
CSCI 5408

***Data Management, Warehousing, And
Analytics***

Assignment 2 - Problem 1

Performing a systematic literature review and providing summary.

Prepared By

Bhavisha Oza (B00935827)

Problem-1: Perform a systematic literature review and provide summary.

“A comparative Analysis of Data Fragmentation in Distributed Database” [1].

1. Summary:

A comparative Analysis of Data Fragmentation in Distributed Database [1] paper proposes a comparative analysis of data fragmentation in distributed database systems. It begins by highlighting the significance of distributed systems in modern computing technology and explains that distributed databases consist of data partitions or fragments that are distributed or replicated across multiple physical locations. The paper emphasizes the importance of the distributed database design process, which includes initial design, redesign, and materialization phases.

The introduction of the paper emphasizes the significance of distributed systems in modern computing technology, particularly in handling large amounts of data. Distributed databases are logical databases that are physically distributed across multiple locations connected by a network. The distribution of data involves fragmentation, replication, and allocation processes. The authors highlight the advancements in communication technology, software, and hardware, making distributed database systems more feasible and efficient.

The initial design, redesign, and materialisation of the redesign phases of the distributed database design process are covered. Algorithms for allocation and fragmentation are used in the early design to reduce the expense of transaction processing. New fragmentation and allocation methods are created throughout the redesign process based on modifications to the distributed database environment. The new fragmentation and allocation mechanism is put into place during the materialisation phase.

Related works in the field of distributed database systems are presented, including studies on fragmentation schemes and allocation schemes. Various researchers have proposed different approaches for vertical and horizontal fragmentation in distributed databases, aiming to improve query performance and minimize communication costs. They emphasise the benefits and drawbacks of fragmentation and discuss several strategies put forth by earlier researchers.

The paper then introduces three types of fragmentations: horizontal fragmentation (HF), vertical fragmentation (VF), and mixed fragmentation (MF). A relation or class is divided into disjoint tuples or instances using horizontal fragmentation, with each fragment being stored at a separate node. Vertical fragmentation divides a relation into sets of columns or characteristics, each set containing one or more of the table's main key properties. Combining horizontal and vertical fragmentations, known as mixed fragmentation, enables the creation of more complex fragmentation schemes.

Data fragmentation is explained as the process of breaking a single object into two or more fragments or segments. The three types of fragmentations are illustrated using the example of a Human Resources table, showing how the table can be horizontally fragmented into multiple fragments, vertically fragmented into separate sets of columns, or mixed fragmented with both horizontal and vertical partitions.

Finally, the research discusses fragmentation accuracy rules such as completeness, reconstruction, and disjointness. Each data item in the original relation must be discovered in at least one fragment to be complete. The existence of a relational operator capable of reconstructing the original relation from its fragments are required for reconstruction. Disjointness assures that the fragments do not overlap or have redundancy.

In conclusion, the paper provides an overview of data fragmentation in distributed database systems. It compares horizontal, vertical, and mixed fragmentations and discusses the advantages and disadvantages of each. In the end, the authors suggest that the choice of data fragmentation for distributed database depends on the specific requirements and objectives of the system. Understanding the different types of fragmentation and applying the correctness rules can help in designing efficient distributed databases.

2. Scope of Improvements:

There are few areas in the paper where some improvements can be made. As the paper focuses on the fragmentation types and its advantages, disadvantages, it should shed some lights on the methodologies as well. The paper does not mention the methodology used for the comparative analysis. It is important to describe the criteria and metrics used to evaluate the different types of fragmentation and how the analysis was conducted. Providing a clear methodology will enhance the credibility of the study and allow readers to understand the basis of the comparisons made.

It would be beneficial to provide insights into future research directions or potential areas for improvement in data fragmentation techniques in distributed databases. The paper does not address potential challenges or limitations of data fragmentation in distributed databases just like “*When applied to more complex applications such as CAD/CAM, software design, office information systems, and expert systems, the relational data model exhibits limitations in terms of complex object support, type system, and rule management*” [2]. It would be valuable to discuss issues and the issues which still requires a more work such as Sensitivity, Complexity, Transaction, Replication, Interface, Design, Interconnection [2].

References:

- [1] A. Al-Sanhani, A. Hamdan, A. Al-Dahoud and A. Al-Dahoud "A comparative Analysis of Data Fragmentation in Distributed Database," *2017 8th International Conference on Information Technology (ICIT)*, Amman (11733), Jordan, 2017, pp. 724-729. DOI: 10.1109/ICITECH.2017.8079934.
- [2] M. T. Ozu and P. Valduriez, "Distributed database systems: where are we now?," *in Computer*, vol. 24, no. 8, pp. 72-78, Aug. 1991. DOI: 10.1109/2.84879.

CSCI 5408

***Data Management, Warehousing, And
Analytics***

Assignment 2 - Problem 2

Prototype of a light-weight DBMS using Java programming language.

Prepared By

Bhavisha Oza (B00935827)

Problem-2: Prototype of a light-weight DBMS using Java programming language (no 3rd party libraries allowed).

1. Summary:

This document outlines the requirements for developing a console-based Java application using a standard Java IDE [1]. The application is designed to accept user input in the form of SQL queries after successful login authentication. The Java code do follow JavaDocs specification for commenting styles, including annotations such as "@param" and "@return" [2]. The design principles used in the application's development is utilising the SOLID design principles [3].

Two main functionalities are there in the application. First, a two-factor authentication module which is necessary for user authentication, which involves using ID, password, and question/answer authentication [4]. The application supports multiple users, and it is designed in a separate class to handle the authentication process. Secondly, the application implements a custom-designed persistent storage system to store data, user information, logs, etc. A custom file format is used, and standard formats like JSON, XML, and CSV are not used. Custom delimiters are designed for storing and accessing data within a text file.

In addition to the required functionalities, the application also implements various SQL queries such as CREATE, SELECT, INSERT, UPDATE, and DELETE on any number of tables. Separate methods have been created for each query. Furthermore, a transaction handling logic is implemented, allowing users to initiate and end transactions. To ensure the ACID (Atomicity, Consistency, Isolation, Durability) properties, the processed queries do not immediately update the custom-made database text file. Instead, they are stored in intermediate data structures such as Lists. Only when a "Commit" operation is performed, the changes are applied to the text file.

Overall, the application's focus is on developing a console-based Java program that supports user authentication, custom storage, SQL query execution, and transaction handling following the ACID properties.

Requirements & Tasks fulfilled:

1. Use a standard **Java IDE** to develop your application. Any JDK version is acceptable [1]:

IntelliJ IDEA Community Edition is used for the entire code.

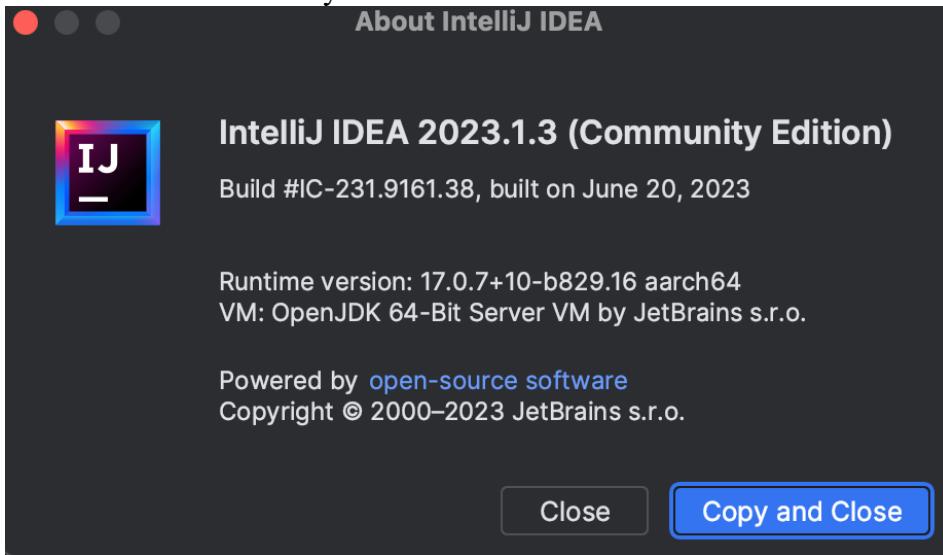


Figure 1: Java IDE setup.

2. While writing the Java code, followed the **JavaDocs specification** for commenting styles, such as @param, @return [2].

The screenshot shows the IntelliJ IDEA interface with the 'UserAuthenticationModule.java' file open. The code editor displays the following JavaDoc comments and code snippets:

```
/*
 * Module for user authentication.
 */
3 usages
public class UserAuthenticationModule {
    7 usages
    private final Map<String, User> usersMap;
    3 usages
    private final UserFileHandler fileHandler;
    11 usages
    private final Scanner scanner;
    2 usages
    private boolean isAuthenticated = false;
    // Constructor for a UserAuthenticationModule.
    1 usage
    public UserAuthenticationModule() {...}

    /**
     * Starts the user authentication module.
     *
     * @return true if authentication is successful, false otherwise
     */
    1 usage
    public boolean start() {...}

    // Displays the menu options.
    1 usage
    private void displayMenu() {...}
}
```

Figure 2: User Authentication module class- JavaDocs specification for commenting styles.

```

Project ▾
  ✓ CSC15408_A2 ~/ideaProjects/CSC15408
    > .idea
    > logs
    ✓ src
      ✓ main
        ✓ java
          ✓ authentication
            ✓ User
            ✓ UserAuthenticationModule
            ✓ UserFileHandler
      ✓ database
        ✓ CreateTable
        ✓ DeleteTable
        ✓ InsertTable
        ✓ SelectTable
        ✓ UpdateTable
      > utils
        ✓ RunApplication
      ✓ resources
    > test
    ✓ target
      ✓ .gitignore
      pom.xml
  > External Libraries
  Scratches and Consoles

UserFileHandler.java ×
1 package authentication;
2
3 import java.io.*;
4 import java.util.Map;
5
6 /**
7  * Utility class for handling user file operations.
8  */
9 public class UserFileHandler {
10    private final String filePath;
11
12    /**
13     * Constructs a UserFileHandler object.
14     *
15     * @param filePath the path of the user file
16     */
17    public UserFileHandler(String filePath) {
18        this.filePath = filePath;
19    }
20
21    /**
22     * Loads user data from the user file into the provided map.
23     *
24     * @param userMap the map to store the loaded user data
25     */
26    public void loadUsers(Map<String, User> userMap) { ... }

