PORTFOLIO ASSIGNMENT

June 2021

CMI7509-2021

Carl Wilson [U0370630]

u0370630@unimail.hud.ac.uk

1. POSTGRES TUTORIAL

1.1 Introduction

This case study focusses on using a relational database, Postgres, for a hotel room booking system as the data is structured, predictable, requires a high level of integrity [nobody wants to be double booked!] and highly transactional in nature.

The case study will work through the schema, or database layout and design, before illustrating how to create the tables, provide some sample data and then run several typical queries that might be required.

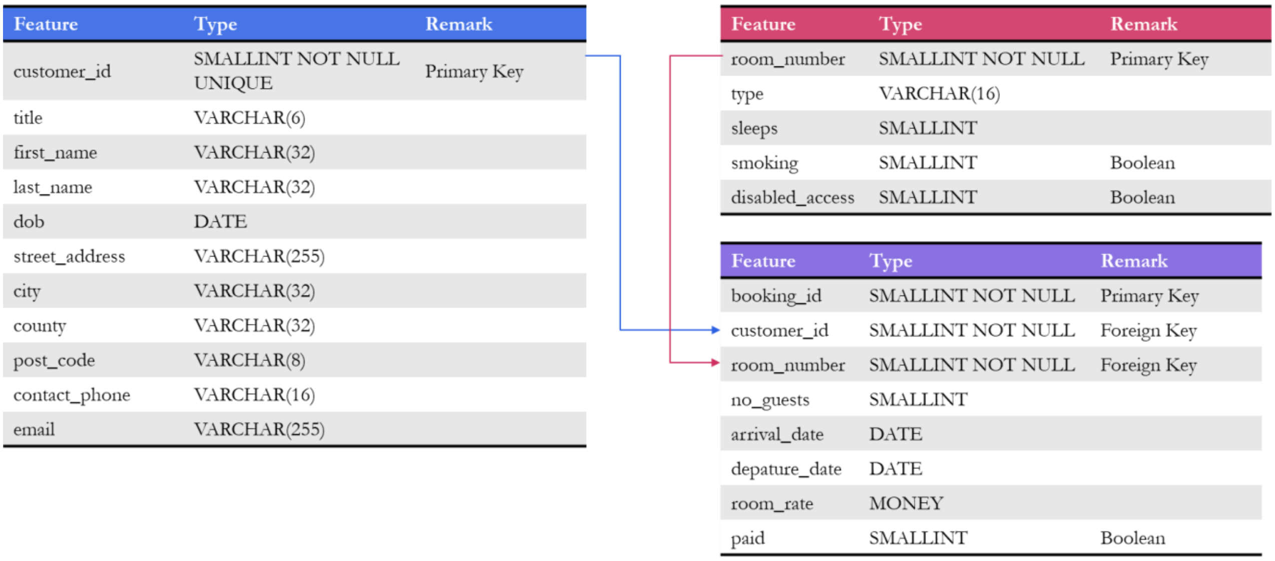
This tutorial was created using PostgreSQL 13.2, installed on Windows 10 and is entirely run from the command shell.

1.2 Tables

The database will comprise three tables [relations]: customer data, room data and booking data with the booking table taking foreign keys from customer and room data.

Each table will have a mixture of text, numerical, date and monetary formats.

1.3 Schema



Feature	Type	Remark
customer_id	SMALLINT NOT NULL UNIQUE	Primary Key
title	VARCHAR(6)	
first_name	VARCHAR(32)	
last_name	VARCHAR(32)	
dob	DATE	
street_address	VARCHAR(255)	
city	VARCHAR(32)	
county	VARCHAR(32)	
post_code	VARCHAR(8)	
contact_phone	VARCHAR(16)	
email	VARCHAR(255)	

Feature	Type	Remark
room_number	SMALLINT NOT NULL	Primary Key
type	VARCHAR(16)	
sleeps	SMALLINT	
smoking	SMALLINT	Boolean
disabled_access	SMALLINT	Boolean

Feature	Type	Remark
booking_id	SMALLINT NOT NULL	Primary Key
customer_id	SMALLINT NOT NULL	Foreign Key
room_number	SMALLINT NOT NULL	Foreign Key
no_guests	SMALLINT	
arrival_date	DATE	
depature_date	DATE	
room_rate	MONEY	
paid	SMALLINT	Boolean

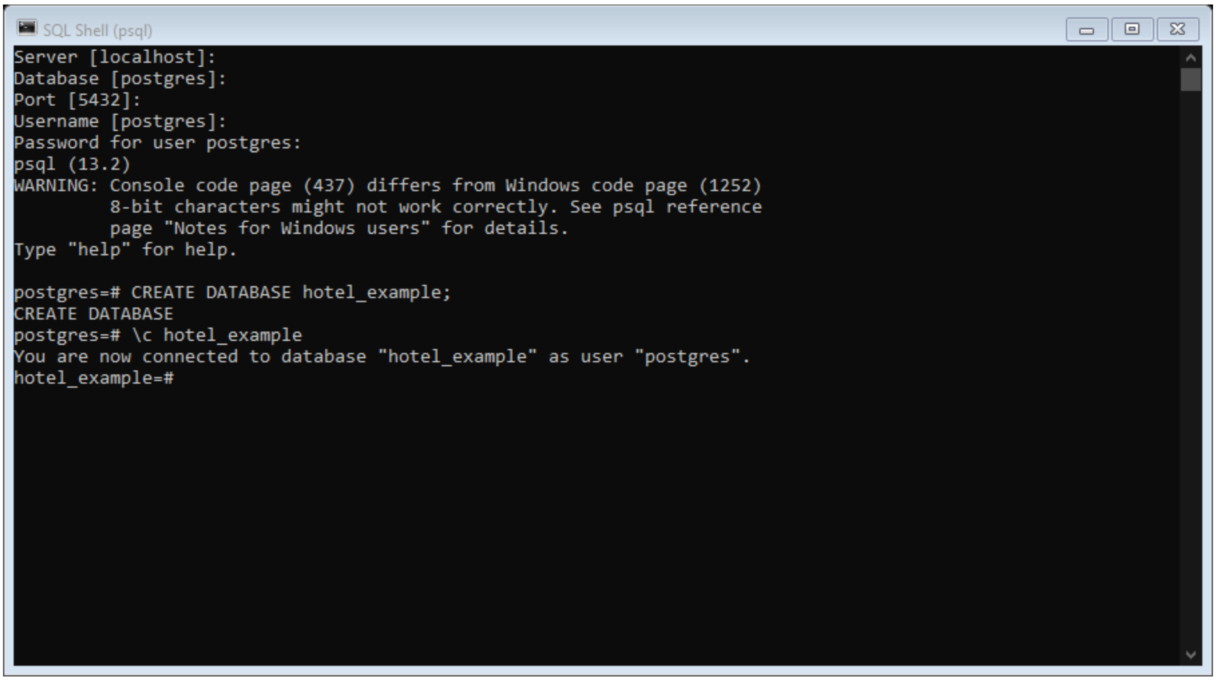
1.4 Database Creation

From the SQL Shell [psql], log in to the local host and any available database with a user that has database creation rights. In this instance the user “postgres” will be used.

