP1204: Database Theory and Practice Coursework

Huw Jones
27618153

April 15, 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,          Rooms:Integer,
  Service:Integer,              Value:Integer,
  NoReaders:Integer,            NoHelpful:Integer
)
```

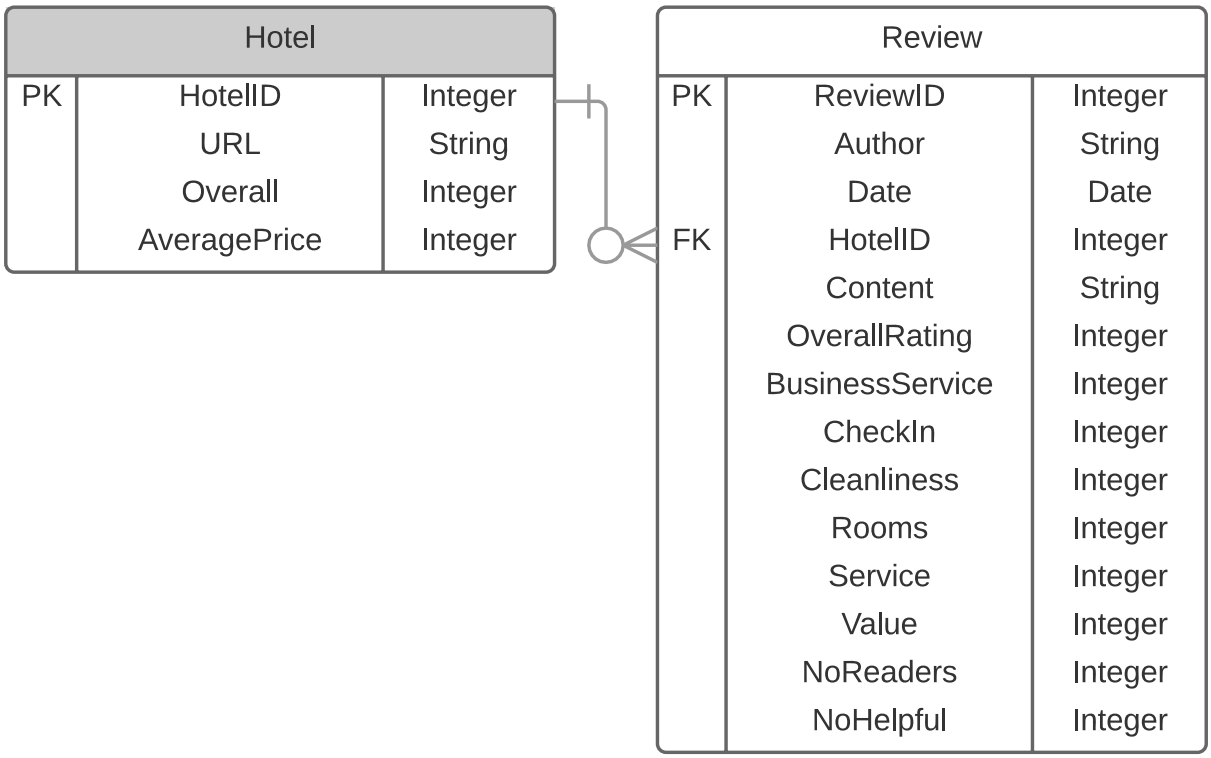
1.2 EX2 - Functional Dependencies

Author	Date	HotelName	→	Content	OverallRating	BusinessService	CheckIn
				Cleanliness	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

$\sigma_{Author=X}(Review)$

2.2 EX6 - Finding users with more than two reviews

$\sigma(Review)$

2.3 EX7 - Finding all hotels with more than 10 reviews

2.4 EX8 - Finding all hotels with overall rating and cleanliness

3 SQL

3.1 EX9 - Creating HotelReviews Table

```
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,
  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 **??** for the Unix code. 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.

3.3 EX11 - Creating Normalised Tables

3.4 EX12 - Populating Normalised Tables

3.5 EX13 - Creating Indexes

4 Data Retrieval and Analysis

4.1 EX14 - Relational Algebra to SQL

5 Conclusions

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 "  rooms INTEGER,"
```

```
  echo "  service INTEGER,"
  echo "  value INTEGER,"
  echo "  noReaders INTEGER NOT NULL DEFAULT 0,"
  echo "  noHelpful INTEGER NOT NULL DEFAULT 0"
  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" -E 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)
  # Set record separator to ""
  RS = "";
  # Set field separator to "\n"
  FS = "\n";
  # Set record counter to 0
  recordNum = 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, 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]["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++){
    # Get hotel properties (first, or 0th record)
    if(recordNum == 0){
      if(match($i, "<Overall Rating>")){
        sub(/<Overall Rating>/, "", $i);
        overallRating = $i;
      } else if(match($i, "<Avg. Price>")){
        sub(/<Avg. Price>\$/, "", $i);
        # Remove thousand separator
        gsub(/,/, "", $i);
        # Check to see if avg price is not "Unknown" (note, it's spelt correctly here,
            but the if will check for strings)
        if( $i + 0 != $i ){
          avgPrice = -1;
        } else {
          avgPrice = $i;
        }
      } else if(match($i, "<URL>")){
        sub(/<URL>/, "", $i);
        URL = $i;
      }
    }
    # Get the record fields
    if(match($i, "<Author>")){          # Author
      sub(/<Author>/, "", $i);
      data[recordNum]["author"] = $i;

    } else if(match($i, "<Date>")){       # Date (format to SQL yyyy-mm-dd)
      sub(/<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
      sub(/<Overall>/, "", $i);
      data[recordNum]["overall"] = $i;

    } else if(match($i, "<Business service>")){ # Business Service
      sub(/<Business service>/, "", $i);
      data[recordNum]["business"] = $i;

    }  else if(match($i, "<Content>")){     # Content
      sub(/<Content>/, "", $i);
      data[recordNum]["content"] = $i;

    } else if(match($i, "<Check in / front desk>")){  # Check In
      sub(/<Check in \/ front desk>/, "", $i);
      data[recordNum]["checkin"] = $i;

    } else if(match($i, "<Cleanliness>")){      # Cleanliness
      sub(/<Cleanliness>/, "", $i);
      data[recordNum]["cleanliness"] = $i;

    } else if(match($i, "<Rooms>")){      # Rooms
      sub(/<Rooms>/, "", $i);
      data[recordNum]["rooms"] = $i;

    } else if(match($i, "<Service>")){      # Service
      sub(/<Service>/, "", $i);
      data[recordNum]["service"] = $i;

    } else if(match($i, "<Value>")){      # Value
      sub(/<Value>/, "", $i);
      data[recordNum]["value"] = $i;

    } else if(match($i, "<No. Reader>")){   # Number of Readers
      sub(/<No. Reader>/, "", $i);
      data[recordNum]["readers"] = $i;
```

```
    } else if(match($i, "<No. Helpful>")){    # Number of Helpful
      sub(/<No. Helpful>/, "", $i);
      data[recordNum]["helpful"] = $i;

    }
  }
  recordNum++;
}

END {
  # Loop over the data and format the rows into the INSERT statement
  for(record in data) {
    print formatRow(record);
  }
}
```