



Department of Information and Communication Technology
Faculty of Technology
University of Ruhuna

Advanced Database Management Systems

ICT 3213

Assignment 02

Group 13

Submitted to: Mr.P.H.P. Nuwan Laksiri

Submitted by: TG/2017/279- M.S.R.Kularathna

TG/2017/254 -R.N.H Vitharana

TG/2017/260 - S.Tharmiya

Table of Contents

01. INTRODUCTION TO PROBLEM	4
02. SOLUTION.....	5
03. PROPOSED ER/EER DIAGRAM	6
04.PROPOSED RELATIONAL MAPPING	7
05. TABLE STRUCTURE	8
06. ARCHITECTURE OF SOLUTION	9
07. TOOLS AND TECHNOLOGIES THAT USED.....	10
08. WHERE HOST YOUR BACKEND AND REASONS FOR THE SELECTION.....	17
09. SECURITY MEASURES THAT TAKEN TO PROTECT DB	17
10. BRIEF DESCRIPTION ABOUT DB ACCOUNTS/USERS AND THE REASONS FOR CREATING SUCH ACCOUNTS/USERS.....	18
11. CODE OF THE CONNECTION CLASS/ES OR FRAMEWORKS	18
12. PROBLEMS THAT YOU FACED DURING THE DEVELOPMENT OF THE BACKEND	19
13. SOLUTIONS/HOW YOU HAVE OVERCOME THE ABOVE IDENTIFIED PROBLEMS	19
14. NEW DATABASE TECHNOLOGIES/TRENDS THAT YOU HAVE USED TO DEVELOP THE BACKEND	19
15. CHANGES THAT DONE IN OUR BACKEND WHEN HOSTING IN A CLOUD ENVIRONMENT.....	19
16. WHAT ARE THE POSSIBILITIES FOR YOU TO REPLACE YOUR RELATIONAL DB BACKEND WITH A NON RELATIONAL DB TECHNOLOGY	20
17. Individual contribution to the backend development.....	21

Figure 1: <i>Proposed ER diagram</i>	6
Figure 2: <i>Proposed Relational Mapping</i>	8
Figure 3: <i>Canteen owner table invoice</i>	8
Figure 4: <i>Architecture Of Solution</i>	9
Figure 5: <i>Stored Procedures</i>	10
Figure 6: <i>Stored Procedures</i>	11
Figure 7: <i>Stored Procedures</i>	11
Figure 8: <i>Stored function</i>	12
Figure 9: <i>Stored function</i>	12
Figure 10: <i>Stored function</i>	13
Figure 11: <i>Event</i>	13
Figure 12: <i>Triggers</i>	14
Figure 13: <i>Triggers</i>	14
Figure 14: <i>Triggers</i>	15
Figure 15: <i>Triggers</i>	15
Figure 16: <i>Triggers</i>	16
Figure 17: <i>Caption</i>	16
Figure 18: <i>Application properties file</i>	20

01. INTRODUCTION TO PROBLEM

The manual system was suffering from a series of drawbacks. Since whole of the system was to be maintained with hands the process of keeping, maintaining and retrieving the information was very tedious and lengthy. The records were never used to be in a systematic order.

There used to be lots of difficulties in associating any particular transaction with a particular context. If any information was to be found it was required to go through the different registers, documents there would never exist anything like report generation. There would always be unnecessary consumption of time while entering records and retrieving records. One more problem was that it was very difficult to find errors while entering the records. Once the records were entered it was very difficult to update these records.

The reason behind it is that there is lot of information to be maintained and have to be kept in mind while running the business .For this reason we have provided features Present system is partially automated (computerized), actually existing system is quite laborious as one has to enter same information at three different places

02. SOLUTION

The purpose of University Canteen Management System is to automate the existing manual system by the help of computerized equipment and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. The required software and hardware are easily available and easy to work with. University Canteen Management System, as described above, can lead to error free, secure, reliable and fast management system.