To create a new database provide the command:

`postegres=# CREATE DATABASE hotel_example`

With the database created provide the command **`postegres=# \c hotel_example`** to connect to the new database:



```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (13.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# CREATE DATABASE hotel_example;
CREATE DATABASE
postgres=# \c hotel_example
You are now connected to database "hotel_example" as user "postgres".
hotel_example=#
```

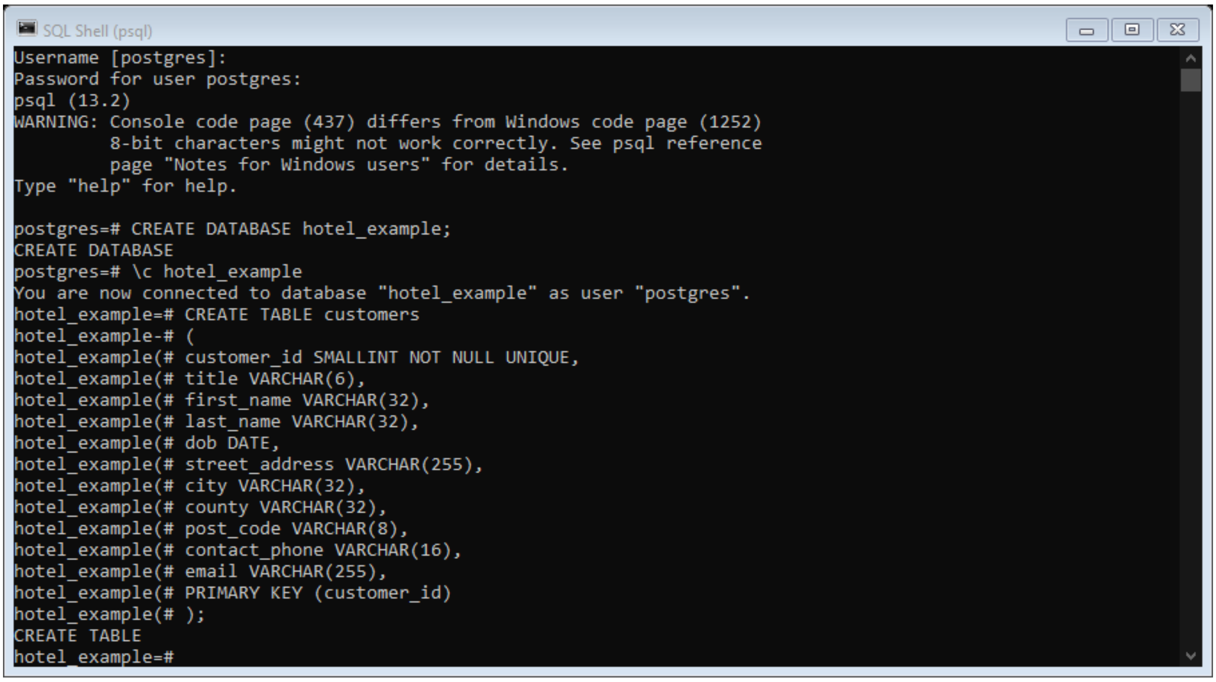
The command line should now change from **`postegres=#`** to **`hotel_example=#`**

1.5 Table/Relation Creation

To create a relation or table, in this instance the customer relation, provide the command:

```
hotel_example=# CREATE TABLE customers(customer_id SMALLINT NOT NULL UNIQUE,title
VARCHAR(6),first_name VARCHAR(32),last_name VARCHAR(32), dob DATE,street_address
VARCHAR(255),city VARCHAR(32),county VARCHAR (32),post_code VARCHAR (8),contact_phone
VARCHAR(16),email VARCHAR(255),PRIMARY KEY (customer_id));
```

If successful, the command line prompt will return: **`CREATE TABLE`**



```
Username [postgres]:
Password for user postgres:
psql (13.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# CREATE DATABASE hotel_example;
CREATE DATABASE
postgres=# \c hotel_example
You are now connected to database "hotel_example" as user "postgres".
hotel_example=# CREATE TABLE customers
hotel_example-# (
hotel_example(# customer_id SMALLINT NOT NULL UNIQUE,
hotel_example(# title VARCHAR(6),
hotel_example(# first_name VARCHAR(32),
hotel_example(# last_name VARCHAR(32),
hotel_example(# dob DATE,
hotel_example(# street_address VARCHAR(255),
hotel_example(# city VARCHAR(32),
hotel_example(# county VARCHAR(32),
hotel_example(# post_code VARCHAR(8),
hotel_example(# contact_phone VARCHAR(16),
hotel_example(# email VARCHAR(255),
hotel_example(# PRIMARY KEY (customer_id)
hotel_example(# );
CREATE TABLE
hotel_example=#
```

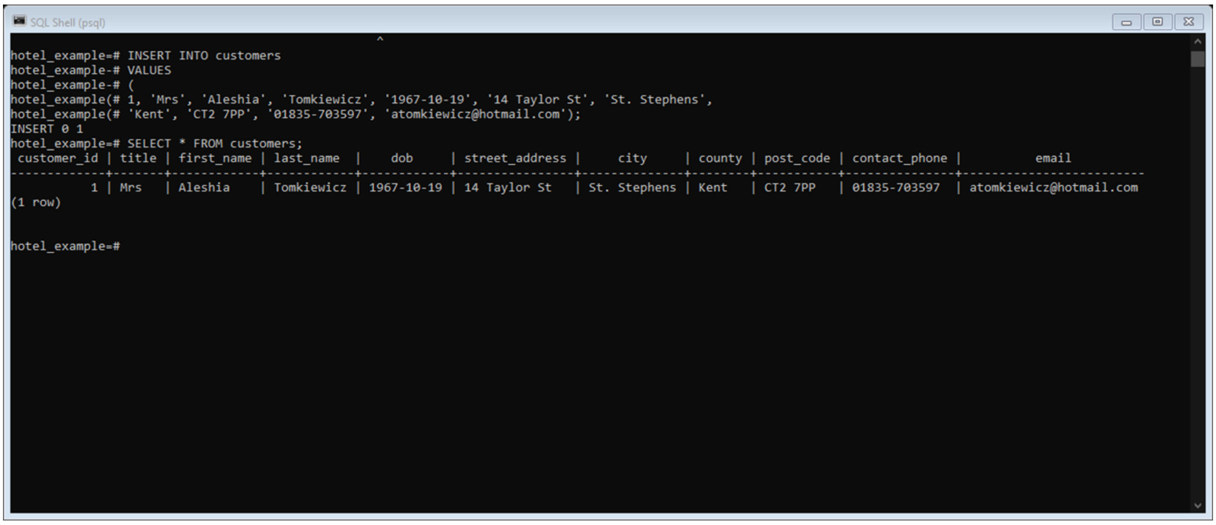
This step is required to be repeated for all three relations as outlined in *1.3 Schema*.

