Temasek Polytechnic

School of Informatics & IT

**Database Application Development (CIA1C06)**

**AY 22/23 Oct Semester**

**Project Part 3b**

**(SQL and Server-Side Codes)**

|  |  |
| --- | --- |
| Name: | JAVEN LAI LE YU |
| Admission Number: | 2202934B |
| Class: | P17 |

Table of Contents

SQL Codes 3-15

SQL Keywords used 16-17

# 

# SQL Codes

Accounts:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Functionalities of Web Application** | **Related SQL code**  **(Full and complete SQL Statements, not partial)** | **URL Route Pattern** | **HTTP Method** | **Result From Postman**  **(Pass/Fail)**  (To be filled by Lecturer) |
| View User’s Account  and all the Reviews they made. | **SELECT** username, picture, email, phone, role**, DATE\_FORMAT**(acc.date, "%d/%m/%Y") **AS** date,  **GROUP\_CONCAT(**review**) AS** reviews,  **GROUP\_CONCAT(**res.name**) AS** restaurants,  **GROUP\_CONCAT(**rev.rating**) AS** ratings  **FROM** accounts **AS** acc **LEFT JOIN** reviews **AS** rev  **ON** acc.idAccount = rev.idAccount  **LEFT JOIN** restaurants **AS res**  **ON** res.idRestaurant = rev.idRestaurant  **WHERE** acc.idAccount **= TRIM(**?**)**  **ORDER BY** res.idRestaurant | /api/account/me | GET |  |
| Create New Account | **INSERT INTO** accounts (idAccount, username, password, phone, email, date) **VALUES** (NULL, TRIM(?), TRIM(?), TRIM(?), TRIM(?), TRIM(?)) | /api/account/register | POST |  |
| Login to Account.  Added idAccount to enhance security. | **SELECT** \* **FROM** accounts **WHERE** username = TRIM(?) **AND** idAccount = TRIM(?) | /api/account/login | POST |  |
| Update Account by Account’s ID  Using idAccount to specify the account. | **UPDATE** foodiedb.accounts **SET** username = TRIM(?), password = TRIM(?), phone = TRIM(?), email = TRIM(?) **WHERE** idAccount = TRIM(?) | /api/account/me | PUT |  |
| Delete Account by Account’s ID | **DELETE** **FROM** accounts **WHERE** idAccount = TRIM(?) | /api/account/me | DELETE |  |
| Upload Profile Picture to User’s Account | **UPDATE** accounts **SET** picture = TRIM(?) **WHERE** idAccount = TRIM(?) | /api/account/me/upload | PUT |  |
| Get count of how many existing Accounts there are. | **SELECT** **COUNT**(\*) **AS** count **FROM** accounts | /api/account/count | GET |  |
| Reset Password 1: Check if Account name and details are accurate. | **SELECT \* FROM** accounts **WHERE** username **= TRIM(?) AND** phone **= TRIM(?) AND** email **= TRIM(?)** | /api/account/resetPassword | POST |  |
| 2: Reset Password. Use BINARY to keep strict check. | **UPDATE** accounts **SET** password **= TRIM(?) WHERE** username **= TRIM(?) AND BINARY** password **= TRIM(?)** | /api/account/resetPassword2 | POST |  |