```

Figure 3: User File Handler class- JavaDocs specification for commenting styles.

```

Project ▾
  ✓ CSC15408_A2 ~/ideaProjects/CSC15408
    > .idea
    > logs
    ✓ src
      ✓ main
        ✓ java
          ✓ authentication
            ✓ User
            ✓ UserAuthenticationModule
            ✓ UserFileHandler
      ✓ database
        ✓ CreateTable
        ✓ DeleteTable
        ✓ InsertTable
        ✓ SelectTable
        ✓ UpdateTable
      > utils
        ✓ RunApplication
      ✓ resources
    > test
    ✓ target
      ✓ .gitignore
      pom.xml
  > External Libraries
  Scratches and Consoles

User.java ×
12 /**
13  * Constructor for User object with the specified username, password, security question, and answer.
14  *
15  * @param username the username of the user
16  * @param password the password of the user
17  * @param question the security question chosen by the user
18  * @param answer the answer to the security question
19  */
20 public User(String username, String password, String question, String answer) { ... }
21
22 /**
23  * Returns the username of the user.
24  *
25  * @return the username of the user
26  */
27 public String getUsername() { return username; }
28
29 /**
30  * Sets the username of the user.
31  *
32  * @param username the username of the user
33  */
34 public void setUsername(String username) { this.username = username; }
35
36 /**
37  * Returns the password of the user.
38  *
39  * @return the password of the user
40  */
41
42 /**
43  * Sets the password of the user.
44  *
45  * @param password the password of the user
46  */
47
48 /**
49  * Returns the security question of the user.
50  *
51  * @return the security question of the user
52  */
53
54 /**
55  * Sets the security question of the user.
56  *
57  * @param question the security question of the user
58  */
59
60 /**
61  * Returns the answer of the user.
62  *
63  * @return the answer of the user
64  */
65
66 /**
67  * Sets the answer of the user.
68  *
69  * @param answer the answer of the user
70  */
71
72 /**
73  * Returns the question of the user.
74  *
75  * @return the question of the user
76  */
77
78 /**
79  * Sets the question of the user.
80  *
81  * @param question the question of the user
82  */
83
84 /**
85  * Returns the user object.
86  *
87  * @return the user object
88  */
89
90 /**
91  * Sets the user object.
92  *
93  * @param user the user object
94  */
95
96 /**
97  * Returns the security question of the user.
98  *
99  * @return the security question of the user
100 */
101
102 /**
103  * Sets the security question of the user.
104  *
105  * @param question the security question of the user
106  */
107
108 /**
109  * Returns the answer of the user.
110  *
111  * @return the answer of the user
112  */
113
114 /**
115  * Sets the answer of the user.
116  *
117  * @param answer the answer of the user
118  */
119
120 /**
121  * Returns the user object.
122  *
123  * @return the user object
124  */
125
126 /**
127  * Sets the user object.
128  *
129  * @param user the user object
130  */
131
132 /**
133  * Returns the user object.
134  *
135  * @return the user object
136  */
137
138 /**
139  * Sets the user object.
140  *
141  * @param user the user object
142  */
143
144 /**
145  * Returns the user object.
146  *
147  * @return the user object
148  */
149
150 /**
151  * Sets the user object.
152  *
153  * @param user the user object
154  */
155
156 /**
157  * Returns the user object.
158  *
159  * @return the user object
160  */
161
162 /**
163  * Sets the user object.
164  *
165  * @param user the user object
166  */
167
168 /**
169  * Returns the user object.
170  *
171  * @return the user object
172  */
173
174 /**
175  * Sets the user object.
176  *
177  * @param user the user object
178  */
179
180 /**
181  * Returns the user object.
182  *
183  * @return the user object
184  */
185
186 /**
187  * Sets the user object.
188  *
189  * @param user the user object
190  */
191
192 /**
193  * Returns the user object.
194  *
195  * @return the user object
196  */
197
198 /**
199  * Sets the user object.
200  *
201  * @param user the user object
202  */
203
204 /**
205  * Returns the user object.
206  *
207  * @return the user object
208  */
209
210 /**
211  * Sets the user object.
212  *
213  * @param user the user object
214  */
215
216 /**
217  * Returns the user object.
218  *
219  * @return the user object
220  */
221
222 /**
223  * Sets the user object.
224  *
225  * @param user the user object
226  */
227
228 /**
229  * Returns the user object.
230  *
231  * @return the user object
232  */
233
234 /**
235  * Sets the user object.
236  *
237  * @param user the user object
238  */
239
240 /**
241  * Returns the user object.
242  *
243  * @return the user object
244  */
245
246 /**
247  * Sets the user object.
248  *
249  * @param user the user object
250  */
251
252 /**
253  * Returns the user object.
254  *
255  * @return the user object
256  */
257
258 /**
259  * Sets the user object.
260  *
261  * @param user the user object
262  */
263
264 /**
265  * Returns the user object.
266  *
267  * @return the user object
268  */
269
270 /**
271  * Sets the user object.
272  *
273  * @param user the user object
274  */
275
276 /**
277  * Returns the user object.
278  *
279  * @return the user object
280  */
281
282 /**
283  * Sets the user object.
284  *
285  * @param user the user object
286  */
287
288 /**
289  * Returns the user object.
290  *
291  * @return the user object
292  */
293
294 /**
295  * Sets the user object.
296  *
297  * @param user the user object
298  */
299
300 /**
301  * Returns the user object.
302  *
303  * @return the user object
304  */
305
306 /**
307  * Sets the user object.
308  *
309  * @param user the user object
310  */
311
312 /**
313  * Returns the user object.
314  *
315  * @return the user object
316  */
317
318 /**
319  * Sets the user object.
320  *
321  * @param user the user object
322  */
323
324 /**
325  * Returns the user object.
326  *
327  * @return the user object
328  */
329
330 /**
331  * Sets the user object.
332  *
333  * @param user the user object
334  */
335
336 /**
337  * Returns the user object.
338  *
339  * @return the user object
340  */
341
342 /**
343  * Sets the user object.
344  *
345  * @param user the user object
346  */
347
348 /**
349  * Returns the user object.
350  *
351  * @return the user object
352  */
353
354 /**
355  * Sets the user object.
356  *
357  * @param user the user object
358  */
359
360 /**
361  * Returns the user object.
362  *
363  * @return the user object
364  */
365
366 /**
367  * Sets the user object.
368  *
369  * @param user the user object
370  */
371
372 /**
373  * Returns the user object.
374  *
375  * @return the user object
376  */
377
378 /**
379  * Sets the user object.
380  *
381  * @param user the user object
382  */
383
384 /**
385  * Returns the user object.
386  *
387  * @return the user object
388  */
389
390 /**
391  * Sets the user object.
392  *
393  * @param user the user object
394  */
395
396 /**
397  * Returns the user object.
398  *
399  * @return the user object
400  */
401
402 /**
403  * Sets the user object.
404  *
405  * @param user the user object
406  */
407
408 /**
409  * Returns the user object.
410  *
411  * @return the user object
412  */
413
414 /**
415  * Sets the user object.
416  *
417  * @param user the user object
418  */
419
420 /**
421  * Returns the user object.
422  *
423  * @return the user object
424  */
425
426 /**
427  * Sets the user object.
428  *
429  * @param user the user object
430  */
431
432 /**
433  * Returns the user object.
434  *
435  * @return the user object
436  */
437
438 /**
439  * Sets the user object.
440  *
441  * @param user the user object
442  */
443
444 /**
445  * Returns the user object.
446  *
447  * @return the user object
448  */
449
450 /**
451  * Sets the user object.
452  *
453  * @param user the user object
454  */
455
456 /**
457  * Returns the user object.
458  *
459  * @return the user object
460  */
461
462 /**
463  * Sets the user object.
464  *
465  * @param user the user object
466  */
467
468 /**
469  * Returns the user object.
470  *
471  * @return the user object
472  */
473
474 /**
475  * Sets the user object.
476  *
477  * @param user the user object
478  */
479
480 /**
481  * Returns the user object.
482  *
483  * @return the user object
484  */
485
486 /**
487  * Sets the user object.
488  *
489  * @param user the user object
490  */
491
492 /**
493  * Returns the user object.
494  *
495  * @return the user object
496  */
497
498 /**
499  * Sets the user object.
500  *
501  * @param user the user object
502  */
503
504 /**
505  * Returns the user object.
506  *
507  * @return the user object
508  */
509
510 /**
511  * Sets the user object.
512  *
513  * @param user the user object
514  */
515
516 /**
517  * Returns the user object.
518  *
519  * @return the user object
520  */
521
522 /**
523  * Sets the user object.
524  *
525  * @param user the user object
526  */
527
528 /**
529  * Returns the user object.
530  *
531  * @return the user object
532  */
533
534 /**
535  * Sets the user object.
536  *
537  * @param user the user object
538  */
539
540 /**
541  * Returns the user object.
542  *
543  * @return the user object
544  */
545
546 /**
547  * Sets the user object.
548  *
549  * @param user the user object
550  */
551
552 /**
553  * Returns the user object.
554  *
555  * @return the user object
556  */
557
558 /**
559  * Sets the user object.
560  *
561  * @param user the user object
562  */
563
564 /**
565  * Returns the user object.
566  *
567  * @return the user object
568  */
569
570 /**
571  * Sets the user object.
572  *
573  * @param user the user object
574  */
575
576 /**
577  * Returns the user object.
578  *
579  * @return the user object
580  */
581
582 /**
583  * Sets the user object.
584  *
585  * @param user the user object
586  */
587
588 /**
589  * Returns the user object.
590  *
591  * @return the user object
592  */
593
594 /**
595  * Sets the user object.
596  *
597  * @param user the user object
598  */
599
600 /**
601  * Returns the user object.
602  *
603  * @return the user object
604  */
605
606 /**
607  * Sets the user object.
608  *
609  * @param user the user object
610  */
611
612 /**
613  * Returns the user object.
614  *
615  * @return the user object
616  */
617
618 /**
619  * Sets the user object.
620  *
621  * @param user the user object
622  */
623
624 /**
625  * Returns the user object.
626  *
627  * @return the user object
628  */
629
630 /**
631  * Sets the user object.
632  *
633  * @param user the user object
634  */
635
636 /**
637  * Returns the user object.
638  *
639  * @return the user object
640  */
641
642 /**
643  * Sets the user object.
644  *
645  * @param user the user object
646  */
647
648 /**
649  * Returns the user object.
650  *
651  * @return the user object
652  */
653
654 /**
655  * Sets the user object.
656  *
657  * @param user the user object
658  */
659
660 /**
661  * Returns the user object.
662  *
663  * @return the user object
664  */
665
666 /**
667  * Sets the user object.
668  *
669  * @param user the user object
670  */
671
672 /**
673  * Returns the user object.
674  *
675  * @return the user object
676  */
677
678 /**
679  * Sets the user object.
680  *
681  * @param user the user object
682  */
683
684 /**
685  * Returns the user object.
686  *
687  * @return the user object
688  */
689
690 /**
691  * Sets the user object.
692  *
693  * @param user the user object
694  */
695
696 /**
697  * Returns the user object.
698  *
699  * @return the user object
700  */
701
702 /**
703  * Sets the user object.
704  *
705  * @param user the user object
706  */
707
708 /**
709  * Returns the user object.
710  *
711  * @return the user object
712  */
713
714 /**
715  * Sets the user object.
716  *
717  * @param user the user object
718  */
719
720 /**
721  * Returns the user object.
722  *
723  * @return the user object
724  */
725
726 /**
727  * Sets the user object.
728  *
729  * @param user the user object
730  */
731
732 /**
733  * Returns the user object.
734  *
735  * @return the user object
736  */
737
738 /**
739  * Sets the user object.
740  *
741  * @param user the user object
742  */
743
744 /**
745  * Returns the user object.
746  *
747  * @return the user object
748  */
749
750 /**
751  * Sets the user object.
752  *
753  * @param user the user object
754  */
755
756 /**
757  * Returns the user object.
758  *
759  * @return the user object
760  */
761
762 /**
763  * Sets the user object.
764  *
765  * @param user the user object
766  */
767
768 /**
769  * Returns the user object.
770  *
771  * @return the user object
772  */
773
774 /**
775  * Sets the user object.
776  *
777  * @param user the user object
778  */
779
780 /**
781  * Returns the user object.
782  *
783  * @return the user object
784  */
785
786 /**
787  * Sets the user object.
788  *
789  * @param user the user object
790  */
791
792 /**
793  * Returns the user object.
794  *
795  * @return the user object
796  */
797
798 /**
799  * Sets the user object.
800  *
801  * @param user the user object
802  */
803
804 /**
805  * Returns the user object.
806  *
807  * @return the user object
808  */
809
810 /**
811  * Sets the user object.
812  *
813  * @param user the user object
814  */
815
816 /**
817  * Returns the user object.
818  *
819  * @return the user object
820  */
821
822 /**
823  * Sets the user object.
824  *
825  * @param user the user object
826  */
827
828 /**
829  * Returns the user object.
830  *
831  * @return the user object
832  */
833
834 /**
835  * Sets the user object.
836  *
837  * @param user the user object
838  */
839
840 /**
841  * Returns the user object.
842  *
843  * @return the user object
844  */
845
846 /**
847  * Sets the user object.
848  *
849  * @param user the user object
850  */
851
852 /**
853  * Returns the user object.
854  *
855  * @return the user object
856  */
857
858 /**
859  * Sets the user object.
860  *
861  * @param user the user object
862  */
863
864 /**
865  * Returns the user object.
866  *
867  * @return the user object
868  */
869
870 /**
871  * Sets the user object.
872  *
873  * @param user the user object
874  */
875
876 /**
877  * Returns the user object.
878  *
879  * @return the user object
880  */
881
882 /**
883  * Sets the user object.
884  *
885  * @param user the user object
886  */
887
888 /**
889  * Returns the user object.
890  *
891  * @return the user object
892  */
893
894 /**
895  * Sets the user object.
896  *
897  * @param user the user object
898  */
899
900 /**
901  * Returns the user object.
902  *
903  * @return the user object
904  */
905
906 /**
907  * Sets the user object.
908  *
909  * @param user the user object
910  */
911
912 /**
913  * Returns the user object.
914  *
915  * @return the user object
916  */
917
918 /**
919  * Sets the user object.
920  *
921  * @param user the user object
922  */
923
924 /**
925  * Returns the user object.
926  *
927  * @return the user object
928  */
929
930 /**
931  * Sets the user object.
932  *
933  * @param user the user object
934  */
935
936 /**
937  * Returns the user object.
938  *
939  * @return the user object
940  */
941
942 /**
943  * Sets the user object.
944  *
945  * @param user the user object
946  */
947
948 /**
949  * Returns the user object.
950  *
951  * @return the user object
952  */
953
954 /**
955  * Sets the user object.
956  *
957  * @param user the user object
958  */
959
960 /**
961  * Returns the user object.
962  *
963  * @return the user object
964  */
965
966 /**
967  * Sets the user object.
968  *
969  * @param user the user object
970  */
971
972 /**
973  * Returns the user object.
974  *
975  * @return the user object
976  */
977
978 /**
979  * Sets the user object.
980  *
981  * @param user the user object
982  */
983
984 /**
985  * Returns the user object.
986  *
987  * @return the user object
988  */
989
990 /**
991  * Sets the user object.
992  *
993  * @param user the user object
994  */
995
996 /**
997  * Returns the user object.
998  *
999  * @return the user object
1000  */
1001
1002 /**
1003  * Sets the user object.
1004  *
1005  * @param user the user object
1006  */
1007
1008 /**
1009  * Returns the user object.
1010  *
1011  * @return the user object
1012  */
1013
1014 /**
1015  * Sets the user object.
1016  *
1017  * @param user the user object
1018  */
1019
1020 /**
1021  * Returns the user object.
1022  *
1023  * @return the user object
1024  */
1025
1026 /**
1027  * Sets the user object.
1028  *
1029  * @param user the user object
1030  */
1031
1032 /**
1033  * Returns the user object.
1034  *
1035  * @return the user object
1036  */
1037
1038 /**
1039  * Sets the user object.
1040  *
1041  * @param user the user object
1042  */
1043
1044 /**
1045  * Returns the user object.
1046  *
1047  * @return the user object
1048  */
1049
1050 /**
1051  * Sets the user object.
1052  *
1053  * @param user the user object
1054  */
1055
1056 /**
1057  * Returns the user object.
1058  *
1059  * @return the user object
1060  */
1061
1062 /**
1063  * Sets the user object.
1064  *
1065  * @param user the user object
1066  */
1067
1068 /**
1069  * Returns the user object.
1070  *
1071  * @return the user object
1072  */
1073
1074 /**
1075  * Sets the user object.
1076  *
1077  * @param user the user object
1078  */
1079
1080 /**
1081  * Returns the user object.
1082  *
1083  * @return the user object
1084  */
1085
1086 /**
1087  * Sets the user object.
1088  *
1089  * @param user the user object
1090  */
1091
1092 /**
1093  * Returns the user object.
1094  *
1095  * @return the user object
1096  */
1097
1098 /**
1099  * Sets the user object.
1100  *
1101  * @param user the user object
1102  */
1103
1104 /**
1105  * Returns the user object.
1106  *
1107  * @return the user object
1108  */
1109
1110 /**
1111  * Sets the user object.
1112  *
1113  * @param user the user object
1114  */
1115
1116 /**
1117  * Returns the user object.
1118  *
1119  * @return the user object
1120  */
1121
1122 /**
1123  * Sets the user object.
1124  *
1125  * @param user the user object
1126  */
1127
1128 /**
1129  * Returns the user object.
1130  *
1131  * @return the user object
1132  */
1133
1134 /**
1135  * Sets the user object.
1136  *
1137  * @param user the user object
1138  */
1139
1140 /**
1141  * Returns the user object.
1142  *
1143  * @return the user object
1144  */
1145
1146 /**
1147  * Sets the user object.
1148  *
1149  * @param user the user object
1150  */
1151
1152 /**
1153  * Returns the user object.
1154  *
1155  * @return the user object
1156  */
1157
1158 /**
1159  * Sets the user object.
1160  *
1161  * @param user the user object
1162  */
1163
1164 /**
1165  * Returns the user object.
1166  *
1167  * @return the user object
1168  */
1169
1170 /**
1171  * Sets the user object.
1172  *
1173  * @param user the user object
1174  */
1175
1176 /**
1177  * Returns the user object.
1178  *
1179  * @return the user object
1180  */
1181
1182 /**
1183  * Sets the user object.
1184  *
1185  * @param user the user object
1186  */
1187
1188 /**
1189  * Returns the user object.
1190  *
1191  * @return the user object
1192  */
1193
1194 /**
1195  * Sets the user object.
1196  *
1197  * @param user the user object
1198  */
1199
1200 /**
1201  * Returns the user object.
1202  *
1203  * @return the user object
1204  */
1205
1206 /**
1207  * Sets the user object.
1208  *
1209  * @param user the user object
1210  */
1211
1212 /**
1213  * Returns the user object.
1214  *
1215  * @return the user object
1216  */
1217
1218 /**
1219  * Sets the user object.
1220  *
1221  * @param user the user object
1222  */
1223
1224 /**
1225  * Returns the user object.
1226  *
1227  * @return the user object
1228  */
1229
1230 /**
1231  * Sets the user object.
1232  *
1233  * @param user the user object
1234  */
1235
1236 /**
1237  * Returns the user object.
1238  *
1239  * @return the user object
1240  */
1241
1242 /**
1243  * Sets the user object.
1244  *
1245  * @param user the user object
1246  */
1247
1248 /**
1249  * Returns the user object.
1250  *
1251  * @return the user object
1252  */
1253
1254 /**
1255  * Sets the user object.
1256  *
1257  * @param user the user object
1258  */
1259
1260 /**
1261  * Returns the user object.
1262  *
1263  * @return the user object
1264  */
1265
1266 /**
1267  * Sets the user object.
1268  *
1269  * @param user the user object
1270  */
1271
1272 /**
1273  * Returns the user object.
1274  *
1275  * @return the user object
1276  */
1277
1278 /**
1279  * Sets the user object.
1280  *
1281  * @param user the user object
1282  */
1283
1284 /**
1285  * Returns the user object.
1286  *
1287  * @return the user object
1288  */
1289
1290 /**
1291  * Sets the user object.
1292  *
1293  * @param user the user object
1294  */
1295
1296 /**
1297  * Returns the user object.
1298  *
1299  * @return the user object
1300  */
1301
1302 /**
1303  * Sets the user object.
1304  *
1305  * @param user the user object
1306  */
1307
1308 /**
1309  * Returns the user object.
1310  *
1311  * @return the user object
1312  */
1313
1314 /**
1315  * Sets the user object.
1316  *
1317  * @param user the user object
1318  */
1319
1320 /**
1321  * Returns the user object.
1322  *
1323  * @return the user object
1324  */
1325
1326 /**
1327  * Sets the user object.
1328  *
1329  * @param user the user object
1330  */
1331
1332 /**
1333  * Returns the user object.
1334  *
1335  * @return the user object
1336  */
1337
1338 /**
1339  * Sets the user object.
1340  *
1341  * @param user the user object
1342  */
1343
1344 /**
1345  * Returns the user object.
1346  *
1347  * @return the user object
1348  */
1349
1350 /**
1351  * Sets the user object.
1352  *
1353  * @param user the user object
1354  */
1355
1356 /**
1357  * Returns the user object.
1358  *
1359  * @return the user object
1360  */
1361
1362 /**
1363  * Sets the user object.
1364  *
1365  * @param user the user object
1366  */
1367
1368 /**
1369  * Returns the user object.
1370  *
1371  * @return the user object
1372  */
1373
1374 /**
1375  * Sets the user object.
1376  *
1377  * @param user the user object
1378  */
1379
1380 /**
1381  * Returns the user object.
1382  *
1383  * @return the user object
1384  */
1385
1386 /**
1387  * Sets the user object.
1388  *
1389  * @param user the user object
1390  */
1391
1392 /**
1393  * Returns the user object.
1394  *
1395  * @return the user object
1396  */
1397
1398 /**
1399  * Sets the user object.
1400  *
1401  * @param user the user object
1402  */
1403
1404 /**
1405  * Returns the user object.
1406  *
1407  * @return the user object
1408  */
1409
1410 /**
1411  * Sets the user object.
1412  *
1413  * @param user the user object
1414  */
1415
1416 /**
1417  * Returns the user object.
1418  *
1419  * @return the user object
1420  */
1421
1422 /**
1423  * Sets the user object.
1424  *
1425  * @param user the user object
1426  */
1427
1428 /**
1429  * Returns the user object.
1430  *
1431  * @return the user object
1432  */
1433
1434 /**
1435  * Sets the user object.
1436  *
1437  * @param user the user object
1438  */
1439
1440 /**
1441  * Returns the user object.
1442  *
1443  * @return the user object
1444  */
1445
1446 /**
1447  * Sets the user object.
1448  *
1449  * @param user the user object
1450  */
1451
1452 /**
1453  * Returns the user object.
1454  *
1455  * @return the user object
1456  */
1457
1458 /**
1459  * Sets the user object.
1460  *
1461  * @param user the user object
1462  */
1463
1464 /**
1465  * Returns the user object.
1466  *
1467  * @return the user object
1468  */
1469
1470 /**
1471  * Sets the user object.
1472  *
1473  * @param user the user object
1474  */
1475
1476 /**
1477  * Returns the user object.
1478  *
1479  * @return the user object
1480  */
1481
1482 /**
1483  * Sets the user object.
1484  *
1485  * @param user the user object
1486  */
1487
1488 /**
1489  * Returns the user object.
1490  *
1491  * @return the user object
1492  */
1493
1494 /**
1495  * Sets the user object.
1496  *
1497  * @param user the user object
1498  */
1499
1500 /**
1501  * Returns the user object.
1502  *
1503  * @return the user object
1504  */
1505
1506 /**
1507  * Sets the user object.
1508  *
1509  * @param user the user object
1510  */
1511
1512 /**
15
```