1.6 Data Entry

With the relations created, it is time to start populating, to enter the first customer data command:

```
hotel_example=# INSERT INTO customers VALUES (1,‘Mrs’,‘Aleshia’,‘Tomkiewicz’,‘1967-10-19’,‘14 Taylor St’,‘St. Stephens’,‘Kent’,‘CT2 7PP’,‘01835-703597’,‘atomkiewicz@hotmail.com’);
```

This should be repeated for all customer entries, this can either be completed via the shell or by creating a SQL script to import all rows, all sample data can be found in the appendix.



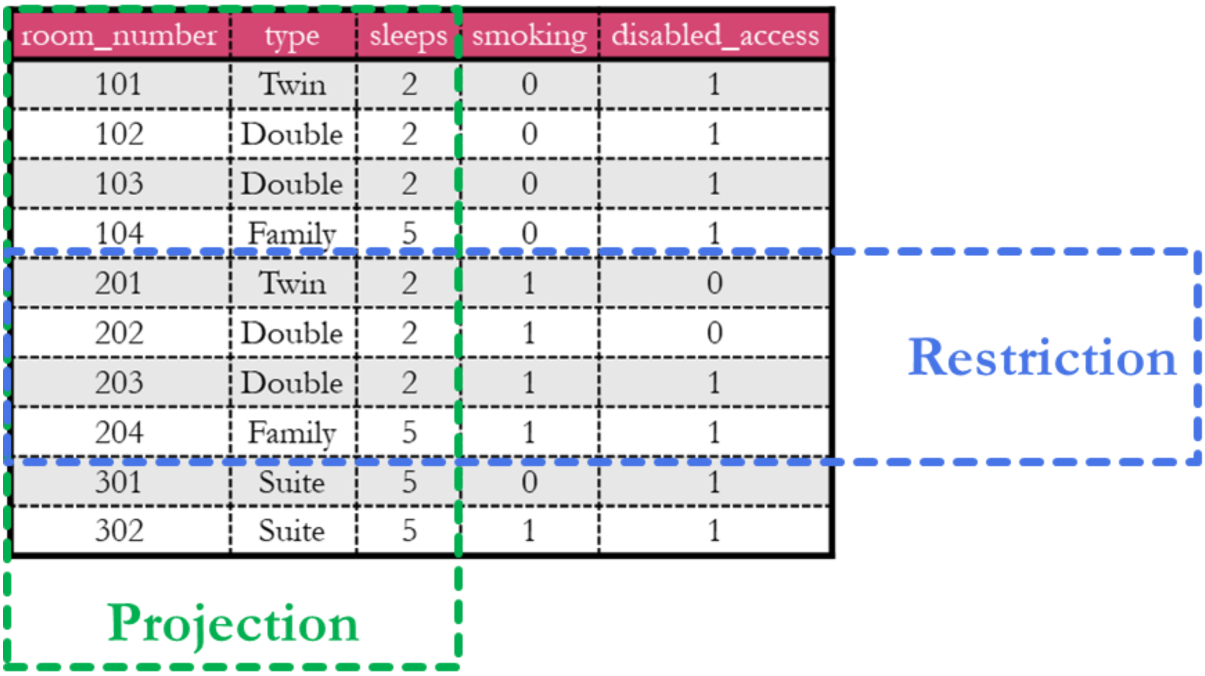
```
hotel_example=# INSERT INTO customers
hotel_example-# VALUES
hotel_example-# (
hotel_example(# 1, 'Mrs', 'Aleshia', 'Tomkiewicz', '1967-10-19', '14 Taylor St', 'St. Stephens',
hotel_example(# 'Kent', 'CT2 7PP', '01835-703597', 'atomkiewicz@hotmail.com');
INSERT 0 1
hotel_example=# SELECT * FROM customers;
 customer_id | title | first_name | last_name  |    dob     | street_address |     city     | county | post_code | contact_phone |          email
-------------+-------+------------+------------+------------+----------------+--------------+--------+-----------+---------------+-------------------------
           1 | Mrs   | Aleshia    | Tomkiewicz | 1967-10-19 | 14 Taylor St   | St. Stephens | Kent   | CT2 7PP   | 01835-703597  | atomkiewicz@hotmail.com
(1 row)


hotel_example=#
```

```
hotel_example=# INSERT INTO rooms VALUES (101,‘Twin’,2,0,1);
```

```
hotel_example=# INSERT INTO bookings VALUES (1,7,102,2,‘2021-06-15’,‘2021-06-18’,75,1)
```

1.7 Basic Querying

There are two basic operations that can be done with a relation: projections and restrictions. They can also be combined.



room_number	type	sleeps	smoking	disabled_access
101	Twin	2	0	1
102	Double	2	0	1
103	Double	2	0	1
104	Family	5	0	1
201	Twin	2	1	0
202	Double	2	1	0
203	Double	2	1	1
204	Family	5	1	1
301	Suite	5	0	1
302	Suite	5	1	1

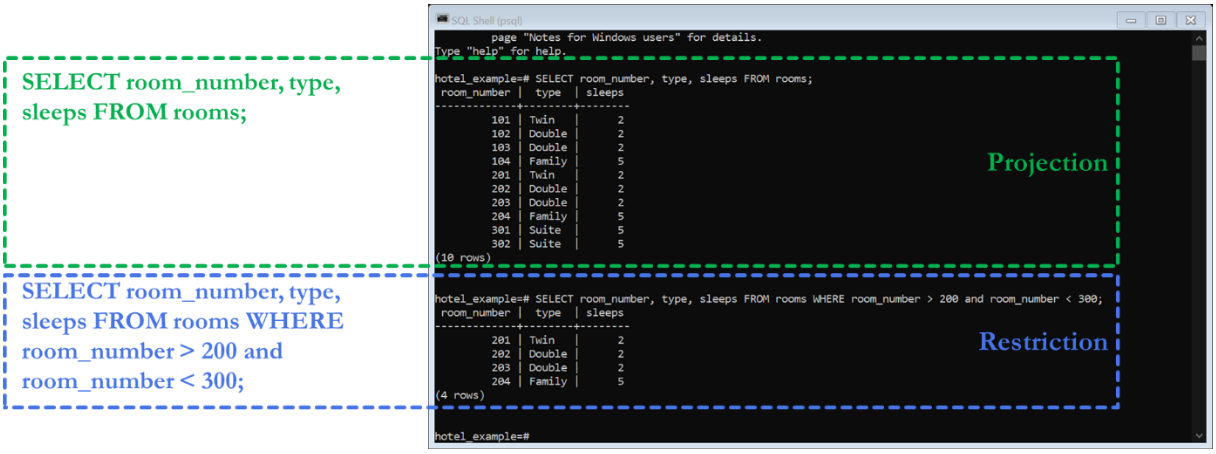
The standard query syntax structure is as follows:

SELECT ***columns*** **FROM** ***table/relation*** **WHERE** ***some criteria*;**

Examples:

```
SELECT room_number, type, sleeps FROM rooms;
```

```
SELECT room_number, type, sleeps FROM rooms WHERE room_number>200 and room_number<300;
```



