MYSQL EXERCISES

ABSTRACT

This document outlines the MySQL exercises to go through during the Databases course

QA Consulting Academy Team

Databases

Contents

MySQL Exercises

Installation and Setup

1. Access the MySQL installer file from the local installers and downloads folder, or alternatively download it at http://dev.mysql.com/downloads/installer/

2. Start the executable and install the full option that includes the server and tools like MySQL Workbench.

    a. You may need to have previously installed Microsoft .NET 4.0 Framework and Microsoft Visual C++ 2013 Redistributable Package (MSVC2013)

    b. Leave the configuration options the same in Type & Networking, if you do change anything, make a note of it.

    c. Set up your MySQL password as 'password' so it is easier to troubleshoot for now

    d. It is possible to add users at this point or later, for now we are just setting up the root user.

    e. Leave the configuration options in 'Windows Service' as they are. This will install MySQL as a windows service.

    f. Leave everything else as it is then press execute to finish the configuration.

3. To start MySQL Workbench simply find it in Start -> Programs and click on it

Connections

1. To add a connection click on the + icon next to MySQL Connections on the home screen.

2. In the 'Setup New Connection' box that appears, enter a 'Connection Name' such as MyConnection

3. The default connection values are for a local setup, so you shouldn't need to change these. Click the 'Test Connection' button to check

4. Click on 'Configure Server Management'. Press 'Next' and your connection will be tested. This should be successful, and if not, you need to go back and change your configuration.

5. It should skip the Management and OS and SSH Configuration options as they are only for remote MySQL Connections

6. Select the appropriate MySQL service for the connection, it should automatically be selected for you

7. It will then test if it can start and stop the service

8. Follow the rest of the steps without changing anything then click Finish. You will return to the Setup New Connection window. Click 'Ok' to start the connection.

9. Your new connection should now be listed on the home window

10. You can click this connection to open the SQL editor for this connection

11. Click 'Server Status' in the left panel in order to display the current server status. Feel free to explore some of the other options within the Navigator panel.

Modelling

1. On the home screen, click the + next to models

2. Click the + button next to the Physical Schemata toolbar. Modify the name field to something like test_db. Confirm this change in the Physical Schemas panel

3. Double click 'Add table' in the Physical Schemas section

4. The table editor will load, change the Table Name field to test_table

5. Add columns to your table by double clicking a Column Name cell. MySQL Workbench will automatically default the first field to idtest_table, keep the data type as INT and select the check boxes for PK (Primary Key), NN (Not Null), and AI (Auto Increment).

6. Add some additional columns:

    a. Col Name: fname Data type: VARCHAR(45) Properties: NN
    b. Col Name: birthday Data type: DATE Properties: none

7. You can see a visual representation of what you've made by selecting Model -> Create Diagram from Catalog Objects

8. Save the model

9. Select Database -> Forward Engineer…. Be sure to add your root password. Then use the default options to export your object to the live server. You should just need to select to export the MySQL Table Objects. It will then show you the SQL script it has created in order to export this. Review this to understand what is happening.

Let's create another table using the visual tool.

1. On the MySQL Model page double click on Add Diagram to open a new canvas

2. Use the tools on the left to design your diagram.

3. Click on the rectangular grid in the middle of the toolbar, use this to create a table by then clicking on the canvas. Hover over any of the options for detail.

4. Right click the table and click on edit in new tab

5. Change the table name to invoice in the Name field

6. In columns create a column called invoice_id

7. Have the data type as INT

8. Pressing tab will make a new column. Add a description and customer_id column.

9. Place a new table on the canvas and call it invoice_item

10. Click the 1:n Non-Identifying Relationship tool. Then click the invoice_item table followed by the invoice table to create a foreign key in the invoice_item table on the 'many' side of the relationship.