It can assist the user to concentrate on their other activities rather to concentrate on the record keeping. Thus it will help organization in better utilization of resources. The organization can maintain computerized records without redundant entries.

That means that one need not be distracted by information that is not relevant, while being able to reach the information. The aim is to automate its existing manual system by the help of computerized equipment and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. Basically, the project describes how to manage for good performance and better services for the user.

The main objective of the Project on University Canteen Management System is to manage the details of University Canteen, student, student Meal, Meal Type, Canteen Staff. It manages all the information about University Canteen, Bill Payment, Canteen Staff, University Canteen. The project is totally built at administrative end and thus only the administrator is guaranteed the access. The purpose of the project is to build an application program to reduce the manual work for managing the University Canteen, student, Bill Payment, student Meal. It tracks all the details about the student Meal, Meal Type, Canteen Staff.

03. PROPOSED ER/EER DIAGRAM

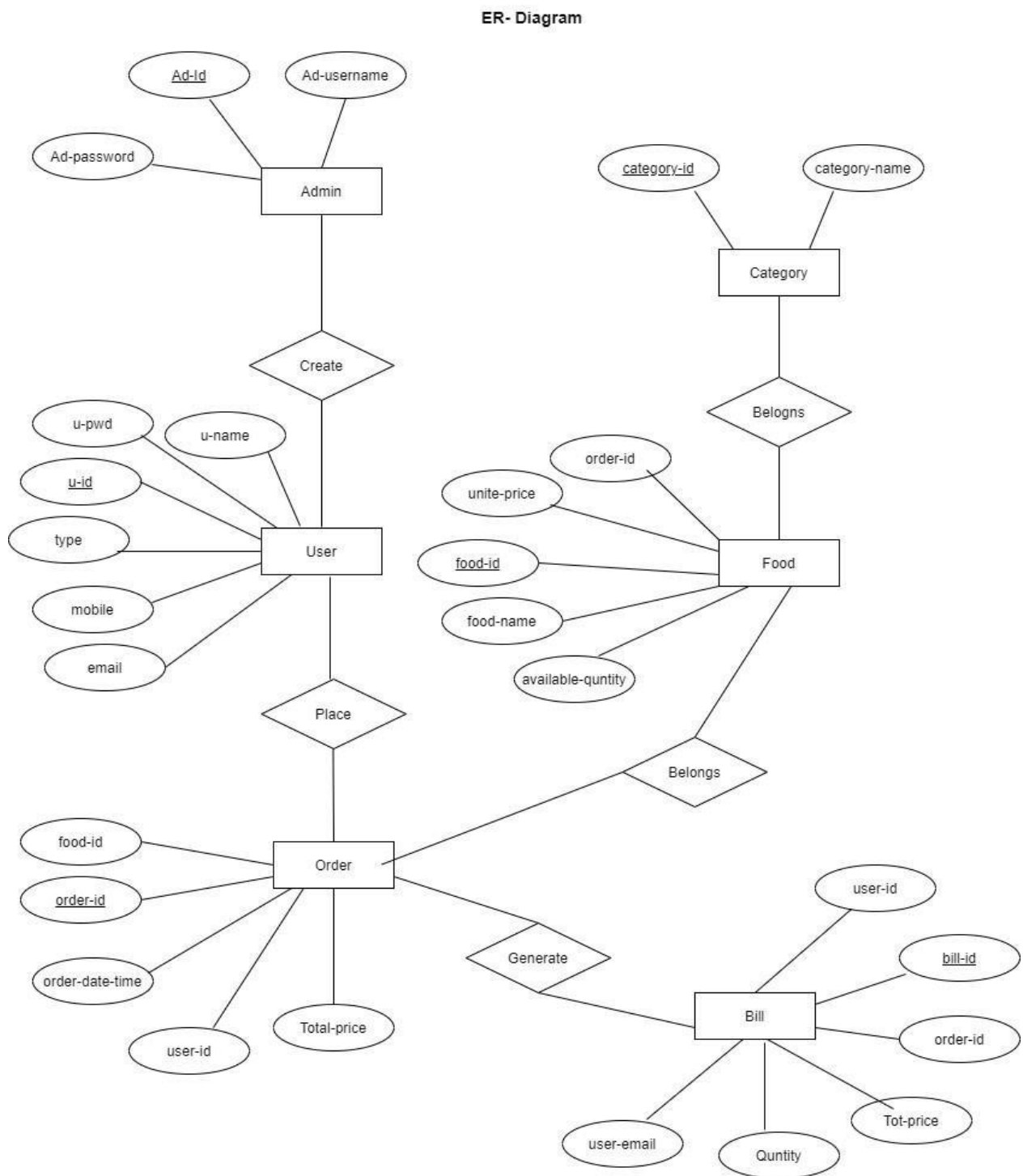


Figure 1: Proposed ER diagram

04.PROPOSED RELATIONAL MAPPING

Relational Mapping

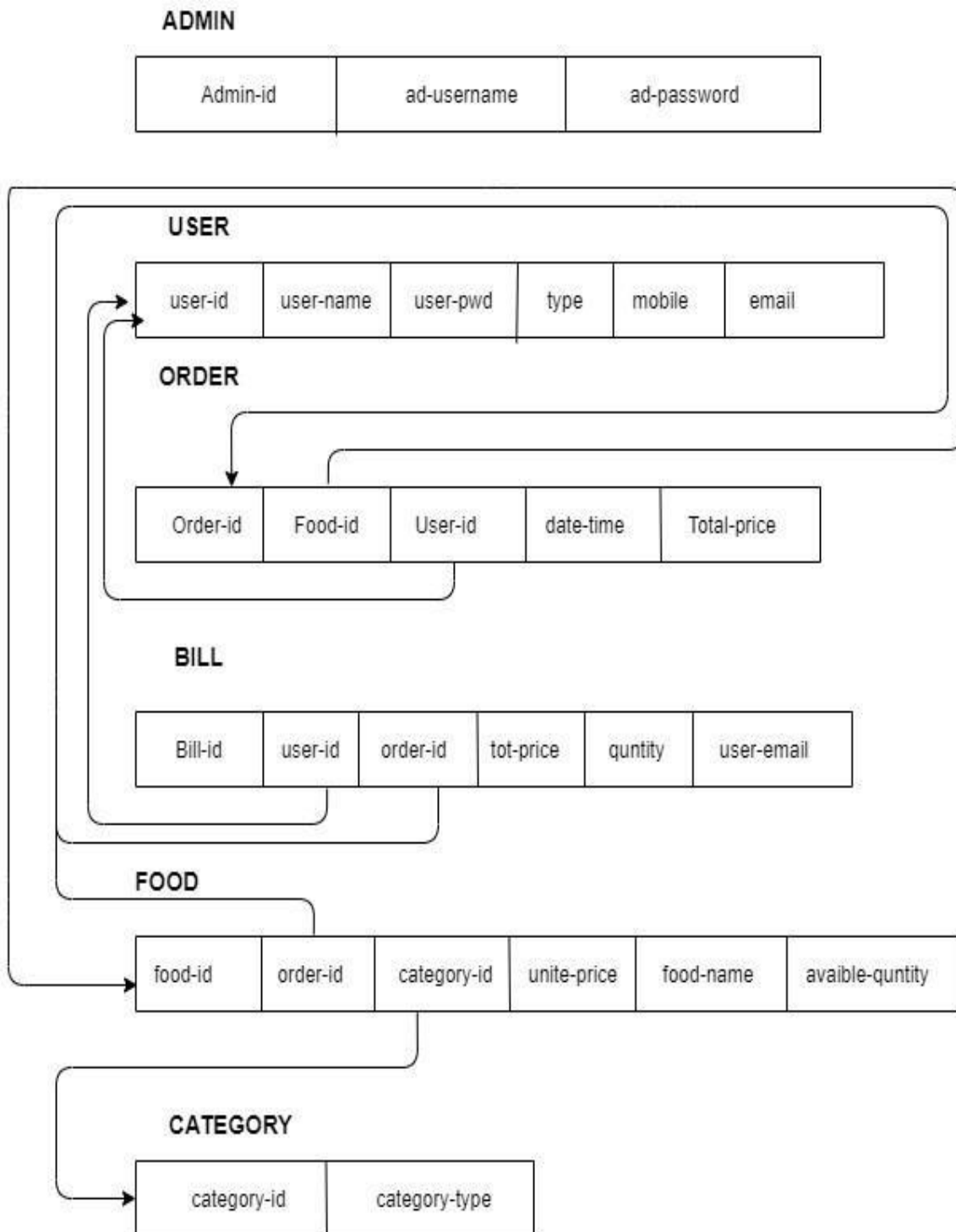


Figure 2: Proposed Relational Mapping

05. TABLE STRUCTURE

Table Stucture

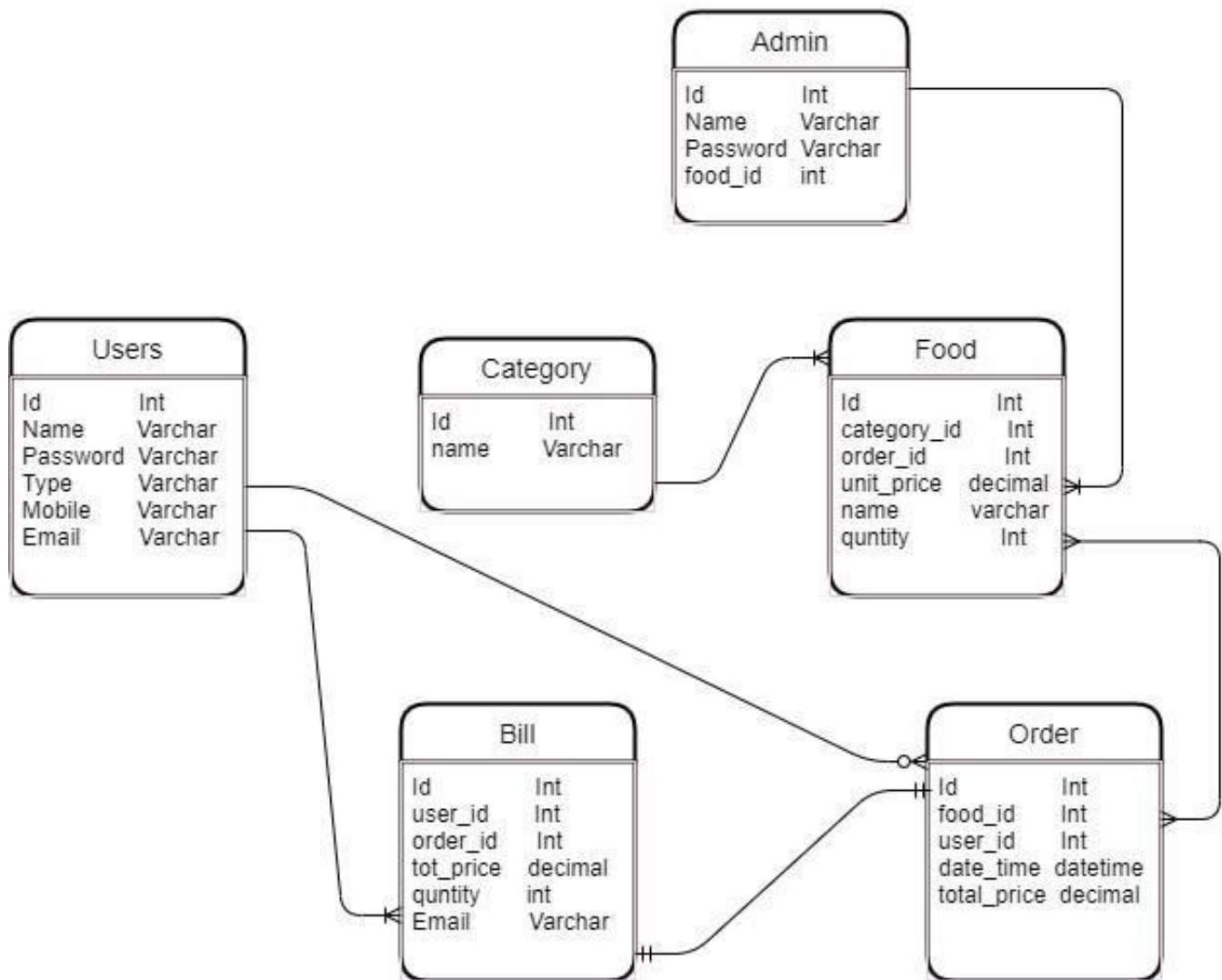


Figure 3: Canteen owner table invoice

06. ARCHITECTURE OF SOLUTION