SELECT room_number, type, sleeps FROM rooms;

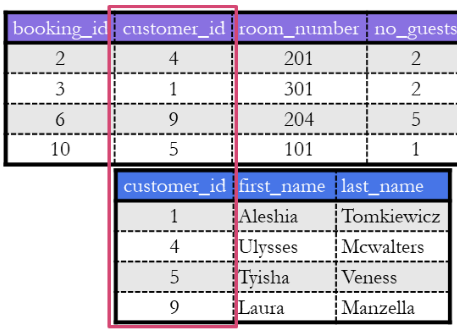
SELECT room_number, type, sleeps FROM rooms WHERE room_number > 200 and room_number < 300;

```
         page "Notes for Windows users" for details.
Type "help" for help.

hotel_example=# SELECT room_number, type, sleeps FROM rooms;
 room_number |  type  | sleeps
-------------+--------+--------
         101 | Twin   |      2
         102 | Double |      2
         103 | Double |      2
         104 | Family |      5
         201 | Twin   |      2
         202 | Double |      2
         203 | Double |      2
         204 | Family |      5
         301 | Suite  |      5
         302 | Suite  |      5
(10 rows)

hotel_example=# SELECT room_number, type, sleeps FROM rooms WHERE room_number > 200 and room_number < 300;
 room_number |  type  | sleeps
-------------+--------+--------
         201 | Twin   |      2
         202 | Double |      2
         203 | Double |      2
         204 | Family |      5
(4 rows)

hotel_example=#
```

1.8 Joins

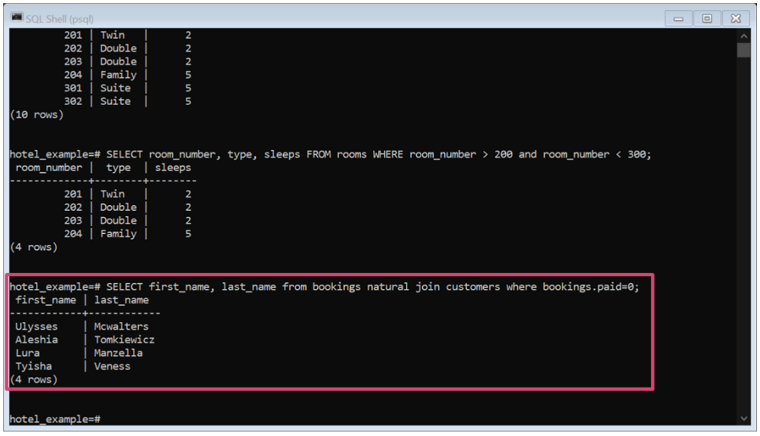
Natural joins can be used to create new relations; with foreign keys used to associate rows.

`SELECT first_name,last_name from bookings natural join customers where bookings.paid=0;`



booking_id	customer_id	room_number	no_guests	arrival_date	departure_date	room_rate	paid
2	4	201	2	2021-06-05	2021-06-06	£75	0
3	1	301	2	2021-06-04	2021-06-06	£135	0
6	9	204	5	202-06-06	2021-06-08	£110	0
10	5	101	1	2021-06-05	2021-06-07	£65	0

customer_id	first_name	last_name
1	Aleshia	Tomkiewicz
4	Ulysses	Mcwalters
5	Tyisha	Veness
9	Laura	Manzella



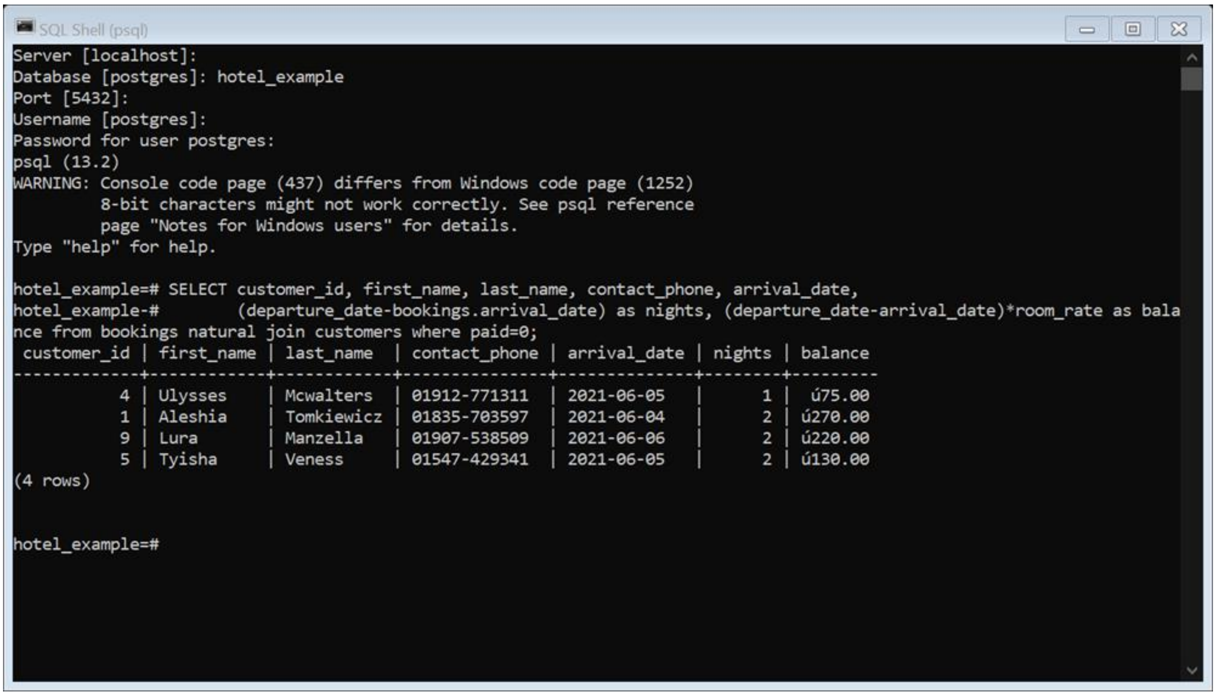
```
         201 | Twin   |      2
         202 | Double |      2
         203 | Double |      2
         204 | Family |      5
         301 | Suite  |      5
         302 | Suite  |      5
(10 rows)


hotel_example=# SELECT room_number, type, sleeps FROM rooms WHERE room_number > 200 and room_number < 300;
 room_number |  type  | sleeps
-------------+--------+--------
         201 | Twin   |      2
         202 | Double |      2
         203 | Double |      2
         204 | Family |      5
(4 rows)


hotel_example=# SELECT first_name, last_name from bookings natural join customers where bookings.paid=0;
 first_name | last_name
------------+-----------
 Ulysses    | Mcwalters
 Aleshia    | Tomkiewicz
 Lura       | Manzella
 Tyisha     | Veness
(4 rows)


hotel_example=#
```

1.9 Aggregation

Additionally, restrictions, projections, joins and aggregation can be used to create new relations.