11. If you ever need to, click on the arrow in the toolbar to return to a regular mouse pointer

12. Click on the invoice_item table and select the foreign keys tab

13. Click on the foreign key name field and it should show the referenced table and appropriate column

14. You can delete the relationship by right clicking the line joining the tables and choosing 'delete'

15. Try adding new objects using the toolbar, remember to save when you are finished

Querying

1. Open up the SQL Editor (double click on your connection on the home tab)

2. To execute a query simply write your SQL in the Query tab then press the lightning button, you can also save your queries for later

3. From the navigator panel under SCHEMAS select the test_table from the test_db we created earlier. Right click on the table and click on 'Select Rows - Limit 1000'

4. Obviously the results panel will return NULL since we haven't added anything to the table. However, you can add data to the results grid.

5. Note: the ID column will automatically increment so you do not need to add values to this column. Input at least 5 names with birth dates to your table. Note that DATE types are

formatted as YYYY-MM-DD

6. Click ‘Apply’ to send these changes to the live server

7. View the results grid again and you should see the ID field filled with auto increments

8. You could also execute a query like ‘USE test_db; SELECT * FROM test_table;’ in order to display this data

Let’s upload the movielens dataset and perform some queries on it. For these exercises, try running them in the command line. You can log in by typing mysql -u <user> -p entering root for the username and password when prompted.

1. Open up the movielens.sql file. Make sure you understand what is happening at each step for creating each table and then inserting data into it.

2. Create a new database called movielens by running CREATE DATABASE movielens; then select to use that database by running use movielens;

3. Run the movielens.sql file by running ‘source C:/path/to/file/movielens.sql;’ where you should replace the path with wherever you have stored the file

4. Type show tables; to explore the different tables that have been imported.

5. Perform select commands like SELECT * FROM genres_movies LIMIT 10; in order to explore the data available.

6. Let’s start trying to answer some questions. Start with - what is the most common movie genre?

    a. SELECT genre_id, count(genre_id) as cnt from genres_movies group by genre_id; - this will return all the genre ids and how often they appear to describe movies.

    b. SELECT genre_id, count(genre_id) as cnt from genres_movies group by genre_id order by cnt desc limit 1; - this will return the genre_id with the highest amount of occurrences.

    c. SELECT g.name, count(m.genre_id) as cnt from genres_movies m join genres g on (m.genre_id = g.id) group by genre_id order by cnt desc limit 1; - this will return the most common movie genre associated with the movies in this dataset and how often it appears.

7. With which occupation is the Horror genre most frequently rated?

    a. First, find out which id the Horror genre matches up to **11**

b. Think through the problem to determine which tables you will need to access to start considering your joins.

c. Start with the following that should have all of the tables you need joined, then consider how you can start to filter it to get the information you need.
select * from ratings r join genres_movies g on r.movie_id=g.movie_id join users u on r.user_id=u.id where g.id=11 limit 10;

d. We can either find the occupation id from this and then cross reference it to the occupations table, or we can also join the occupation table to our query.
select o.name, count(u.occupation_id) as cnt from ratings r join genres_movies g on r.movie_id=g.movie_id join users u on r.user_id=u.id join occupations o on u.occupation_id=o.id where g.id=11 group by u.occupation_id order by cnt desc limit 1;

8. Earlier we saw that students gave the most ratings for horror movies, however, does this just mean that students give the most ratings or is the horror genre the most popular with students?

a. First, let's check how many ratings have been given by each occupation. This will give us a good indication if there are simply many students adding ratings.
select o.name, count(r.rating) as cnt from ratings r join users u on r.user_id = u.id join occupations o on u.occupation_id=o.id group by u.occupation_id order by cnt desc;

b. You should see that students and other appear at the top yet again! This means we should probably be looking in to some different statistics in order to determine the most popular genres amongst occupations. We could go down a few different routes here - the exercises will walk you through one then you should attempt the other.