The screenshot shows a Java project structure on the left and a code editor on the right. The project tree includes a .idea folder, logs, src (containing main, java, authentication, database, utils, RunApplication, resources, and test), target, .gitignore, pom.xml, External Libraries, and Scratches and Consoles. The code editor displays `CreateTable.java` with the following content:

```
package database;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * The CreateTable class is responsible for creating tables in a database.
 */
public class CreateTable {
    /**
     * Creates a table based on the given query.
     *
     * @param query the CREATE TABLE query
     */
    public static void createTable(String query) {...}
}
```

Figure 5: CreateTable class- JavaDocs specification for commenting styles.

```
Project ▾ RunApplication.java SelectTable.java CreateTable.java DeleteTable.java ×  
CSCI5408_A2 ~/ideaProjects/CSCI540  
1 usage  
private static final String FILE_NAME = "delete-table_data.txt";  
3 usages  
private static final String DELIMITER = "\n";  
/**  
 * Deletes rows from a table based on the given query.  
 *  
 * @param query the DELETE query  
 */  
1 usage  
20 > public static void deleteTable(String query) {...}  
73  
74 /**  
 * Checks if the WHERE clause matches the row.  
 *  
 * @param whereClause the WHERE clause of the DELETE query  
 * @param columns the columns of the table  
 * @param values the values of the row  
 * @return true if the WHERE clause matches the row, false otherwise  
 */  
75 1 usage  
76  
77 private static boolean matchesWhereClause(String whereClause, String[] columns, String[] values) {...}  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107
```

Figure 6: Delete Table class- JavaDocs specification for commenting styles.