In this example two new relations are created: *Number of nights staying* and *Balance owed*.

`SELECT customer_id,first_name,last_name,contact_phone,arrival_date,(departure_date-bookings.arrival_date) as nights,(departure_date-arrival_date)*room_rate as balance from bookings natural join customers where paid=0;`



```
Server [localhost]:
Database [postgres]: hotel_example
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (13.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

hotel_example=# SELECT customer_id, first_name, last_name, contact_phone, arrival_date,
hotel_example-#        (departure_date-bookings.arrival_date) as nights, (departure_date-arrival_date)*room_rate as bala
nce from bookings natural join customers where paid=0;
 customer_id | first_name | last_name  | contact_phone | arrival_date | nights | balance
-------------+------------+------------+---------------+--------------+--------+---------
           4 | Ulysses    | Mcwalters  | 01912-771311  | 2021-06-05   |      1 |  ú75.00
           1 | Aleshia    | Tomkiewicz | 01835-703597  | 2021-06-04   |      2 | ú270.00
           9 | Lura       | Manzella   | 01907-538509  | 2021-06-06   |      2 | ú220.00
           5 | Tyisha     | Veness     | 01547-429341  | 2021-06-05   |      2 | ú130.00
(4 rows)


hotel_example=#
```

2. MONGODB TUTORIAL

2.1 Introduction

This case study focusses on using a document database, MongoDB, to store movie reviews.

MongoDB was selected for this case study due to the variety of data that would be required to be stored in such as database that would make using a relational model very difficult or inefficient.

Part 1 of the tutorial will work through launching and interacting with MongoDB how to create a database, collections and inserting, updating, and deleting documents.

Part 2 of the tutorial will move on to querying a larger dataset.

This tutorial was created using MongoDB version 4.4.2, installed on Windows 10 and with the PATH environment variable pointed towards “.\MongoDB\Server\4.4\bin”.

2.2 Launching Mongo & Creating a Database

To launch MongoDB from a windows Command Prompt type:

```
>mongo
```

To then list the available databases type:

```
>show dbs
```

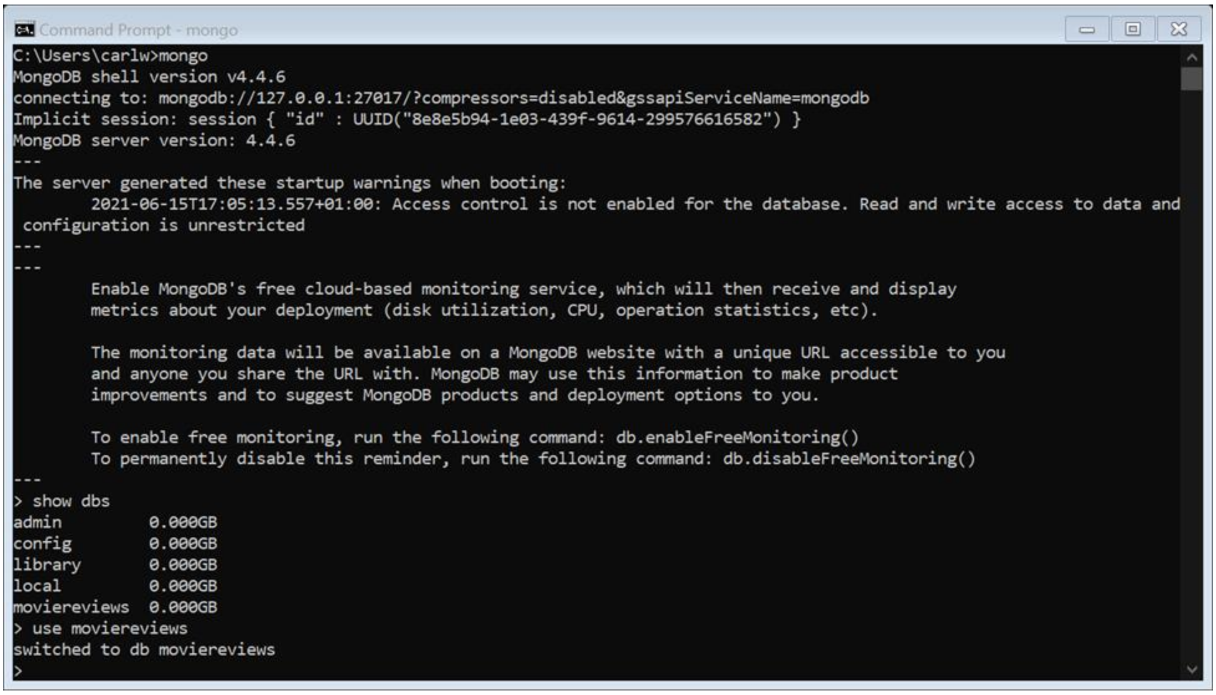
If we wanted to use one of these databases, we would simply type, for example:

```
>use library
```

Similarly, to create a new database, due to the schema-less design of MongoDB you can simply command:

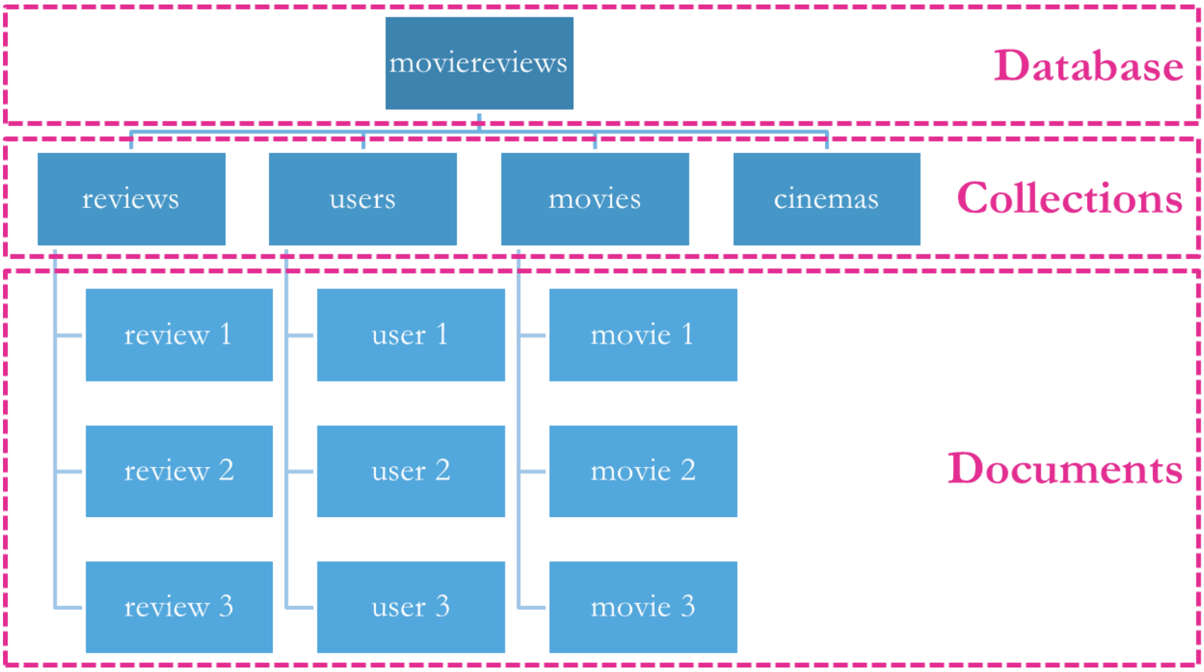
```
>use moviereviews
```

If the database does not exit, it will be created.



```
C:\Users\carlw>mongo
MongoDB shell version v4.4.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("8e8e5b94-1e03-439f-9614-299576616582") }
MongoDB server version: 4.4.6
---
The server generated these startup warnings when booting:
        2021-06-15T17:05:13.557+01:00: Access control is not enabled for the database. Read and write access to data and
 configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> show dbs
admin         0.000GB
config        0.000GB
library       0.000GB
local         0.000GB
moviereviews  0.000GB
> use moviereviews
switched to db moviereviews
>
```