c. Let's try and find some different ideas around who enjoys the horror movies the most. Instead of looking at number of ratings, let's look at the average rating given by each occupation for horror movies.
select o.name, avg(r.rating) as cnt from ratings r join genres_movies g on r.movie_id = g.movie_id join users u on r.user_id=u.id join occupations o on u.occupation_id=o.id where g.id=11 group by u.occupation_id order by cnt desc;

d. Of course this will also return occupation names that only gave 1 or 2 reviews! See if you can remove any occupations that have given less than 3 reviews. **SELECT o.name, AVG(r.rating) as average, COUNT(r.rating) as cnt FROM occupations o JOIN users u ON u.occupation_id = o.id JOIN ratings r ON r.user_id = u.id JOIN genres_movies g ON g.movie_id = r.movie_id WHERE g.id = 11 GROUP BY o.name HAVING cnt > 2 ORDER BY average DESC;**

e. Attempt to find out which is the most highly rated genre amongst each occupation. **SELECT o.name as occup, g.name as gen FROM occupations o JOIN users u ON o.id = u.occupation_id JOIN ratings r ON u.id = r.user_id JOIN genres_movies gm ON**

r.movie_id = gm.movie_id JOIN genres g ON gm.id = g.id GROUP BY o.name;

9. Which movie has the most reviews? **SELECT m.title, COUNT(r.rating) as cnt FROM ratings r JOIN movies m ON r.movie_id = m.id GROUP BY m.title ORDER BY cnt desc limit 1;**
10. Which movie with more than 10 reviews is most highly rated? **SELECT m.title, AVG(r.rating) as average, COUNT(r.rating) as cnt FROM ratings r JOIN movies m ON r.movie_id = m.id GROUP BY m.title HAVING cnt>10 ORDER BY average desc limit 1;**
11. Which movie with more than 10 reviews is rated as the worst? **SELECT m.title, AVG(r.rating) as average, COUNT(r.rating) as cnt FROM ratings r JOIN movies m ON r.movie_id = m.id GROUP BY m.title HAVING cnt>10 ORDER BY average asc limit 1;**
12. Continue exploring this dataset to further refine your querying skills.

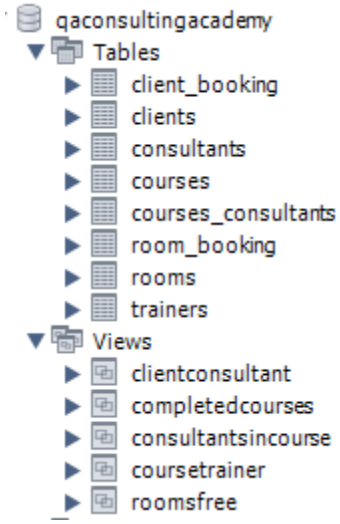
Tasks

For each task consider carefully as to what information would be needed for such an operation and the queries that need answering. The scenarios outline a brief description for the situation, but it is up to you to expand on these using your user stories and create a complete documentation as to what it can be used for. Document your process including diagrams and queries.

For each scenario:

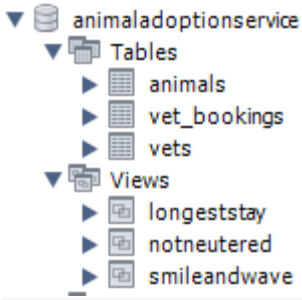
- Write appropriate user stories
- Create an ERD from those user stories
- Create the database within MySQL
- Populate with data
- At a minimum, perform the queries necessary to answer the questions within the scenario and to fulfil your user stories
- Create users with permissions to create an access hierarchy

1. QA Consulting Academy.
    a. As a Trainer, I want to see what rooms are not in use next week so I can book it for a course
    b. As a Trainer, I want to see what consultants are currently taking a particular course so I know who to expect when training
    c. As an Academy Director, I want to see which Trainer owns which course so I know who to liaise with regarding a particular opportunity
    d. As a Consultant Development Liaison I want to see what courses a consultant has gone through so I know what skills they have
    e. As an Account Manager I want to see what consultants a client is assigned to so that I can let clients know