Architecture of solution

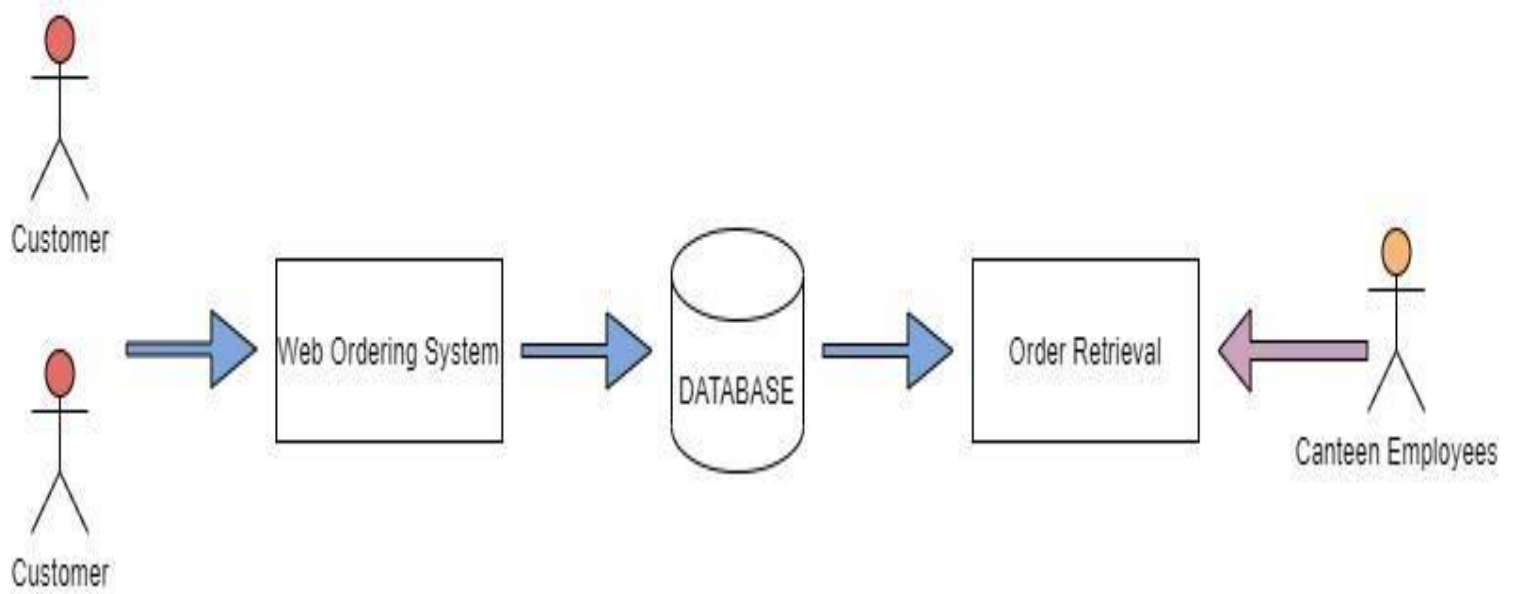


Figure 4: Architecture Of Solution

07. TOOLS AND TECHNOLOGIES THAT USED

- IntelliJ
- GitHub
- MySQL workbench

We have used Spring boot framework as our backend development. Interface of the system has created using HTML5. MySQL is used as database and phpMyAdmin act as the localhost. As the developing IDE, INELIJ is used to develop the system. Furthermore, EVENTS, TRIGGERS, STROED FNCTION and STROED PROCEDURES have used to get more efficient result from the system under database