2.3 Mongo Hierarchy & Collections



The underlying structure to a Mongo database essentially has three key levels: Database, Collections, Documents.

Each *documen*t is an object, and each object can contain objects nested within. A document is analogous to a tuple or row in a relational database. Each document will typically contain several attributes knows as fields, analogous to columns in a relational database.

2.4 Inserting Mongo Documents

Collections can be created by simply adding documents, if the collection doesn't exist it will be created, if it does then the document will be appended.

The syntax for inserting a document is relatively straight forward:

```
>db.collection.insert({field:value,field:value,field:[value,value,value,…]})
```

Let us insert our first move "Arrival":

```
>db.movies.insert({"title":"Arrival","category":["Drama","Sci-fi"],"cast":["Amy Adams",
"Jeremy Renner","Forest Whitaker","Michael Stuhlbarg","Tzi Ma"],"length":116,"year":2016,
"rating":"PG-13"})
```

Confirmation is provided by the shell by returning:

```
WriteResult({"nInserted":1})
```

2.5 Updating documents

We can check the entry by commanding:

```
>db.movies.findOne()
```

The “_id” field is a unique identifier for each document within the database.

The id is required to update a single specific document, for instance changing the rating from PG-13 to R-18:

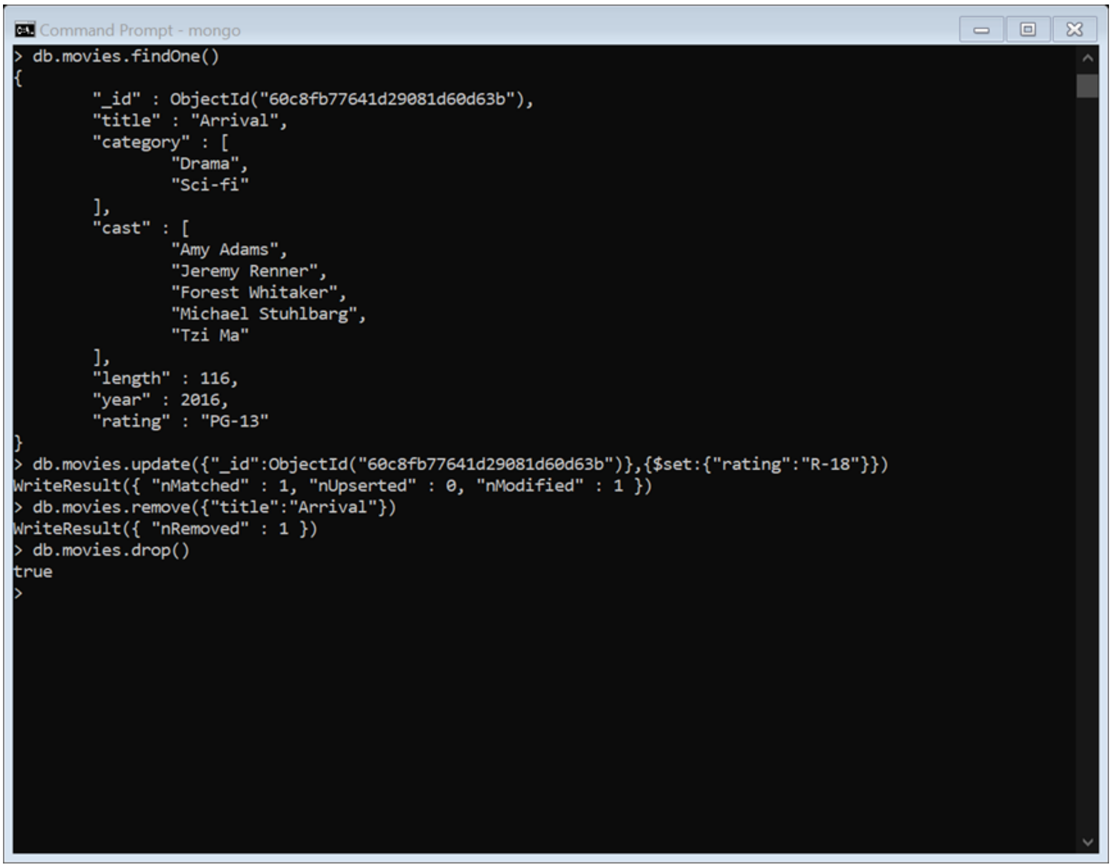
```
>db.movies.update({“_id”:ObjectId(“60c8fb77641d29081d60d63b”)},{$set:{“rating”:“R-18”}})
```

The entry can be removed by commanding:

```
>db.movies.remove({“title”:“Arrival”})
```

Finally, the collection can be removed by commanding:

```
>db.movies.drop()
```



```
> db.movies.findOne()
{
        "_id" : ObjectId("60c8fb77641d29081d60d63b"),
        "title" : "Arrival",
        "category" : [
                "Drama",
                "Sci-fi"
        ],
        "cast" : [
                "Amy Adams",
                "Jeremy Renner",
                "Forest Whitaker",
                "Michael Stuhlbarg",
                "Tzi Ma"
        ],
        "length" : 116,
        "year" : 2016,
        "rating" : "PG-13"
}
> db.movies.update({"_id":ObjectId("60c8fb77641d29081d60d63b")},{$set:{"rating":"R-18"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.movies.remove({"title":"Arrival"})
WriteResult({ "nRemoved" : 1 })
> db.movies.drop()
true
>
```