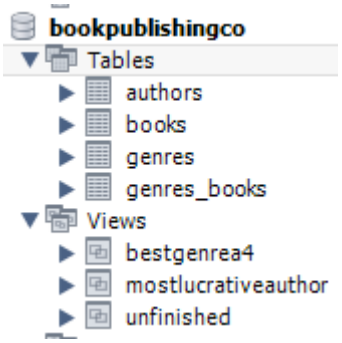
```
/*Q1 ONLY FOR 2019-07-08 NOT DYNAMIC*/
CREATE VIEW roomsFree AS SELECT name as 'Free rooms' FROM rooms WHERE
 idrooms != (SELECT rooms_id FROM room_booking WHERE startDate =
"2019-07-08");
/*Q2 ONLY FOR CHRISTOPHER NOT DYNAMIC*/
CREATE VIEW consultantsInCourse AS SELECT cour.name as Course, cons.
name as Consultant FROM courses cour JOIN courses_consultants cc ON
cour.idcourses = cc.courses_id JOIN consultants cons ON cc.
consultants_id = cons.idconsultants JOIN trainers t ON cour.
trainers_id = t.idtrainers WHERE t.name = "Christopher" AND cc.status
 = 1;
/*Q3*/
CREATE VIEW courseTrainer AS SELECT c.name as Course, t.name as
Trainer FROM trainers t JOIN courses c ON c.trainers_id = t.
idtrainers ORDER BY c.idcourses asc;
/*Q4 ONLY FOR IMRAN NOT DYNAMIC*/
CREATE VIEW completedCourses AS SELECT cour.name as Course, cons.name
 as Consultant FROM courses cour JOIN courses_consultants cc ON cour.
idcourses = cc.courses_id JOIN consultants cons ON cc.consultants_id
= cons.idconsultants WHERE cons.name = "Imran" AND cc.status = 0;
= cons.idconsultants WHERE cons.name = "Imran" AND cc.status = 0;
/*Q5*/
CREATE VIEW clientConsultant AS SELECT cli.name as 'Client', cons.
name as 'Consultant' FROM clients cli JOIN client_booking clibo ON
cli.idclients = clibo.clients_id JOIN consultants cons ON clibo.
consultants_id = cons.idconsultants ORDER BY cli.idclients;
```

2. Animal adoption service.
    a. As a Health Care Officer, I want to query what animals have not been neutered so I can book an appointment with the vets
    b. As a Social Media Consultant, I want to query which animals have been there the longest so I can create a campaign online to get them homed quickly
    c. As a Receptionist, I want to query animals by specific criteria to help direct customers to the animals they wish to adopt.




```
/*Q1*/
CREATE VIEW notNeutered AS SELECT name AS 'Name' FROM animals WHERE
idanimals NOT IN (SELECT idanimals FROM vet_bookings) AND neutered =
0 AND exit_date IS NULL;
/*Q2*/
CREATE VIEW longestStay AS SELECT name As 'Name', entry_date AS
'Entered' FROM animals WHERE exit_date IS NULL ORDER BY entry_date;
/*Q3 ONLY FOR PENGUINS NOT DYNAMIC*/
CREATE VIEW smileAndWave AS  SELECT name AS 'Name', anim_type AS
'Species' FROM animals WHERE exit_date IS NULL AND anim_type =
'Penguin';
```

3. Book publishing company.
    a. As a Publishing Director, I want to search for our most lucrative author so I can appropriately reward them
    b. As a Marketing Executive, I wish to search for most popular genres by a specific author to create tailored advertising schemes
    c. As an Author Liaison, I want to search for books that are not yet complete by the order of the date they were started, so I can manage authors effectively




```
/*Q1*/
CREATE VIEW mostLucrativeAuthor AS SELECT a.fname AS 'Author', b.
stock_sold AS 'Books Sold' FROM authors a JOIN books b ON a.idauthors
 = b.idauthors ORDER BY stock_sold desc LIMIT 1;
/*Q2 ONLY FOR AUTHOR 4 NOT DYNAMIC*/
CREATE VIEW bestGenreA4 AS SELECT  a.fname AS 'Author', g.name AS
'Genre', SUM(b.stock_sold) as sold FROM genres g JOIN genres_books gb
 ON g.idgenres = gb.idgenres JOIN books b ON gb.idbooks = b.idbooks
JOIN authors a ON a.idauthors = b.idauthors WHERE b.idauthors = 4
GROUP BY g.idgenres ORDER BY sold desc LIMIT 1;
/*Q3*/
CREATE VIEW unfinished AS SELECT a.fname, b.title, b.start_date FROM
authors a JOIN books b ON a.idauthors = b.idauthors WHERE b.published
 = 0 ORDER BY b.start_date;
```