Restaurants:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Functionalities of Web Application** | **Related SQL code**  **(Full and complete SQL Statements, not partial)** | **URL Route Pattern** | **HTTP Method** | **Result From Postman**  **(Pass/Fail)**  (To be filled by Lecturer) |
| View All Restaurants available in database. | **SELECT** restaurants.idRestaurant, restaurants.name, restaurants.description, **GROUP\_CONCAT(DISTINCT** image**) AS** images**, ROUND(AVG(**rating**),** 1**) AS** ratio**, COUNT(**rating**) AS** reviewers **FROM** restaurants **RIGHT JOIN** images **ON** restaurants.idRestaurant =images.idRestaurant **LEFT JOIN** reviews **ON** restaurants.idRestaurant =reviews.idRestaurant **GROUP BY** idRestaurant **ORDER BY** idRestaurant | /api/restaurant/ | GET |  |
| Get a selected Restaurant using its id | **SELECT restaurants.idRestaurant, name, description, idAccount, menu,**  **DATE\_FORMAT(date, "%d %M %Y")**  **AS date,**  **GROUP\_CONCAT(image)**  **AS images**  **FROM restaurants**  **CROSS JOIN images**  **ON restaurants.idRestaurant = images.idRestaurant**  **WHERE restaurants.idRestaurant = TRIM(?)** | /api/restaurant/:idRestaurant | GET |  |
| Get all Categories for a Restaurant. | **SELECT** **DISTINCT**(category) **FROM** categories **WHERE** idRestaurant = TRIM(?) | /api/restaurant/:idRestaurant | GET |  |
| Get all branches and their respective branch\_details for a specific Restaurant. | **SELECT** idRestaurant, branch, **GROUP\_CONCAT**(title) **AS** titles, **GROUP\_CONCAT**(content) **AS** contents **FROM** branches **LEFT JOIN** branch\_details **ON** branches.idBranch = branch\_details.idBranch **WHERE** idRestaurant = TRIM(?) **GROUP BY** branch **ORDER BY** title | /api/restaurant/:idRestaurant | GET |  |
| Search recommendation algorithm:  Top 4 Searches linked to a Restaurant + the latest / most recent Search made by the User. | **(SELECT** searchQuery**,**  **FLOOR( COUNT(**  **(**restaurants.name **LIKE** searchQuery**)**  **OR (SOUNDEX(**searchQuery**) = SOUNDEX**(restaurants.name))  **OR (**searchQuery **IN** (restaurants.name))  ) / 9) **AS** count  **FROM** (searches, restaurants)  **GROUP BY** searchQuery  **ORDER BY** count **DESC**  **LIMIT** 4)    **UNION**    **(SELECT** searchQuery**, NULL**  **FROM** searches  **WHERE** idAccount **= TRIM(?)**  **ORDER BY** datetime **DESC LIMIT 1)**    **LIMIT** 5 | /api/restaurant/log | GET |  |
| Search for Restaurant using name + Filter result using category + Sort.  Robust search that uses Wildcard and Soundex.  Soundex allows similar sounding names to be accepted – more results shown.  Adds error tolerance in case of User typo; User may even discover a new restaurant with similar name.  $(?) Allows input to alter and run as the SQL query – to specify how results should be ordered. | **SELECT** restaurants.idRestaurant, name, description,  **GROUP\_CONCAT(DISTINCT** image**) AS** images**,**  **ROUND(AVG(**rating**),** 1**) AS** ratio**, COUNT(**rating**) AS** reviewers**,**  **GROUP\_CONCAT(DISTINCT** category**) AS** categories  **FROM** restaurants  **LEFT JOIN** categories  **ON** restaurants.idRestaurant = categories.idRestaurant  **LEFT JOIN** reviews  **ON** restaurants.idRestaurant = reviews.idRestaurant  **RIGHT JOIN** images  **ON** restaurants.idRestaurant = images.idRestaurant  **WHERE** (restaurants.name  **LIKE CONCAT(**'%', **TRIM**(?), '%'**)**  **OR SOUNDEX(**?**)**  **LIKE SOUNDEX(**restaurants.name**))**  **AND (**categories.category **LIKE CONCAT(**'%',**TRIM**(?),'%'))  **GROUP BY** idRestaurant  **ORDER BY ${**sort**}** | /api/restaurant/search | POST |  |
| Log a Search if that search is made by a logged in User.  The IF clause sets an empty search as NULL. The database is rejects a NULL value for that column – and the search isn’t logged. | **INSERT INTO** searches (idAccount, searchQuery, datetime)  **VALUES(**  **IF(TRIM(?)** != '', **TRIM(?), NULL),**  **IF(TRIM(?)** != '', **TRIM(?), NULL),**  ?) | /api/restaurant/log | POST |  |
| Checks if a Search is the same or a shortened version of the search previously made by the same User less than a minute ago. | **SELECT COUNT(\*) AS** count  **FROM** searches  **WHERE** idAccount **= TRIM(?)**  **AND** searchQuery **LIKE CONCAT(TRIM(?), "%")**  **AND DATE\_FORMAT(**datetime**, "%Y-%m-%d %h:%i") = ?** | /api/restaurant/log | POST |  |
| Create a New Restaurant | **INSERT INTO** restaurants (name, description, menu, idAccount, date**) VALUES** (TRIM(?), TRIM(?), TRIM(?), TRIM(?), ?) | /api/restaurant/log | POST |  |
| Edit/Update an existing Restaurant which the Account owns | **UPDATE** restaurants **SET** name = TRIM(?), description = TRIM(?), menu = ? **WHERE** idRestaurant = TRIM(?) **AND** idAccount = TRIM(?) | /api/restaurant/edit | PUT |  |
| Delete an existing Restaurant which the Account owns | **DELETE FROM** restaurants **WHERE idRestaurant = TRIM(?) AND** idAccount = TRIM(?) | /api/restaurant/edit | DELETE |  |
| Select all the Restaurant an Account owns | **SELECT** restaurants.idRestaurant, restaurants.name, restaurants.description, **GROUP\_CONCAT(**image**) FROM** restaurants **RIGHT JOIN** images **ON** restaurants.idRestaurant = images.idRestaurant **WHERE** idAccount = TRIM(?) **GROUP BY** idRestaurant **ORDER BY** idRestaurant | /api/restaurant/edit | GET |  |
| Update role whenever an Account creates or deletes a Restaurant listing | **UPDATE** accounts **SET** role = TRIM(?) **WHERE** idAccount = TRIM(?) | /api/restaurant/edit | POST, PUT, DELETE |  |