2.6 Importing Data

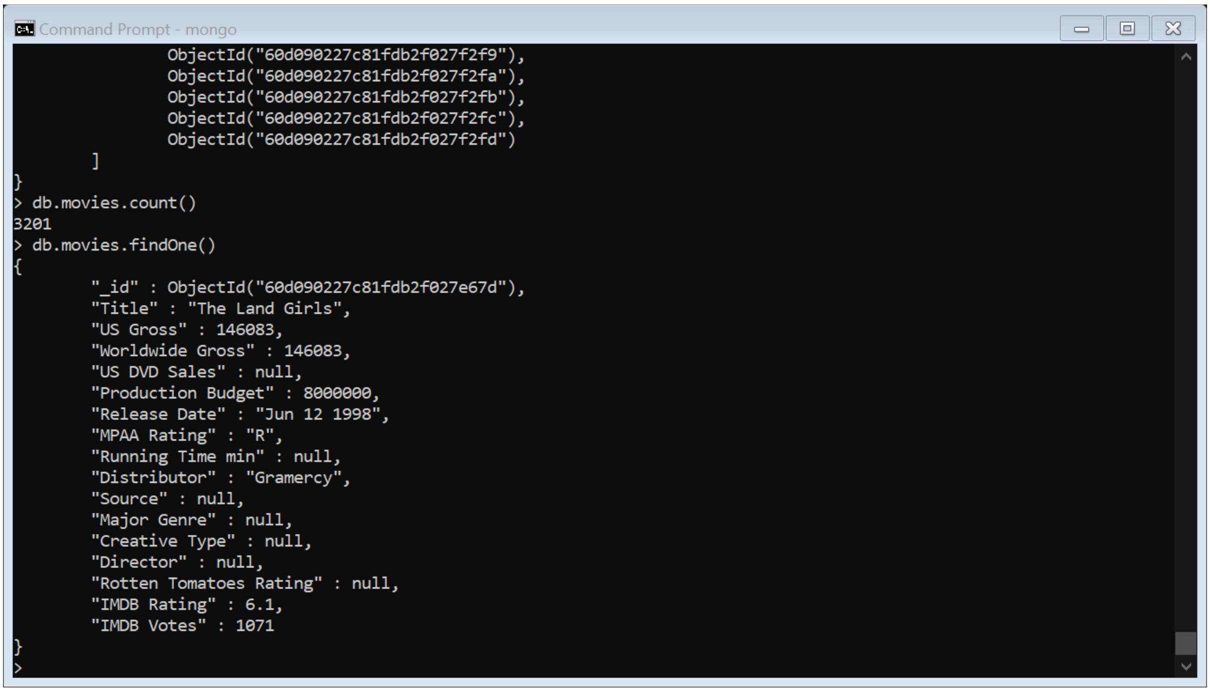
For the next portion of the tutorial, an external dataset (Vega, 2020) will be used.

1. Navigate to: https://github.com/vega/vega-datasets/blob/master/data/movies.json
2. Click “view raw”.
3. Copy all the text on the page including the square brackets.
4. Then in Mongo command:

`>db.movies.insertMany(`*`paste here`*`)` i.e:

`>db.movies.insertMany([{"Title": "The Land Girls”… "IMDB Votes": 4789}])`

This will take a few moments to paste into the command prompt, ensure the final close parenthesis is included.



```
                ObjectId("60d090227c81fdb2f027f2f9"),
                ObjectId("60d090227c81fdb2f027f2fa"),
                ObjectId("60d090227c81fdb2f027f2fb"),
                ObjectId("60d090227c81fdb2f027f2fc"),
                ObjectId("60d090227c81fdb2f027f2fd")
        ]
}
> db.movies.count()
3201
> db.movies.findOne()
{
        "_id" : ObjectId("60d090227c81fdb2f027e67d"),
        "Title" : "The Land Girls",
        "US Gross" : 146083,
        "Worldwide Gross" : 146083,
        "US DVD Sales" : null,
        "Production Budget" : 8000000,
        "Release Date" : "Jun 12 1998",
        "MPAA Rating" : "R",
        "Running Time min" : null,
        "Distributor" : "Gramercy",
        "Source" : null,
        "Major Genre" : null,
        "Creative Type" : null,
        "Director" : null,
        "Rotten Tomatoes Rating" : null,
        "IMDB Rating" : 6.1,
        "IMDB Votes" : 1071
}
>
```

5. Command: **`>db.movies.count()`** the result should return 3201 as above.

2.7 Querying

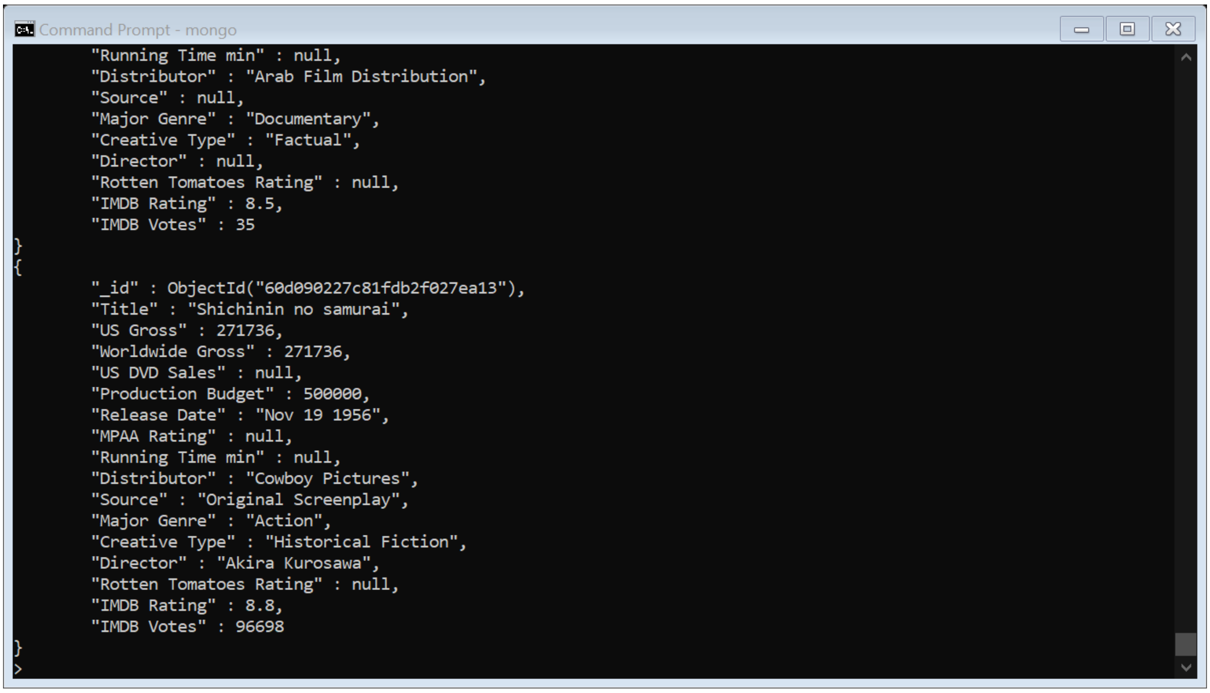
Queries are generally constructed in JavaScript notation as regular expressions in the following format:

```
>db.collection.find({
    logical_operator:[
        {field1{comparison_operator:value}},
        {field2:{comparison_operator:value}}
        ]}).method
```

For example, finding how many movies has a US Gross less than or equal to than [$LTE] $1M but have an IMDB Rating greater than or equal [$GTE] to 8.5:

```
>db.movies.find({$and:[{"US Gross":{$lte:1000000}},{"IMDB Rating":{$gte:8.5}}]}).count()
```

