

# 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  
)
```

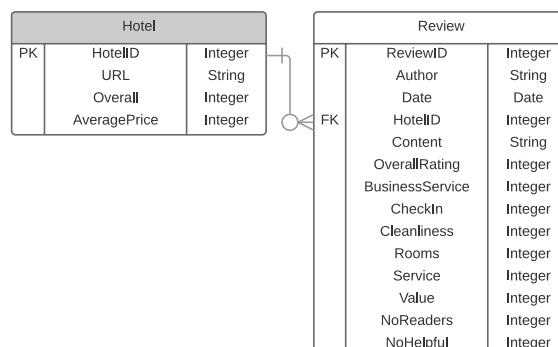
### 1.2 EX2 - Functional Dependencies

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

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

### 2.2 EX6 - Finding users with more than two reviews

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

### 2.3 EX7 - Finding all hotels with more than 10 reviews

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

### 2.4 EX8 - Finding all hotels with overall rating and cleanliness

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

## 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,
  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 identify 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

```
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);
CREATE INDEX params on Reviews(overall, businessService, checkIn, location, rooms, service, value);

```

## 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

# 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" -E generate_sh.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 delimited by newlines (\n)
    RS = "\n";      # Set record separator to "\n"
    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 rowNum Row (record) number of the row to format
function formatRow(rowNum){
    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[rowNum]["author"] "\" , ";
    insert = insert data[rowNum]["date"] " , ";
    insert = insert hotelID " , ";
    insert = insert "\" " URL "\" , ";
    insert = insert nullCheck(avgPrice) " , ";
    insert = insert overallRating " , ";
    insert = insert "\" " escapeField(data[rowNum]["content"]) "\" , ";
    insert = insert data[rowNum]["overall"] " , ";
    insert = insert nullCheck(data[rowNum]["business"]) " , ";
    insert = insert nullCheck(data[rowNum]["checkin"]) " , ";
    insert = insert nullCheck(data[rowNum]["cleanliness"]) " , ";
    insert = insert nullCheck(data[rowNum]["location"]) " , ";
    insert = insert nullCheck(data[rowNum]["rooms"]) " , ";
    insert = insert nullCheck(data[rowNum]["service"]) " , ";
    insert = insert nullCheck(data[rowNum]["value"]) " , ";
    insert = insert zeroCheck(data[rowNum]["readers"]) " , ";
    insert = insert zeroCheck(data[rowNum]["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>")){
                sub(/<Overall_Rating>/, "", $i);
                overallRating = $i;
            } else if(match($i, "<Avg. Price>")){
                sub(/<Avg. Price>$/, "", $i);
                # Remove thousand separator
            }
        }
        formatRow(i);
        recordNum++;
    }
}

```

```

    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>")){
    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 -n +%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, "<Location>")){    # Location
    sub(/<Location>/, "", $i);
    data[recordNum]["location"] = $i;
} else if(match($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);
    }
}

```