List (Favourites and Bookmarks):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Functionalities of Web Application** | **Related SQL code**  **(Full and complete SQL Statements, not partial)** | **URL Route Pattern** | **HTTP Method** | **Result From Postman**  **(Pass/Fail)**  (To be filled by Lecturer) |
| View User’s Favourites or Bookmarks (Specified using type) | **SELECT** list.\*, **DATE\_FORMAT**(list.datetime, '%d/%m/%Y %H:%i:%s') **AS** date**, GROUP\_CONCAT(**image**) AS** images, name, description **FROM** list **LEFT JOIN** restaurants **ON** list.idRestaurant = restaurants.idRestaurant **RIGHT JOIN** images **ON** images.idRestaurant = list.idRestaurant **WHERE** list.idAccount = TRIM(?) **AND** type = ‘B’ **GROUP BY** list.idRestaurant | /api/list/bookmark | GET |  |
| Add a Restaurant to User’s Favourites or Bookmarks | **INSERT** **INTO** list (idAccount, idRestaurant, type, datetime) **VALUES** (TRIM(?), TRIM(?), ‘B’, **NOW**()) | /api/list/bookmark | POST |  |
| Delete a Restaurant from User’s Favourites or Bookmarks | **DELETE** **FROM** list **WHERE** idAccount = TRIM(?) **AND** idRestaurant = TRIM(?) **AND** type = ‘B’ | /api/list/bookmark | DELETE |  |

Reviews:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Functionalities of Web Application** | **Related SQL code**  **(Full and complete SQL Statements, not partial)** | **URL Route Pattern** | **HTTP Method** | **Result From Postman**  **(Pass/Fail)**  (To be filled by Lecturer) |
| Select the Review and the Username of who made the review, for a particular Restaurant | **SELECT** R.idRestaurant, R.idAccount, R.rating, R.review, **DATE\_FORMAT(**R.date, "%d/%m/%Y"**) AS** date, R.edited,  A.username  **FROM** reviews **AS** R  **JOIN** accounts **AS** A  **ON** A.idAccount = R.idAccount  **WHERE** idRestaurant **= TRIM(?)**  **ORDER BY** R.date **DESC** | /api/review/:idRestaurant | GET |  |
| Add a new Review for a Restaurant | **INSERT INTO** reviews (idAccount, idRestaurant, rating, review, date) **VALUES** (TRIM(?), TRIM(?), TRIM(?), TRIM(?), TRIM(?)) | /api/review/edit/:idRestaurant | POST |  |
| Edit the review made by its Account | **UPDATE** reviews  **SET** rating = TRIM(?), review = TRIM(?), date = TRIM(?), edited = "T"  **WHERE** idAccount = TRIM(?)  **AND** idRestaurant = TRIM(?) | /api/review/edit/:idRestaurant | PUT |  |
| Delete Review made by the Account | **DELETE** **FROM** reviews **WHERE** idRestaurant = TRIM(?) **AND** idAccount = TRIM(?) | /api/review/edit/:idRestaurant | DELETE |  |

