COMP1204: DB Coursework

Jonathan Keable
ID: 25616196
email: jek1g15@ecs.soton.ac.uk
10th April 2016

1 Relational Model and ERD

The Relation R1 of the review can be defined as follows:

Attribute	**Data Type**
ReviewID	INTEGER
HotelID	INTEGER
Hotel Average Overall	REAL
Hotel Average Price	INTEGER
Hotel URL	STRING
UserID	STRING
Content	STRING
Date	STRING
OverallRating	INTEGER
Value	INTEGER
Rooms	INTEGER
Location	INTEGER
Cleanliness	INTEGER
CheckIn/FrontDesk	INTEGER
Service	INTEGER
BusinessService	INTEGER
NoReaders Review	INTEGER
NoHelpful Review	INTEGER

Here the ReviewID and HotelID are the primary key, with review IDs being unique within a hotels set of reviews.
The Main Functional Dependency of this relation would be:

{ReviewID,HotelID} $\rightarrow$ {All other attributes}

If a user can only post one review per Hotel then:

{UserID, HotelID} $\rightarrow$ {All other attributes}

Or perhaps if a user can only post one review for a hotel each day then:

{UserID, Date, HotelID} $\rightarrow$ {All other attributes}

However because multiple reviews can be places as a guest under the name "Trip Advisor Member" these dependencies in fact do not work, so we have to use unique review IDs instead.

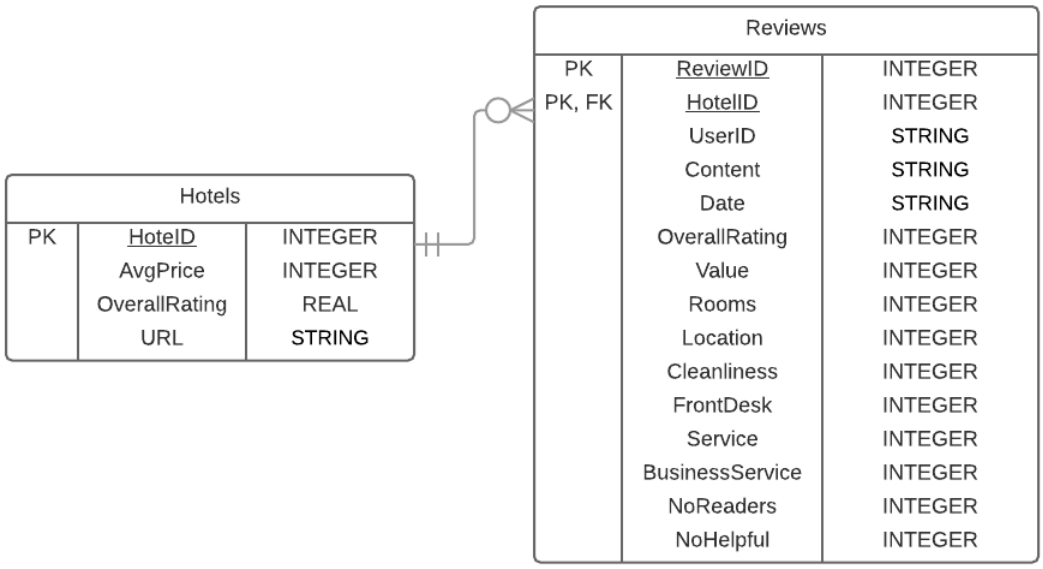
Also,

{HotelID} → {Hotel Average Overall, Hotel Average Price, Hotel URL}

In order to make the relation BCNF we should move these attributes into another relation:

Attribute	**Data Type**
HotelID	INTEGER
Hotel Average Overall	REAL
Hotel Average Price	INTEGER
Hotel URL	STRING

And the HotelID can be a Foreign Key in the original relation. The schema of this relation would be: Hotels(**int:** HotelID, **int:** AverageOverall, **int:** AveragePrice, **string:** URL).
The First relation with these attributes removed other than the HotelID will be known as Reviews.
Now we have normalised the relation, we can create an ERD:



2 Relational Algebra

2.1 Table Creation

Finds all reviews by the same user, where x = UserID.

$$R = \sigma_{username \equiv x}(Reviews) \tag{1}$$

Finds UserID and number of reviews by users with more than 2 reviews.

$$R = \pi_{UserID,noReviews}(\gamma_{UserID,COUNT(UserID)\rightarrow noReviews(Reviews)} > 2) \tag{2}$$

Finds Hotels with move than 10 reviews.

$$R = \pi_{HotelID,HotelOverall,HotelAvgPrice,HotelURL} (\gamma_{HotelID,COUNT(HotelID)\rightarrow noReviews(Reviews \bowtie Hotels)} > 10) \tag{3}$$