The screenshot shows the IntelliJ IDEA interface. On the left is the Project tool window displaying the project structure under 'CSCI5408_A2'. The 'src' folder contains 'main' and 'utils' packages. 'main' contains 'java' and 'resources' folders. 'java' contains 'authentication' and 'database' packages. 'authentication' contains 'User', 'UserAuthenticationModule', and 'UserFileHandler' classes. 'database' contains 'CreateTable', 'DeleteTable', 'InsertTable', 'SelectTable', and 'UpdateTable' classes. The 'InsertTable' class is selected in the Project tool window. On the right is the Editor tool window showing the code for 'InsertTable.java'. The code is annotated with JavaDoc comments. The line 18 is highlighted with a yellow background.

```

1 package database;
2
3 import java.io.*;
4 import java.util.regex.Matcher;
5 import java.util.regex.Pattern;
6
7 /**
8 * The InsertTable class is responsible for inserting values into a table in a database.
9 */
10 usage
11 public class InsertTable {
12     1 usage
13     private static final String FILE_NAME = "logs/insert-table_data.txt";
14     1 usage
15     private static final String DELIMITER = "\n";
16
17     /**
18      * Inserts values into a table based on the given query.
19      *
20      * @param query the INSERT query
21      */
22     1 usage
23     public static void insertTable(String query) {...}

```

Figure 7: Insert Table class- JavaDocs specification for commenting styles.

The screenshot shows the IntelliJ IDEA interface. On the left is the Project tool window displaying the project structure under 'CSCI5408_A2'. The 'src' folder contains 'main' and 'utils' packages. 'main' contains 'java' and 'resources' folders. 'java' contains 'authentication' and 'database' packages. 'authentication' contains 'User', 'UserAuthenticationModule', and 'UserFileHandler' classes. 'database' contains 'CreateTable', 'DeleteTable', 'InsertTable', 'SelectTable', and 'UpdateTable' classes. The 'SelectTable' class is selected in the Project tool window. On the right is the Editor tool window showing the code for 'SelectTable.java'. The code is annotated with JavaDoc comments. The line 22 is highlighted with a yellow background.

```

15 usage
16
17 private static final String ROW_DELIMITER = "\r";
18
19 /**
20 * Retrieves and displays table information based on the given SELECT query.
21 *
22 * @param query the SELECT query
23 */
24 1 usage
25 public static void selectTable(String query) {...}
26
27 /**
28 * Evaluates the condition based on the table columns and values.
29 *
30 * @param condition the condition of the WHERE clause
31 * @param tableColumns the columns of the table
32 * @param values the values of the row
33 * @return true if the condition is met, false otherwise
34 */
35 1 usage
36 private static boolean evaluateCondition(String condition, String tableColumns, String[] values) {...}
37
38 /**
39 * Gets the index of the column in the table columns.
40 *
41 * @param columnName the name of the column
42 * @param tableColumns the columns of the table
43 * @return the index of the column in the table columns, or -1 if not found
44 */
45 2 usages
46 private static int getColumnIndex(String columnName, String tableColumns) {...}
47
48 }

```

Figure 8: Select Table class- JavaDocs specification for commenting styles.

```

Project ▾
  CSCI5408_A2 ~/IdeaProjects/CSCI540
    > .idea
    > logs
    > src
      > main
        > java
          > authentication
            ○ User
            ○ UserAuthenticationModule
            ○ UserFileHandler
          > database
            ○ CreateTable
            ○ DeleteTable
            ○ InsertTable
            ○ SelectTable
            ○ UpdateTable
          > utils
            ○ RunApplication
          resources
        > test
      > target
      ○ .gitignore
      M pom.xml
    > External Libraries
    ≡ Scratches and Consoles

UpdateTable.java ×


```

7 /**
8 * The UpdateTable class is responsible for updating data in a table in a database.
9 */
10 public class UpdateTable {
11 1 usage
12 private static final String FILE_NAME = "logs/update-table_data.txt";
13 3 usages
14 private static final String DELIMITER = "\n";
15
16 /**
17 * Updates the data in a table based on the given query.
18 *
19 * @param query the UPDATE query
20 */
21 1 usage
22 public static void updateTable(String query) {...}
23
24 /**
25 * Checks if the WHERE clause matches the row.
26 *
27 * @param whereClause the WHERE clause of the UPDATE query
28 * @param columns the columns of the table
29 * @param values the values of the row
30 * @return true if the WHERE clause matches the row, false otherwise
31 */
32 1 usage
33 private static boolean matchesWhereClause(String whereClause, String[] columns, String[] values) {...}
34
35 /**
36 * Updates the values based on the SET clause.
37 *
38 * @param setClause the SET clause
39 */
40
41 }

```


```

Figure 9: Update Table class- JavaDocs specification for commenting styles.

```

Project ▾
  CSCI5408_A2 ~/IdeaProjects/CSCI540
    > .idea
    > logs
    > src
      > main
        > java
          > authentication
            ○ User
            ○ UserAuthenticationModule
            ○ UserFileHandler
          > database
            ○ CreateTable
            ○ DeleteTable
            ○ InsertTable
            ○ SelectTable
            ○ UpdateTable
          > utils
            ○ Utils
            ○ RunApplication
          resources
        > test
      > target
      ○ .gitignore
      M pom.xml
    > External Libraries
    ≡ Scratches and Consoles

Utils.java ×


```

1 package utils;
2
3 import java.nio.charset.StandardCharsets;
4 import java.security.MessageDigest;
5 import java.security.NoSuchAlgorithmException;
6
7 /**
8 * Utility class for common operations.
9 */
10 public class Utils {
11
12 /**
13 * Hashes a string using the MD5 algorithm.
14 *
15 * @param input the input string to be hashed
16 * @return the hashed string
17 */
18 2 usages
19 public static String hashString(String input) {
20 try (...) catch (NoSuchAlgorithmException e) {
21 System.out.println("Error hashing string: " + e.getMessage());
22 return null;
23 }
24 }
25
26 }

```


```

Figure 10: Utils class- JavaDocs specification for commenting styles.

```

Project ▾
  CSCI5408_A2 ~/IdeaProjects/CSCI5408_A2
    .idea
    logs
    src
      main
        java
          authentication
            User
            UserAuthenticationModule
            UserFileHandler
        database
          CreateTable
          DeleteTable
          InsertTable
          SelectTable
          UpdateTable
        utils
          Utils
        RunApplication
      resources
    test
    target
    .gitignore
    pom.xml
  External Libraries
  Scratches and Consoles

RunApplication.java ×
import authentication.UserAuthenticationModule;
import database.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/*
 * The RunApplication class is responsible for running the application and handling user queries.
 */
public class RunApplication {
    public static void main(String[] args) {
        // Create an instance of the UserAuthenticationModule
        UserAuthenticationModule module = new UserAuthenticationModule();

        // Start the authentication module
        module.start();
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String query;
        boolean exit = false;

        while (!exit) {
            try {
                System.out.println("-----");
                System.out.println("Enter your query in the following format:");
                System.out.println("Create Table: create table <table_name> (<column1>, <column2>, ...);");
                System.out.println("Insert Data: insert into <table_name> values (<value1>, <value2>, ...);");
                System.out.println("Select Data: select * from <table_name> where <condition>;");
                System.out.println("Update Data: update <table_name> set <column_name>=<new_value> where <condition>;");
                System.out.println("Type 'exit' to exit the application.");
                System.out.println("-----");
                query = reader.readLine();
            }

```

Figure 11: RunApplication class- JavaDocs specification for commenting styles.

3. The application code demonstrates some of the **SOLID principles** [3]. These design principles help promote code organization, maintainability, extensibility, and reusability, making the application more robust and easier to understand and maintain. The design principles used in the application program development/execution are:

a. Single Responsibility Principle (SRP):

- The UserAuthenticationModule class is responsible for handling user authentication.
- The User class represents a user with authentication information.
- The UserFileHandler class is responsible for handling user file operations.
- The CreateTable, InsertTable, SelectTable, UpdateTable, and DeleteTable classes are responsible for their respective database operations.

b. Open/Closed Principle (OCP):

- The UserAuthenticationModule class is open for extension but closed for modification. It can be extended to support additional authentication features without modifying its existing code.

c. Liskov Substitution Principle (LSP):

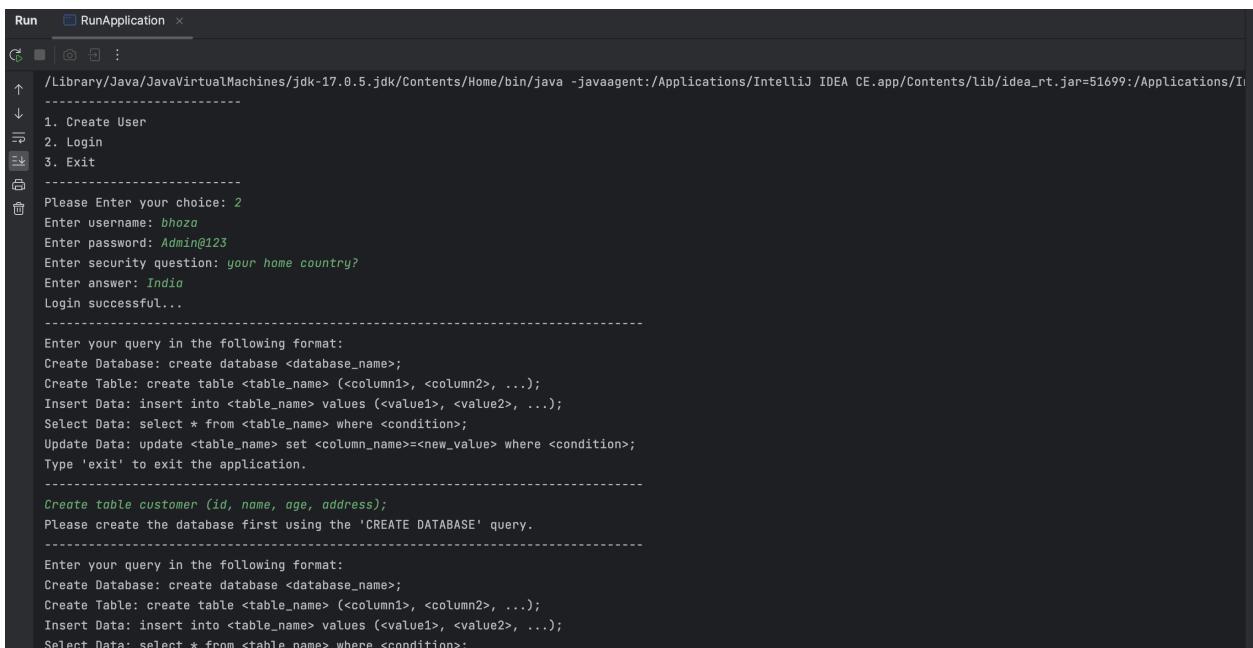
- The UserAuthenticationModule class uses the UserFileHandler class as a dependency, following the LSP. It can use any subclass of UserFileHandler without issues.

d. Dependency Inversion Principle (DIP):

- The UserAuthenticationModule class depends on abstractions (interfaces) like Map and Scanner, instead of concrete implementations.
- The UserAuthenticationModule class depends on the UserFileHandler class, which is an abstraction, allowing for flexibility in choosing different file handling methods.