Upvotes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Functionalities of Web Application** | **Related SQL code**  **(Full and complete SQL Statements, not partial)** | **URL Route Pattern** | **HTTP Method** | **Result From Postman**  **(Pass/Fail)**  (To be filled by Lecturer) |
| **Get Vote details** (likes, dislikes and ratio) **for every Review** in the specified Restaurant **only if that Review has votes.**  No rows are returned for Reviews without votes – for optimization. | **SELECT** t.idRestaurant, t.idAccount, **SUM**(t.likes) **AS** likes, **SUM**(t.dislikes) **AS** dislikes, **SUM**(t.likes - t.dislikes) **AS** ratio  **FROM** (  **SELECT** upvotes.idRestaurant, upvotes.idAccount, **COUNT**(vote) **AS** likes, 0 **AS** dislikes  **FROM** upvotes **WHERE** vote = "L"  **GROUP BY** upvotes.idRestaurant, upvotes.idAccount    **UNION ALL**    **SELECT** upvotes.idRestaurant, upvotes.idAccount, 0 **AS** likes, COUNT(vote) **AS** dislikes  **FROM** upvotes **WHERE** vote = "D"  **GROUP BY** upvotes.idRestaurant, upvotes.idAccount  ) t  **WHERE** t.idRestaurant = ?  **GROUP BY** t.idAccount | /api/upvote/:idRestaurant | GET |  |
| **Get Extensive Vote details of** an **Account** (Account creation date, likes, dislikes and ratio) | **SELECT** a.username, **DATE\_FORMAT**(a.date, "%d/%m/%Y") AS date,  t.idAccount, **SUM**(t.likes) **AS** likes, **SUM**(t.dislikes) **AS** dislikes, **SUM**(t.likes - t.dislikes) **AS** ratio  **FROM** (  **SELECT** upvotes.idAccount, **COUNT**(vote) **AS** likes, 0 **AS** dislikes  **FROM** upvotes **WHERE** vote = "L"  **GROUP BY** upvotes.idAccount    **UNION ALL**    **SELECT** upvotes.idAccount, 0 **AS** likes, COUNT(vote) **AS** dislikes  **FROM** upvotes **WHERE** vote = "D"  **GROUP BY** upvotes.idAccount  ) t  **LEFT JOIN** accounts **AS** a  **ON** t.idAccount = a.idAccount  **WHERE** a.username = ? | /api/upvote/me/:username | GET |  |
| Add a Vote to a Review.  Review’s identification is a Composite Key of idRestaurant and idAccount. | INSERT INTO upvotes (idRestaurant, idAccount, accountId, vote) VALUES (?, ?, ?, ?) | /api/upvote/edit/:idRestaurant | POST |  |
| Remove Vote on a Review. | DELETE FROM upvotes WHERE idRestaurant = ? AND idAccount = ? AND accountId = ? | /api/upvote/edit/:idRestaurant | DELETE |  |
| Change Vote on a Review. | UPDATE upvotes SET vote = ? WHERE idRestaurant = ? AND idAccount = ? AND accountId = ? | /api/upvote/edit/:idRestaurant | PUT |  |

# SQL Keywords used

1. INSERT (C)
2. SELECT (R)
3. UPDATE (U)
4. DELETE (D)
5. BINARY // Checks for Uppercase and Lowercase too
6. FROM
7. SET
8. VALUES
9. WHERE
10. AND
11. AS
12. INTO
13. IF
14. IN
15. ON
16. OR
17. LIKE
18. ORDER BY
19. GROUP BY
20. LIMIT
21. DESC
22. ASC
23. JOIN
24. LEFT JOIN
25. RIGHT JOIN
26. CROSS JOIN
27. SOUNDEX() // enhance and maximize search by showing results of similar words.
28. TRIM() // remove while/empty spaces at start and end of string
29. NOW() // get Date
30. DATE\_FORMAT() // convert date to a more human friendly and readable format: 2022-1-12 => 1 December 2022
31. SUM() // used for calculating values of all rows in same column into a single sum.
32. COUNT() // to find how many rows in a column
33. FLOOR()
34. AVG() // used to find average rating: (SUM() / COUNT())
35. GROUP\_CONCAT() // concatenate values in a column to a single row
36. DISTINCT() // to filter duplicates from being selected
37. ROUND() // round off the nearest d.p. as specified
38. RAND()
39. UNION // combine SELECT queries
40. ALL // complimentary to UNION: take everything in tables.
41. % // wildcard for flexibility
42. \* // access all
43. “,” // access multiple
44. “.” // access column within table
45. “$()” // to insert keywords into statement via JavaScript