Finds Hotels with overall rating greater than 3 and average cleanliness greater than or equal to 5.

$$R = \{\pi_{HotelID,HotelOverall,HotelAvgPrice,HotelURL}(\sigma_{HotelAvgOverall>3}(Hotels \bowtie Reviews))\} \cap \{\gamma_{HotelID,AVG(Cleanliness)\rightarrow avgMark(Hotels \bowtie Reviews)} \geq 5\} \tag{4}$$

3 SQL Queries

The main table in to which values are read:

```
CREATE TABLE HotelReviews
(
REVIEWID INT NOT NULL,
HOTELID INT NOT NULL,
USERID TEXT NOT NULL,
CONTENT TEXT,
DATE TEXT NOT NULL,
OVERALL_RATING INT NOT NULL,
VALUE INT,
ROOMS INT,
LOCATION INT,
CLEANLINESS INT,
CHECKIN_FRONTDESK INT,
SERVICE INT,
BUSINESS_SERVICE INT,
NO_READERS INT,
NO_HELPFUL INT,
HOTEL_OVERALL REAL NOT NULL,
HOTEL_AVG_PRICE INT NOT NULL,
HOTEL_URL TEXT,
PRIMARY KEY (REVIEWID, HOTELID)
);
```

The normalised versions of the tables:

```
CREATE TABLE Reviews
(
REVIEWID INT NOT NULL,
HOTELID INT NOT NULL,
USERID TEXT NOT NULL,
CONTENT TEXT,
DATE TEXT NOT NULL,
OVERALL_RATING INT NOT NULL,
VALUE INT,
```

```
ROOMS INT,
LOCATION INT,
CLEANLINESS INT,
CHECKIN_FRONTDESK INT,
SERVICE INT,
BUSINESS_SERVICE INT,
NO_READERS INT,
NO_HELPFUL INT,
PRIMARY KEY (REVIEWID, HOTELID)
);

CREATE TABLE Hotels
(
HOTELID INT PRIMARY KEY NOT NULL,
HOTEL_OVERALL REAL NOT NULL,
HOTEL_AVG_PRICE INT NOT NULL,
HOTEL_URL TEXT
);
```

To copy the data into the normalised tables I used the following SQL commands:

```
INSERT INTO Reviews (REVIEWID, HOTELID, USERID ,CONTENT, DATE, OVERALL_RATING, VALUE, ROOMS, LO
SELECT REVIEWID, HOTELID, USERID ,CONTENT, DATE, OVERALL_RATING, VALUE, ROOMS, LOCATION, CLEANI
FROM HotelReviews;
```

Above Only the required fields are selected.

```
INSERT INTO Hotels (HOTELID, HOTEL_OVERALL, HOTEL_AVG_PRICE, HOTEL_URL)
SELECT HOTELID, HOTEL_OVERALL, HOTEL_AVG_PRICE, HOTEL_URL
FROM HotelReviews
GROUP BY HOTELID;
```

In this statement the reviews were grouped by hotel as we only need the attributes related to the hotel, not the review, and each hotel must only appear once as it is the primary key.
Useful indexes would be HotelID and ReviewID in the Reviews table as these are the primary keys, and a HotelID index in the Hotels table where it is also a primary key. One may also wish to index by UserID in the reviews table, as you may wish to aggregate information about users such as how many reviews they have. A HotelOverall index could be useful in order to quickly compare Groups of Hotels with a certain overall rating. An example of an index creation statement would be:

```
CREATE INDEX OverallRatings
ON Hotels (HOTEL_OVERALL);
```

3.1 Data Retrieval and Analysis

Select rows where userID = ”x”:

```
SELECT * FROM Reviews
WHERE USERID = 'x';
```

Select UserID and Number of Reviews where a user has more than 2 reviews:

```
SELECT USERID, COUNT(USERID) FROM Reviews
GROUP BY USERID
HAVING COUNT(USERID) > 2;
```

Select Hotels with more than 10 reviews:

```
SELECT Hotels.* FROM Hotels
INNER JOIN Reviews
ON Hotels.HOTELID = Reviews.HOTELID
GROUP BY Reviews.HOTELID
HAVING COUNT(Reviews.HOTELID) > 10;
```

Select Hotels with overall rating greater than 3 and average cleanliness greater than or equal to 5:

```
SELECT * FROM Hotels
WHERE HOTEL_OVERALL > 3
INTERSECT
SELECT Hotels.* FROM Hotels
INNER JOIN Reviews
ON Hotels.HOTELID = Reviews.HOTELID
GROUP BY Reviews.HOTELID
HAVING AVG(Reviews.CLEANLINESS) >= 5;
```