STROED PROCEDURES

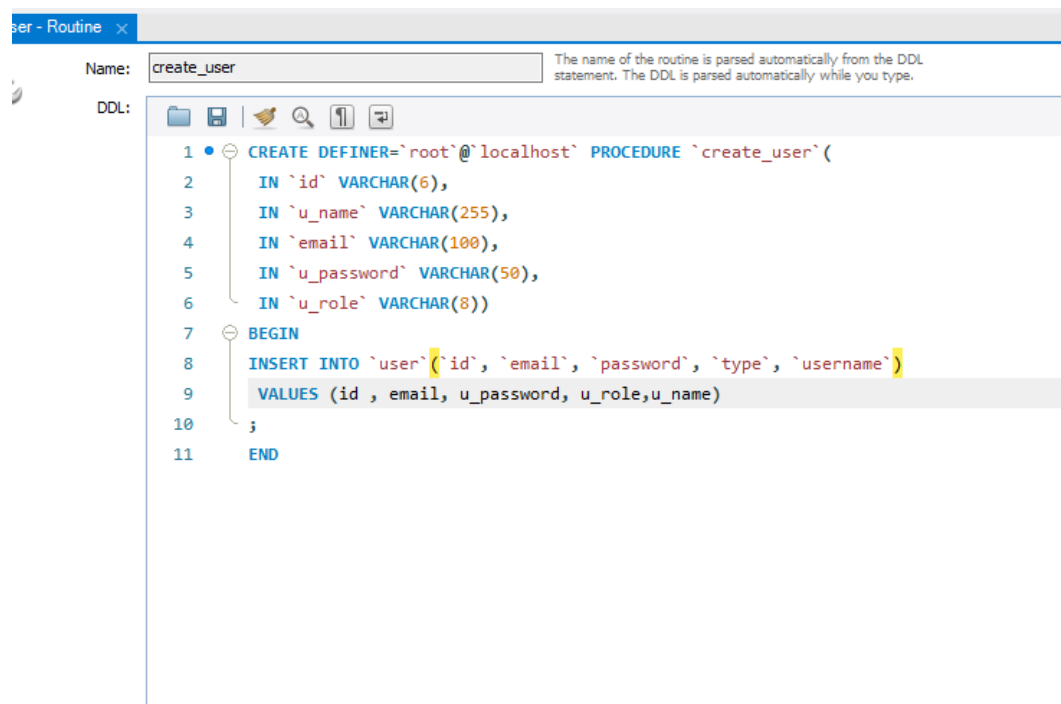


Figure 5: Stored Procedures

Name: The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:

```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `user_update_procedure`
2   (IN `u_id` VARCHAR(6),
3    IN `u_name` VARCHAR(255),
4    IN `u_email` VARCHAR(100),
5    IN `u_password` VARCHAR(50),
6    IN `u_role` VARCHAR(8))
7   BEGIN
8     UPDATE `user`
9     SET
10      `id` = u_id,
11      `username` = u_name,
12      `email` = u_email,
13      `password` = u_password,
14      `type` = u_role
15   WHERE `id` = u_id;
16   END
```

Figure 6: *Stored Procedures*

Help

routine GetItem - Routine **user_account_delete_procedure...** user_update_procedure - Routine view_user - Routine SQL