Additionally:

- The code separates different concerns into separate classes and modules. For example, the authentication functionality is encapsulated in the UserAuthenticationModule, and the database operations are handled by separate database-related classes.
 - The code uses encapsulation to hide internal details and expose only necessary information through getters and setters. For example, the User class encapsulates the user's authentication information, and the UserAuthenticationModule provides methods for creating users, logging in, and checking authentication status.
4. The application is totally **console-based** (no GUI added) and it accept user input in the form of SQL query, once the user has successfully logged in.



```

Run  RunApplication x
/Library/Java/JavaVirtualMachines/jdk-17.0.5.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=51699:/Applications/I
-----
1. Create User
2. Login
3. Exit
-----
Please Enter your choice: 2
Enter username: bhoxa
Enter password: Admin@123
Enter security question: your home country?
Enter answer: India
Login successful...

Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

Create table customer (id, name, age, address);
Please create the database first using the 'CREATE DATABASE' query.

Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;

```

Figure 12: Console based User authentication program.

```

Run Application ×
Help : 
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.
-----
Create database DMWA-Assignment2
Database created: DMWA-Assignment2
Database created successfully!
-----
Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.
-----
Create table customer (id, name, age, address);
Invalid CREATE TABLE query!
-----
Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.
-----
create table customer (id, name, age, address);
Table created successfully!
-----
```

Figure 13: Console-based program which accept user input in the form of SQL query, after the user login.

5. Two factor user authentication:

The application supports multi-user authentication, which is critical for protecting sensitive data. When users create accounts, their passwords are hashed and securely stored in the database. This ensures that even if the database is compromised, the passwords remain secure.

To gain access to the database, users must enter their login credentials and answer to the security questions, which are then validated against the hashed passwords stored in the database using MD5 algorithm [4]. This adds another layer of security by preventing unauthorised access to sensitive data.

Overall, the code provides a console-based interface for user authentication and data management, ensuring the security and persistence of user information.

- The **UserAuthenticationModule** class handles user authentication by allowing users to create accounts, log in with their credentials, and perform various operations based on their authentication status.
- The **User** class represents a user with authentication information such as username, password, security question, and answer.
- The **UserFileHandler** class is responsible for loading and saving user data to a file.
- The **Utils** class provides utility methods, including hashing strings using the MD5 algorithm.
- `user_details.txt` file captures the user login details including the hashing for the password.

Working flow of User Authentication module:

- The application starts by executing the main method in the RunApplication class, which displays the menu options for the user: "1. Create User", "2. Login", and "3. Exit"

```
RunApplication x
/Library/Java/JavaVirtualMachines/jdk-17.0.5.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=52494:/Applications/IntelliJ IDEA CE.app/Contents/bin
-----
1. Create User
2. Login
3. Exit
-----
Please Enter your choice:
```

Figure 14: Menu options for User Authentication

- Let's say the user enters "1" to create a new user. They are prompted to enter a username.

```
RunApplication x
-----
| | | : 
/Library/Java/JavaVirtualMachines/jdk-17.0.5.jdk/Contents
-----
1. Create User
2. Login
3. Exit
-----
Please Enter your choice: 1
Enter username: bhavisha
```

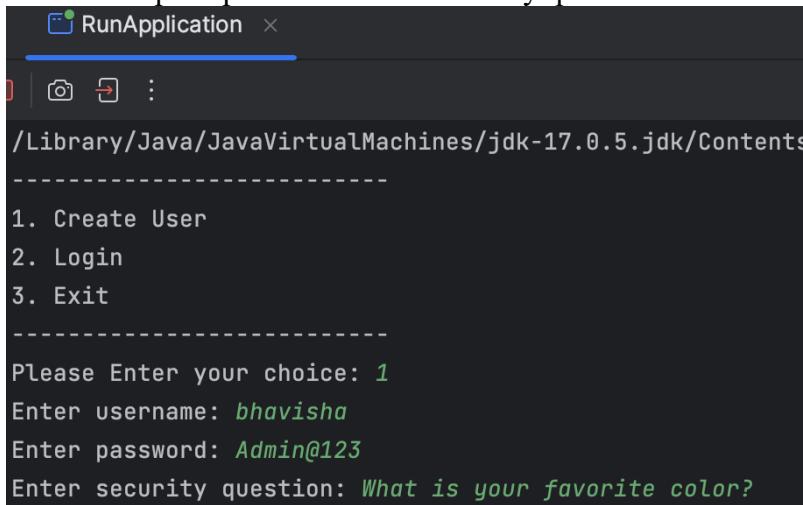
Figure 15: Starting of Register User process.

- Since it's a new user, the username is valid. The user is then prompted to enter a password. The application hashes the entered password using the Utils.hashString method for security.

```
RunApplication x
-----
| | | : 
/Library/Java/JavaVirtualMachines/jdk-17.0.5
-----
1. Create User
2. Login
3. Exit
-----
Please Enter your choice: 1
Enter username: bhavisha
Enter password: Admin@123
```

Figure 16: Username and Password entering for registration.

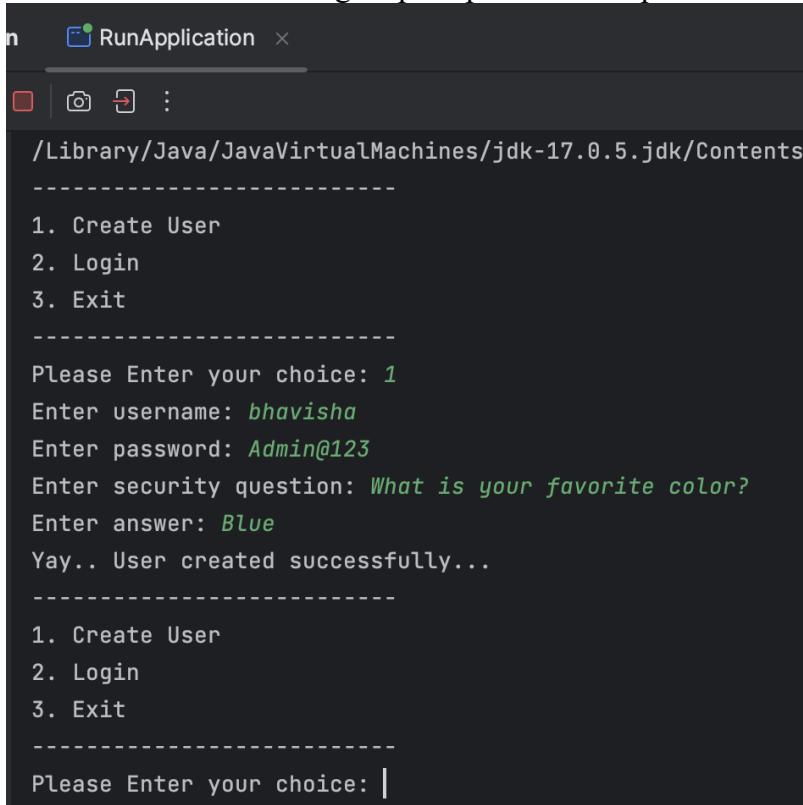
- The user is prompted to choose a security question after that.



```
/Library/Java/JavaVirtualMachines/jdk-17.0.5.jdk/Contents
-----
1. Create User
2. Login
3. Exit
-----
Please Enter your choice: 1
Enter username: bhavisha
Enter password: Admin@123
Enter security question: What is your favorite color?
```

Figure 17: Security question to enter for registration.

- Once the answer is entered, it displays a success message, indicating that the user has been created. and it will again prompt the menu options.



```
n   RunApplication ×
-----
/ Library/Java/JavaVirtualMachines/jdk-17.0.5.jdk/Contents
-----
1. Create User
2. Login
3. Exit
-----
Please Enter your choice: 1
Enter username: bhavisha
Enter password: Admin@123
Enter security question: What is your favorite color?
Enter answer: Blue
Yay.. User created successfully...
-----
1. Create User
2. Login
3. Exit
-----
Please Enter your choice: |
```

Figure 18: User registration completion

- The entered details are stored in text file named “user_details.txt” under logs directory. It supports the multiuser authentication, so it can add all data, when it is registered.

The screenshot shows the IntelliJ IDEA interface. On the left, the Project tool window displays a file structure under the 'CSCI5408_A2' project. Inside the 'logs' directory, there are several text files: 'create-table_data.txt', 'database.txt', 'insert-table_data.txt', 'select-table_data.txt', 'update-table_data.txt', and 'user_details.txt'. The 'user_details.txt' file is selected and open in the main editor window. The content of this file is:

```

1 kp::26b568e4192a164d5b3eacdbd632bc2e::kp::kp
2 bhoza::0e7517141fb53f21ee439b355b5a1d0a::your home country?::India
3 bbhatt::c5c849c1a1cfa8b09e9241b1d3ae7884::fav food?::pav-bhaji
4 bhavisha::0e7517141fb53f21ee439b355b5a1d0a::What is your favorite color?::Blue
5

```

Figure 19: User registration data stored in text file.

- Now let's say user wants to login and enters “2” on console, they are prompted to enter their username.

The screenshot shows a terminal window titled 'Run Application'. The application displays a menu with three options: 1. Create User, 2. Login, 3. Exit. The user selects option 2. The application then prompts for the username and password. The user enters 'bhavisha' and 'Admin@123'. It asks for a security question and the user answers 'Blue'. A success message 'Yay.. User created successfully...' is displayed. The application then loops back to the main menu. The user then enters choice 2 again and is prompted to enter the username.

```

un Run Application x
-----
1. Create User
2. Login
3. Exit
-----
Please Enter your choice: 1
Enter username: bhavisha
Enter password: Admin@123
Enter security question: What is your favorite color?
Enter answer: Blue
Yay.. User created successfully...
-----
1. Create User
2. Login
3. Exit
-----
Please Enter your choice: 2
Enter username: |

```

Figure 20: User Login screen.

- The application checks if the entered username exists, otherwise it will show the message of “Invalid Username!”. Since the username exists, the login process continues, and will happen same for password, security question and answer. If the answer is correct, the login process is considered successful, and a login success message is

displayed.

The screenshot shows a terminal window titled "RunApplication". The menu bar includes "Run" and "RunApplication". Below the menu are standard terminal icons for file operations. The main area displays the following text:

```
Wrong security question...
-----
1. Create User
2. Login
3. Exit
-----
Please Enter your choice: 2
Enter username: bhavisha
Enter password: Admin@123
Enter security question: What is your favorite color?
Enter answer: Blue
Login successful...
```

Figure 21: User login completion

6. Implementation of Queries (DDL & DML):

The application also implements various SQL queries such as CREATE, SELECT, INSERT, UPDATE, and DELETE on any number of tables. Separate methods have been created for each query.