We can view the results by changing `.count()` for `.pretty()`:



```
        "Running Time min" : null,
        "Distributor" : "Arab Film Distribution",
        "Source" : null,
        "Major Genre" : "Documentary",
        "Creative Type" : "Factual",
        "Director" : null,
        "Rotten Tomatoes Rating" : null,
        "IMDB Rating" : 8.5,
        "IMDB Votes" : 35
}
{
        "_id" : ObjectId("60d090227c81fdb2f027ea13"),
        "Title" : "Shichinin no samurai",
        "US Gross" : 271736,
        "Worldwide Gross" : 271736,
        "US DVD Sales" : null,
        "Production Budget" : 500000,
        "Release Date" : "Nov 19 1956",
        "MPAA Rating" : null,
        "Running Time min" : null,
        "Distributor" : "Cowboy Pictures",
        "Source" : "Original Screenplay",
        "Major Genre" : "Action",
        "Creative Type" : "Historical Fiction",
        "Director" : "Akira Kurosawa",
        "Rotten Tomatoes Rating" : null,
        "IMDB Rating" : 8.8,
        "IMDB Votes" : 96698
}
>
```

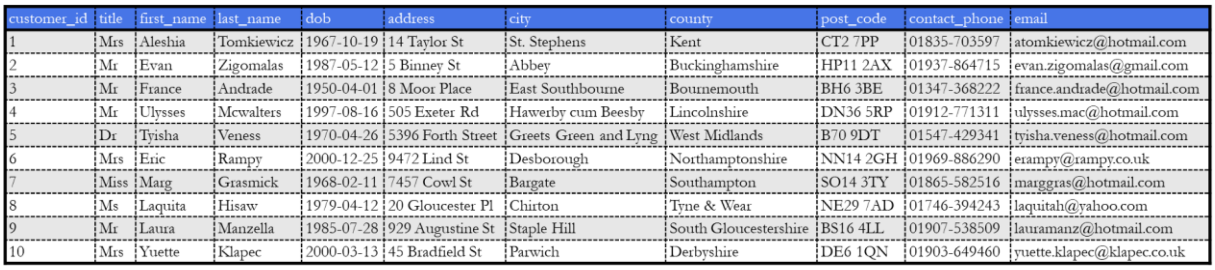
All four of the results returned in this case have a “Running Time min” as null. Is this missing for all entries? We can check:

```
>db.movies.find({“Running Time min”:{$ne:null}}).count()
```

The result is 1209; so 62% of the records have this attribute as null.

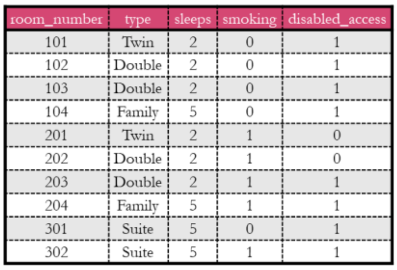
3. APPENDIX

3.1 Postgres Example Data



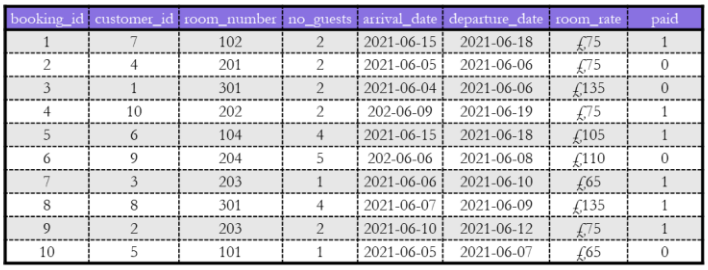
customer_id	title	first_name	last_name	dob	address	city	county	post_code	contact_phone	email
1	Mrs	Aleshia	Tomkiewicz	1967-10-19	14 Taylor St	St. Stephens	Kent	CT2 7PP	01835-703597	atomkiewicz@hotmail.com
2	Mr	Evan	Zigomalas	1987-05-12	5 Binney St	Abbey	Buckinghamshire	HP11 2AX	01937-864715	evan.zigomalas@gmail.com
3	Mr	France	Andrade	1950-04-01	8 Moor Place	East Southbourne	Bournemouth	BH6 3BE	01347-368222	france.andrade@hotmail.com
4	Mr	Ulysses	Mcwalters	1997-08-16	505 Exeter Rd	Hawerby cum Beesby	Lincolnshire	DN36 5RP	01912-771311	ulysses.mac@hotmail.com
5	Dr	Tyisha	Veness	1970-04-26	5396 Forth Street	Greets Green and Lyng	West Midlands	B70 9DT	01547-429341	tyisha.veness@hotmail.com
6	Mrs	Eric	Rampy	2000-12-25	9472 Lind St	Desborough	Northamptonshire	NN14 2GH	01969-886290	erampy@rampy.co.uk
7	Miss	Marg	Grasmick	1968-02-11	7457 Cowl St	Bargate	Southampton	SO14 3TY	01865-582516	marggras@hotmail.com
8	Ms	Laquita	Hisaw	1979-04-12	20 Gloucester Pl	Chirton	Tyne & Wear	NE29 7AD	01746-394243	laquitah@yahoo.com
9	Mr	Laura	Manzella	1985-07-28	929 Augustine St	Staple Hill	South Gloucestershire	BS16 4LL	01907-538509	lauramanz@hotmail.com
10	Mrs	Yuette	Klapec	2000-03-13	45 Bradfield St	Parwich	Derbyshire	DE6 1QN	01903-649460	yuette.klapec@klapec.co.uk

Table 1: Customer Data - customer data from BrianDunning.com (2021).



room_number	type	sleeps	smoking	disabled_access
101	Twin	2	0	1
102	Double	2	0	1
103	Double	2	0	1
104	Family	5	0	1
201	Twin	2	1	0
202	Double	2	1	0
203	Double	2	1	1
204	Family	5	1	1
301	Suite	5	0	1
302	Suite	5	1	1

Table 2: Room Data



booking_id	customer_id	room_number	no_guests	arrival_date	departure_date	room_rate	paid
1	7	102	2	2021-06-15	2021-06-18	£75	1
2	4	201	2	2021-06-05	2021-06-06	£75	0
3	1	301	2	2021-06-04	2021-06-06	£135	0
4	10	202	2	202-06-09	2021-06-19	£75	1
5	6	104	4	2021-06-15	2021-06-18	£105	1
6	9	204	5	202-06-06	2021-06-08	£110	0
7	3	203	1	2021-06-06	2021-06-10	£65	1
8	8	301	4	2021-06-07	2021-06-09	£135	1
9	2	203	2	2021-06-10	2021-06-12	£75	1
10	5	101	1	2021-06-05	2021-06-07	£65	0

Table 3: Booking Data

4. REFERENCES

Dunning, B. (2021). *Free Sample Data*. https://www.briandunning.com/sample-data/uk-500.zip

Vega. (2020). *movies.json*. https://github.com/vega/vega-datasets/blob/master/data/movies.json