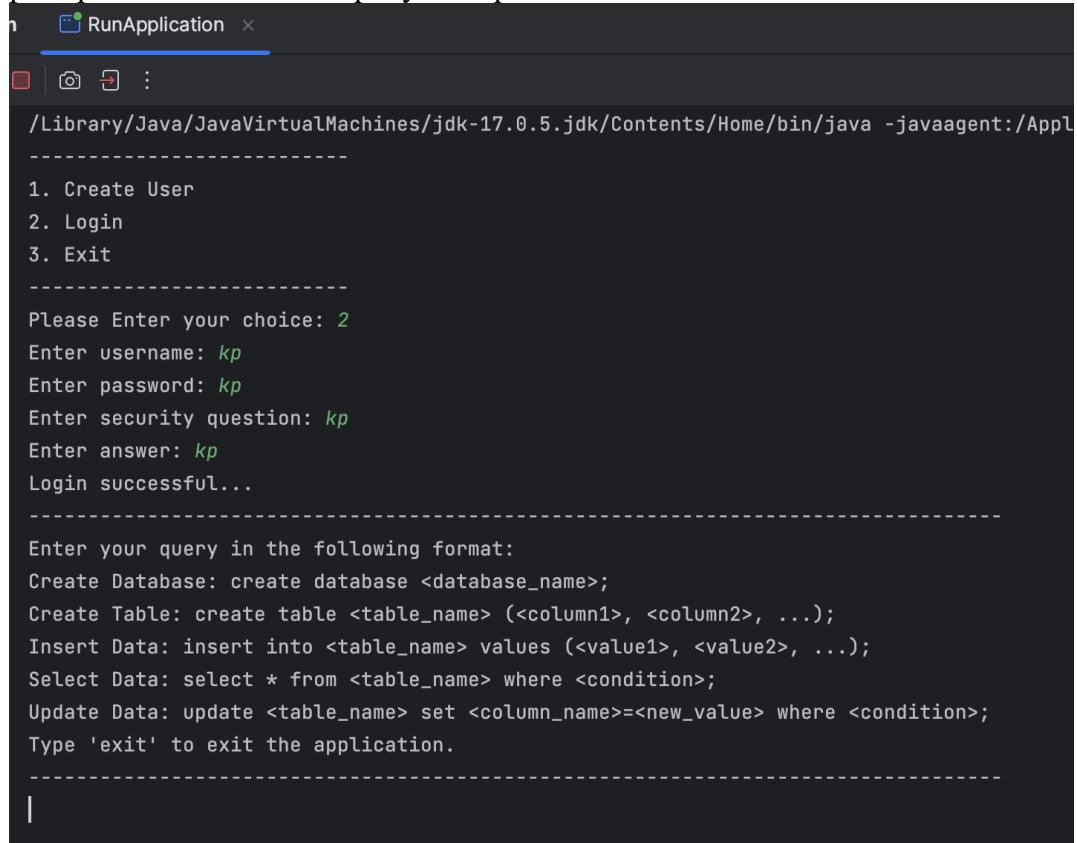
The application parses queries using **regular expressions (regex)**, providing users with a powerful and flexible way to query their database. The application's regex engine is highly efficient and can easily handle complex questions.

The application supports a diverse set of regex patterns, allowing users to query the database in a flexible and efficient manner. Users can, for example, search for records based on specific keywords or retrieve data from specific columns. This allows users to find the information quickly and efficiently they require. The concept of regex and checking if the queries are rightly written or not, have used the regex101 site which highlights the syntax which is written correctly [5].

Working flow of queries implementation (CREATE, SELECT, INSERT, UPDATE, and DELETE):

- The application starts by executing the main method in the RunApplication class, which displays the menu options for the user authentication. Once the user is logged in, it will

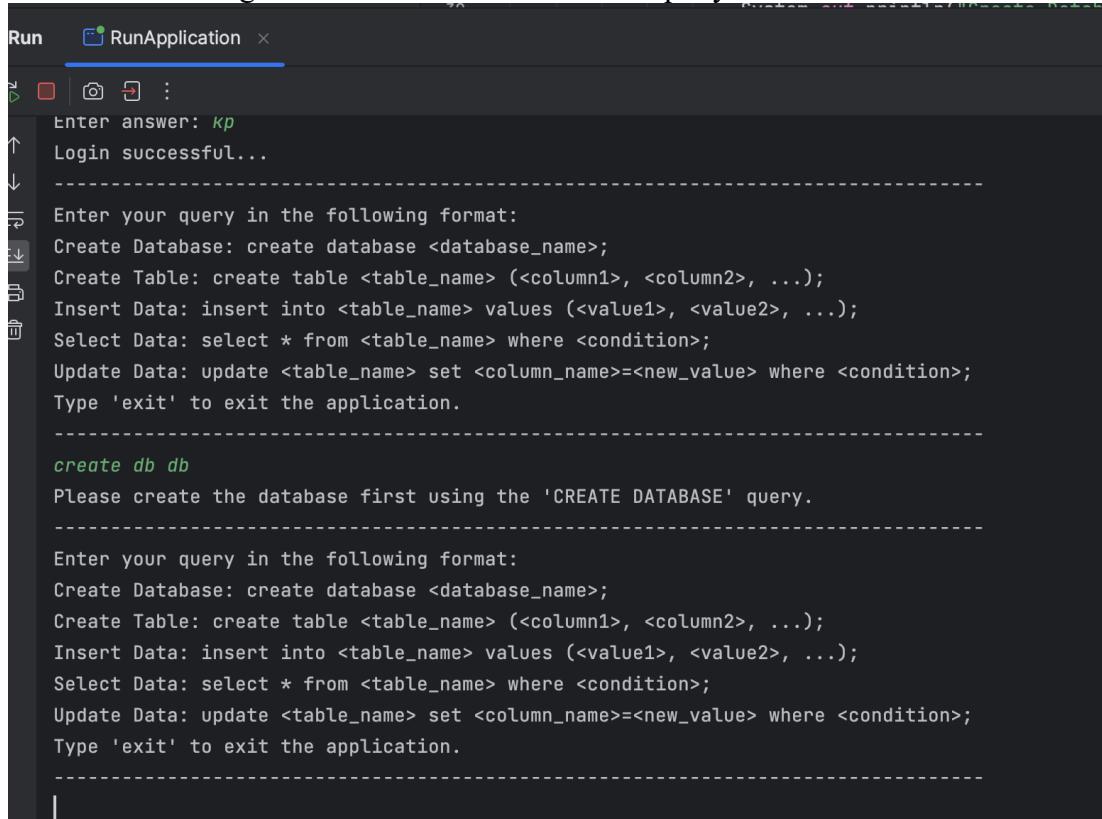
prompt the user to enter a query in a specific format.



```
RunApplication ×
/ Library/Java/JavaVirtualMachines/jdk-17.0.5.jdk/Contents/Home/bin/java -javaagent:/Appl
-----
1. Create User
2. Login
3. Exit
-----
Please Enter your choice: 2
Enter username: kp
Enter password: kp
Enter security question: kp
Enter answer: kp
Login successful...
-----
Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.
-----
```

Figure 22: Query insertion menu after login completion

- If the query is not in the given format, it will show the error message “Please create the database first using the ‘CREATE DATABASE’ query.”



```

Run RunApplication ×

Enter answer: kp
Login successful...
-----
Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

-----
create db db
Please create the database first using the 'CREATE DATABASE' query.

-----
Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

|
```

Figure 23: Error message for invalid query insertion

- Let's say the user wants to create a database and enters the query “create database my_dmwaA2_db;”. The application checks if the query starts with create database (case-insensitive). Since it matches, it continues processing and, a success message is displayed: “Database created: my_dmwaA2_db.”

Please create the database first using the 'CREATE DATABASE' query.

Enter your query in the following format:

Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

```
create database my_dmwaA2_db
Database created: my_dmwaA2_db
Database created successfully!
```

Enter your query in the following format:

Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

Figure 24: Successful Database creation

- If the database name is not null (indicating a valid query), the method proceeds to save the database name to a text file.

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project** view on the left:
 - Project name: CSCI5408_A2 (~/IdeaProjects/CSCI5408_A2)
 - Structure:
 - .idea
 - logs
 - create-database.txt
 - create-table_data.txt
 - insert-table_data.txt
 - select-table_data.txt
 - update-table_data.txt
 - user_details.txt
 - src
 - main
 - java
 - authentication
 - Run** tab at the bottom-left:
 - Selected configuration: RunApplication
 - Toolbars:
 - Run (green play button)
 - Stop (red square)
 - Run Configurations (camera icon)
 - File (document icon)
 - Help (question mark icon)
 - Terminal** window at the bottom-right:
 - Content:

```
type EXIT TO EXIT THE APPLICATION.  
-----  
create database my_dmwaA2_db  
Database created: my_dmwaA2_db  
Database created successfully!
```
 - Toolbars:
 - Up (up arrow)
 - Down (down arrow)
 - Left (left arrow)
 - Right (right arrow)

Figure 25: Create Database saved in text file.

- After processing the “create database” query, the application continues to display the menu options and wait for the next user input. If the user wants to exit, they can enter “exit” to terminate the application.

```

Run Application x
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

create database my_dmwaA2_db
Database created: my_dmwaA2_db
Database created successfully!

Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

exit

Process finished with exit code 0
|
```

Figure 26: Exit to terminate the application.

- Now let's say the user wants to create a table and enters the query “create table students (banner_id, name, course);”. The application checks if the query starts with “create table” (case-insensitive). Since it matches, it continues processing. Inside the CreateTable.createTable method, the query is parsed and validated using regular expressions (regex). The regex pattern “create\s+table\s+(\w+)\s+\((.+)\);” is used to extract the table name and the column definitions. Once the write operation is successful, a success message is displayed: “Table created successfully!”

```

n Run Application x
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

create table students (banner_id, name, course);
Table created successfully!

Enter your query in the following format:

```

Figure 27: Successful Student table creation

- If the query matches the pattern and the table name and column definitions are successfully extracted, the method proceeds to store the table information in a custom file “create-table_data.txt”.

The screenshot shows a Java IDE interface. On the left, the Project Explorer displays a package structure under 'src/main/java'. The 'database' package contains several classes: CreateDatabase, CreateTable, DeleteTable, InsertTable, SelectTable, UpdateTable, and RunApplication. The 'CreateTable' class is currently selected. On the right, there are two tabs: 'RunApplication.java' and 'create-table_data.txt'. The 'create-table_data.txt' tab contains the following text:

```

1 employee
2 id, name, salary
3 hotel
4 name, location, ratings
5 customer
6 id, name, age, address
7 students
8 banner_id, name, course
9

```

Below the tabs is a terminal window titled 'Run Application'. It shows a menu of database operations: Create Database, Create Table, Insert Data, Select Data, Update Data, and Exit. The user has entered the command 'create table students (banner_id, name, course);' and received the response 'Table created successfully!'

Figure 28: Create table saved in text file.

- After processing the “create table” query, the application continues to display the menu options and waits for the next user input. Let's say the user wants to insert values into a students table and enters the query “insert into students values (B00935827, Bhavisha_Oza, MACS);”. The application checks if the query starts with “insert into” (case-insensitive). Since it matches, it continues processing. Inside the `InsertTable.insertTable` method, the query is parsed and validated using regular expressions (regex). The regex pattern `"insert\s+into\s+(\w+)\s+values\s*\((.+)\)\s*;\s*"` is used to extract the table name and the values to be inserted. Once the write operation is successful, a success message is displayed: “Data inserted successfully!”

```

Run  RunApplication x

Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

insert into students values (B00935827, Bhavisha_Oza, MACS);
Data inserted successfully!

Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

```

Figure 29: Successful data insertion in Student table.

- If the query matches the pattern and the table name and values are successfully extracted, the method proceeds to store the inserted values in a custom file “insert-table_data.txt”.

The screenshot shows an IDE interface with the following details:

- Project View:** Shows a project named "CSCI5408_A2" containing files like ".idea", "logs", "create-database.txt", "create-table_data.txt", "insert-table_data.txt" (which is selected), "select-table_data.txt", "update-table_data.txt", and "user_details.txt".
- Run View:** Shows the "RunApplication" configuration.
- Terminal View:** Displays the same successful data insertion message as Figure 29, along with the command-line interface options.
- Content Area:** Shows the contents of the "insert-table_data.txt" file, which contains the inserted data: "B00935827, Bhavisha_Oza, MACS".

Figure 30: Insert table data saved in text file.

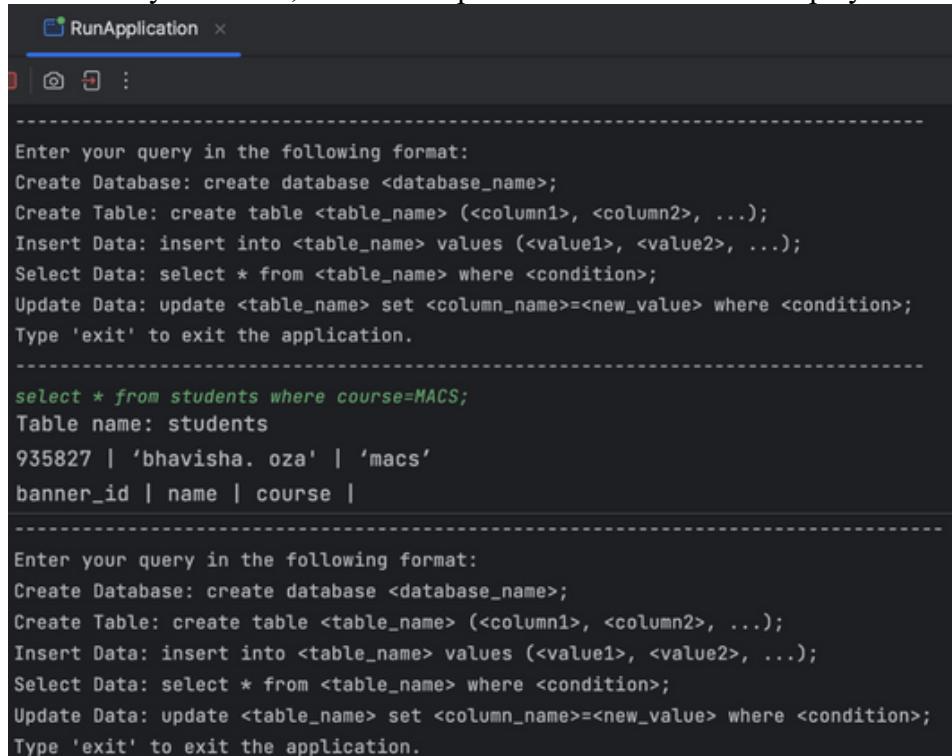
- After processing the “insert table” query, the application continues to display the menu options and waits for the next user input. Let's say the user wants to select values into a students table and enters the query “select * from students where banner_id = 1;”. The application checks if the query starts with "select" (case-insensitive). Since it matches, it

continues processing. Inside the `SelectTable.selectTable` method, the query is parsed and validated using regular expressions (regex). The regex pattern "`select\s+([a-zA-Z_,]+)\s+from\s+([a-zA-Z_]+)(?:\s+where\s+([a-zA-Z\d_]+\s*=\s*\d+));?`" is used to extract the columns, table name, and condition.

It iterates through the lines of the file and checks if the table name matches the requested table. If found, it splits the line using the `TABLE_DELIMITER` constant and extracts the table columns and rows.

Once a condition is provided, it evaluates the condition by comparing the column value with the given condition. If the condition is met, it either displays the entire row or selectively displays the requested columns based on the user's query.

If the query matches the pattern and the columns, table name, and condition are successfully extracted, the method proceeds to retrieve and display the table information.



The screenshot shows a terminal window titled "RunApplication". The terminal displays a help menu for SQL-like commands:

```
Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.
```

Then, a specific query is entered:

```
select * from students where course=MACS;
```

The application responds with the table structure and data:

```
Table name: students
935827 | 'bhavisha. oza' | 'macs'
banner_id | name | course |
```

Finally, another help menu is displayed:

```
Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.
```

Figure 31: Select query for the condition course = MACS.

- After processing the table data, the application continues to display the menu options and waits for the next user input. Let's say the user wants to update data in a table and enters the query "update students set name=Kairavi where course=MACS;". The application checks if the query starts with "update" (case-insensitive). Since it matches, it continues processing.

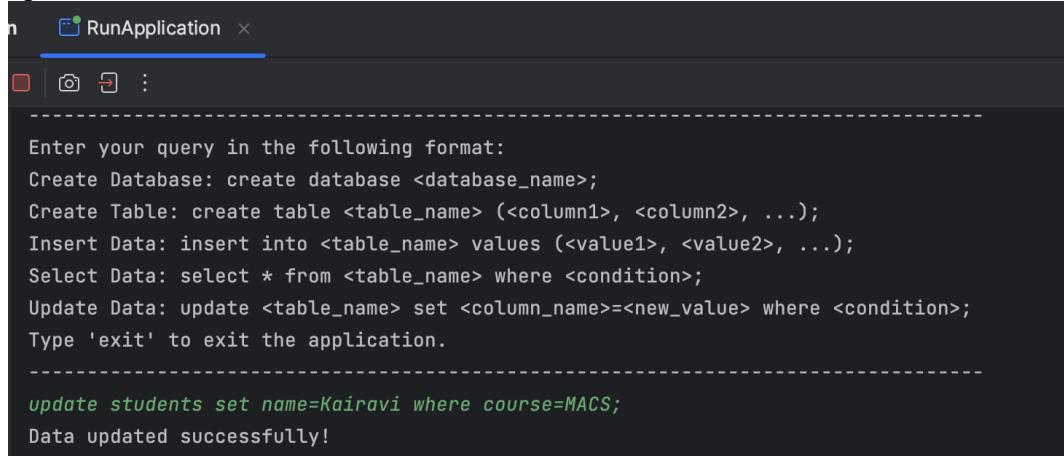
Inside the `UpdateTable.updateTable` method, the query is parsed and validated using regular expressions (regex). The regex pattern

"`update\s+(\w+)\s+set\s+(\w+)\s*=\s*(\w+)\s+where\s+(\w+)\s*=\s*(\w+)\s*`;" is used to extract the table name, set clause, and where clause.

Once the query matches the pattern and the table name, set clause, and where clause are successfully extracted, the method proceeds to update the data.

It checks if the row matches the condition specified in the where clause. If the condition is met, it updates the values based on the set clause.

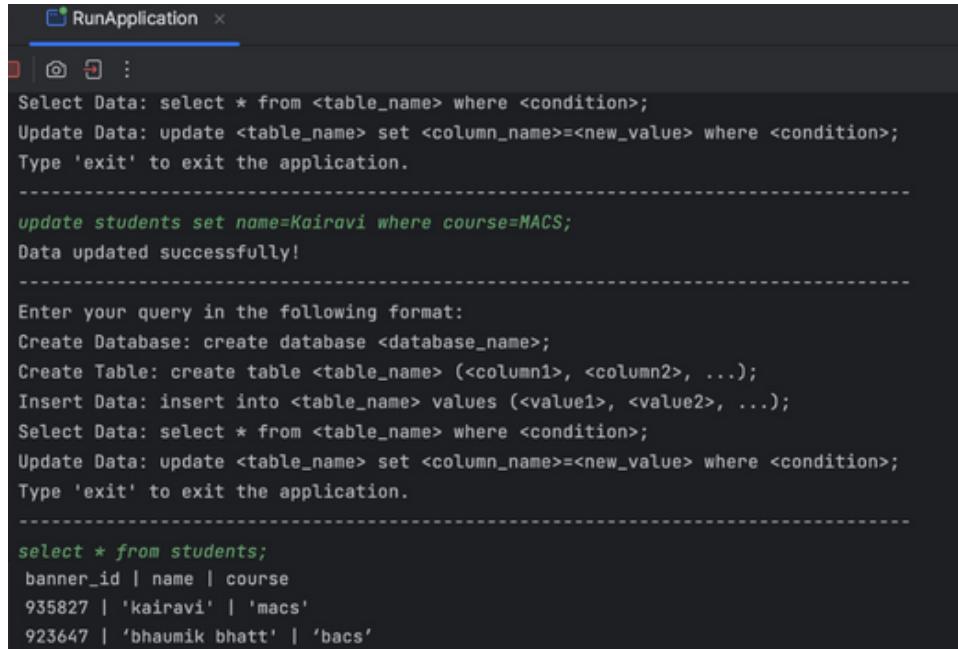
The application displays the message "Data updated successfully!" to indicate that the update was successful.



```
RunApplication x
-----
Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

-----
update students set name=Kairavi where course=MACS;
Data updated successfully!
```

Figure 31: Update query for the condition course = MACS.



```
RunApplication x
-----
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

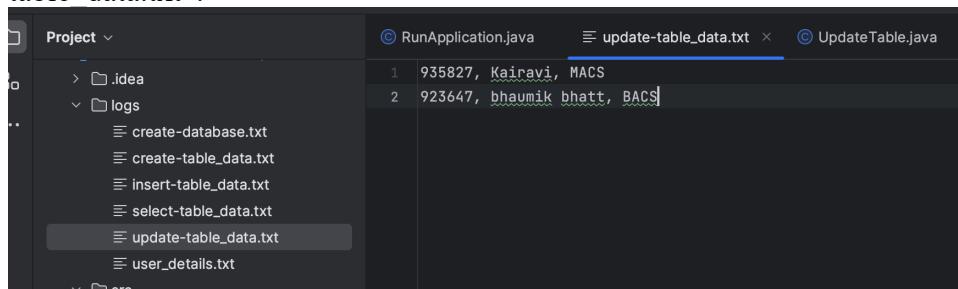
-----
update students set name=Kairavi where course=MACS;
Data updated successfully!

-----
Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

-----
select * from students;
banner_id | name | course
935827 | 'kairavi' | 'macs'
923647 | 'bhaumik bhatt' | 'bacs'
```

Figure 32: Select query for the updated value check name = Kairavi.

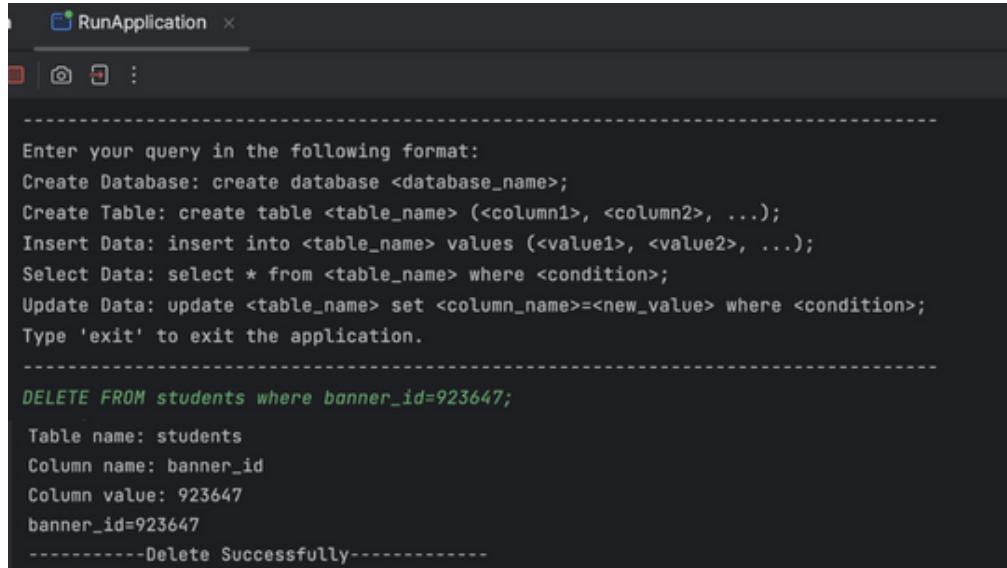
- If the query matches the pattern and the table name and values are successfully extracted, the method proceeds to update the inserted values in a custom file “update-table_data.txt”.



File	Content
Project	.idea logs create-database.txt create-table_data.txt insert-table_data.txt select-table_data.txt update-table_data.txt user_details.txt
RunApplication.java	935827, Kairavi, MACS
update-table_data.txt	935827, Kairavi, MACS
UpdateTable.java	

Figure 33: Update table data saved in text file.

- The application continues to display the menu options and waits for the next user input. Let's say the user wants to delete rows from a table and enters the query "DELETE FROM students where banner_id=923647;". The application checks if the query starts with "delete from" (case-insensitive). Since it matches, it continues processing.
- Inside the DeleteTable.deleteTable method, the query is parsed and validated using regular expressions (regex). The regex pattern "DELETE FROM ([a-zA-Z0-9_]+) WHERE (.+)" is used to extract the table name and the WHERE clause. If the query matches the pattern and the table name and WHERE clause is successfully extracted, the method proceeds to delete the rows. If the file replacement is successful, it displays the message "Data deleted successfully!" to indicate that the deletion was successful.



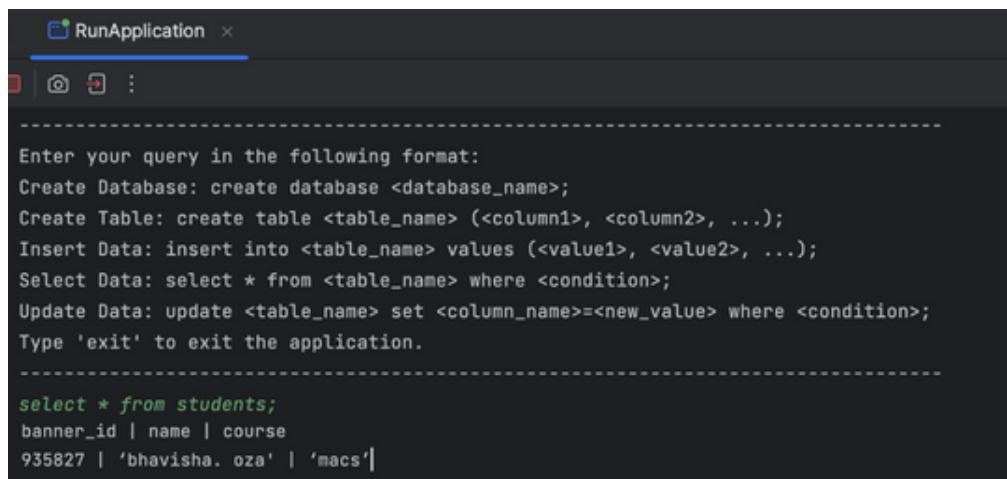
```

RunApplication x
-----[REDACTED]-----
Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

-----[REDACTED]-----
DELETE FROM students where banner_id=923647;
Table name: students
Column name: banner_id
Column value: 923647
banner_id=923647
-----Delete Successfully-----

```

Figure 34: Delete query for the condition banner_id = 923647.



```

RunApplication x
-----[REDACTED]-----
Enter your query in the following format:
Create Database: create database <database_name>;
Create Table: create table <table_name> (<column1>, <column2>, ...);
Insert Data: insert into <table_name> values (<value1>, <value2>, ...);
Select Data: select * from <table_name> where <condition>;
Update Data: update <table_name> set <column_name>=<new_value> where <condition>;
Type 'exit' to exit the application.

-----[REDACTED]-----
select * from students;
banner_id | name | course
935827 | 'bhavisha. oza' | 'macs'

```

Figure 35: Select query for the deleted value banner_id = 923647.

7. Implementation of Transaction:

The implemented code introduces a single transaction handling logic that follows the ACID (Atomicity, Consistency, Isolation, Durability) properties. The system identifies a transaction based on user input, such as "Begin Transaction", "End Transaction", etc.

To ensure the ACID properties, the processed queries are not immediately written to the custom-made database text file. Instead, they are stored in an intermediate data structure, such as Lists. On the other hand, if the user inputs "Rollback", the data structure is emptied, and no changes are made to the text file.

This approach allows for the execution of multiple queries within a single transaction, ensuring that the changes are consistent and durable. By separating the execution from the actual updates to the database file, the code provides a mechanism to handle transactions and maintain data integrity.

Working flow of Transaction:

- The user is prompted to enter a query in a specific format. If the user enters a query that starts a transaction ("begin transaction"), the transaction handler's handleTransaction method is called. The transaction handler displays a transaction menu and waits for the user to enter a query or command.

The user can enter one of the following commands within the transaction: "begin transaction", "end transaction", "commit", "rollback", or a regular query.

The screenshot shows a terminal window titled "RunApplication". The terminal interface includes standard icons for file operations (New, Open, Save, Copy, Paste, Find, Help) and a command history area. The main text area displays a transaction menu and two separate transaction sessions. The menu includes options: "Enter your query or command:", "Begin Transaction: begin transaction", "End Transaction: end transaction", "Commit: commit", and "Rollback: rollback". Two transaction sessions are shown:

- Session 1:** The user enters "begin transaction", and the terminal responds with "Transaction started."
- Session 2:** The user enters "begin transaction" again, and the terminal responds with "Transaction started." (Note: The second "begin transaction" entry appears to be a duplicate or a continuation of the first session.)

Figure 36: Begin Transaction query.

- If the user enters "begin transaction" when not in a transaction, an error message is displayed.

```
n RunApplication x
Enter your query or command:
Begin Transaction: begin transaction
End Transaction: end transaction
Commit: commit
Rollback: rollback
-----
Enter your query: create table visa_requirements (visa_date, age, payment);
You're not in a transaction. Use 'begin transaction' to start a new transaction.
-----
```

Figure 37: Error message for Begin Transaction when not in a transaction.

- If the user enters "begin transaction" while already in a transaction, an error message is displayed.

```
n RunApplication x
Rollback: rollback
-----
Enter your query: begin transaction
Transaction started.
-----
Enter your query or command:
Begin Transaction: begin transaction
End Transaction: end transaction
Commit: commit
Rollback: rollback
-----
Enter your query: begin transaction
A transaction is already in progress.
-----
Enter your query or command:
Begin Transaction: begin transaction
End Transaction: end transaction
Commit: commit
Rollback: rollback
```

Figure 38: Error message for Begin Transaction while already in a transaction.

- Once the user enters “end transaction”, the transaction handler checks if any queries were added to the transaction. If queries were added, it calls the transaction handler's executeQueries method to process and execute the queries.
The executeQueries method iterates over the queries and processes each query. In this example, it simply writes a response for each query to the data file.
If the queries are executed successfully, the transaction is considered committed. If the queries list is empty, the variable exitTransaction is set to true, indicating that the transaction should be exited.

```
Run RunApplication ×
Enter your query or command:
Begin Transaction: begin transaction
End Transaction: end transaction
Commit: commit
Rollback: rollback
-----
Enter your query: create table visa_requirements (visa_date, age, payment);
Query added to transaction.

Enter your query or command:
Begin Transaction: begin transaction
End Transaction: end transaction
Commit: commit
Rollback: rollback

Enter your query: insert into visa_requirements values (13 May 23, 27, CAD 250);
Query added to transaction.

Enter your query or command:
Begin Transaction: begin transaction
End Transaction: end transaction
Commit: commit
Rollback: rollback

Enter your query: end transaction
Queries executed successfully.
```

Figure 39: Transaction completion (Commit)

- If the user enters "rollback" while in a transaction, the transaction handler clears the queries list.

```
Run RunApplication ×
Enter your query or command:
Begin Transaction: begin transaction
End Transaction: end transaction
Commit: commit
Rollback: rollback
-----
Enter your query: insert into visa_requirements values (29 jun 23, 39, INR 7600);
Query added to transaction.

Enter your query or command:
Begin Transaction: begin transaction
End Transaction: end transaction
Commit: commit
Rollback: rollback

Enter your query: rollback
Transaction rolled back.
```

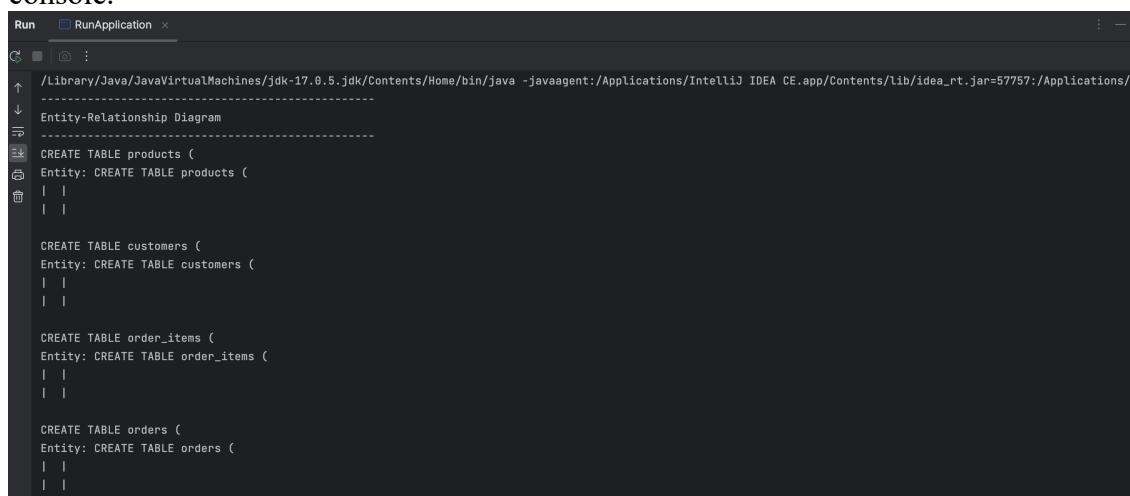
Figure 40: Transaction Rollback.

8. Implementation of ERD*:

The task is to implement an ERD (Entity-Relationship Diagram) generation class that can scan through the data and structure files of a database and generate a console-based representation of the ERD. Primary keys and foreign keys are denoted as "Pk" and "Fk", respectively. The ERD can be displayed as textual output on the screen using symbols like "|" and "-" to represent entity representations and relationships.

I have attempted to generate the ERD by having some custom designed text files. But it was not working correctly so have commented the code into the main method for the same. To accomplish this, the following steps has been taken:

- Created an ERDGenerator class that takes the paths of the data file and structure file as input.
- The generateERD() method is called to generate the Entity-Relationship Diagram (ERD) based on the data and structure files.
- Inside the generateERD() method:
 1. The method readTableColumns() is called to read the table columns from the structure file and store them in a map called tableColumns.
 2. The method readTableData() is called to read the table data from the data file and store it in a map called tableData.
 3. For each table, the entity representation is printed by retrieving the column names, primary keys, and data rows (if available) from the tableColumns and tableData maps. The entity representation is displayed in the console.
 4. The method readRelationships() is called to read the relationships between tables from the structure file and store them in a list called relationships.
 5. The entity representation and relationships are printed in the console. If there are relationships, they are displayed under the "Relationships:" section. If there are no relationships, the message "No relationships found." is displayed.
- The execution of the ERDGenerator class is completed, and the ERD is displayed in the console.



```
Run RunApplication ×
G | @ :
/Library/Java/JavaVirtualMachines/jdk-17.0.5.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Lib/idea_rt.jar=57757:/Applications/
Entity-Relationship Diagram
CREATE TABLE products (
Entity: CREATE TABLE products (
| |
| |
CREATE TABLE customers (
Entity: CREATE TABLE customers (
| |
| |
CREATE TABLE order_items (
Entity: CREATE TABLE order_items (
| |
| |
CREATE TABLE orders (
Entity: CREATE TABLE orders (
| |
| |
```

Figure 41: Attempt to create ERD.

Full code can be found in the given repository: <https://git.cs.dal.ca/boza/csci-5408/-tree/main/A2>.

References:

- [1] “Download intelliJ idea – the leading Java and Kotlin IDE,” *JetBrains* [Online]. Available: <https://www.jetbrains.com/idea/download/?section=mac> [Accessed: 01 July 2023].
- [2] “How to Write Doc Comments for the Javadoc Tool,” *Oracle Canada* [Online]. Available: <https://www.oracle.com/ca-en/technical-resources/articles/java/javadoc-tool.html#styleguide> [Accessed: 06 July 2023].
- [3] S. Desai, “All you Need to Know About Solid Principles in Java,” *edureka* [Online]. Available: <https://www.edureka.co/blog/solid-principles-in-java>, 2021. [Accessed: July 08, 2023].
- [4] “MD5 hash in java,” *GeeksforGeeks* [Online]. Available: <https://www.geeksforgeeks.org/md5-hash-in-java/> [Accessed: 05 July 2023].
- [5] “Build, test, and debug regex,” *regex101* [Online]. Available: <https://regex101.com/> [Accessed: 06 July 2023].