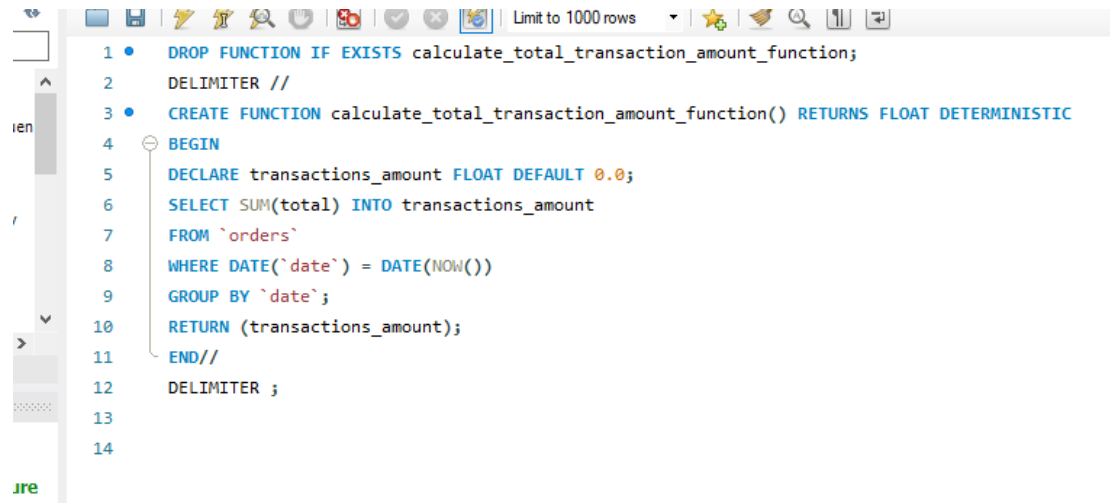
Name: The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:

```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `user_account_delete_procedure`
2   (IN `id` VARCHAR(6))
3   BEGIN
4     DELETE FROM `user` WHERE `id` = id;
5   END
```

Figure 7: *Stored Procedures*

STROED FNCTION

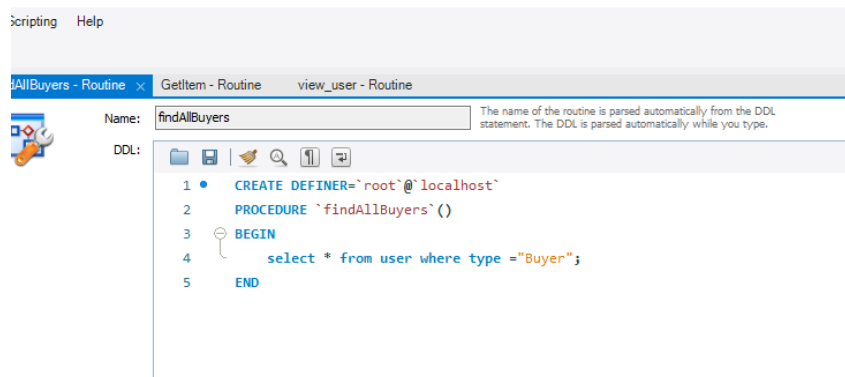


The screenshot shows a SQL IDE window with a toolbar at the top. The main text area contains the following SQL code:

```
1 • DROP FUNCTION IF EXISTS calculate_total_transaction_amount_function;
2 DELIMITER //
3 • CREATE FUNCTION calculate_total_transaction_amount_function() RETURNS FLOAT DETERMINISTIC
4 BEGIN
5 DECLARE transactions_amount FLOAT DEFAULT 0.0;
6 SELECT SUM(total) INTO transactions_amount
7 FROM `orders`
8 WHERE DATE(`date`) = DATE(NOW())
9 GROUP BY `date`;
10 RETURN (transactions_amount);
11 END//
12 DELIMITER ;
13
14
```

On the left side, there is a vertical toolbar with icons for file operations, and a sidebar with a tree view showing a database structure. The status bar at the bottom left indicates 'Limit to 1000 rows'.

Figure 8: *Stored function*

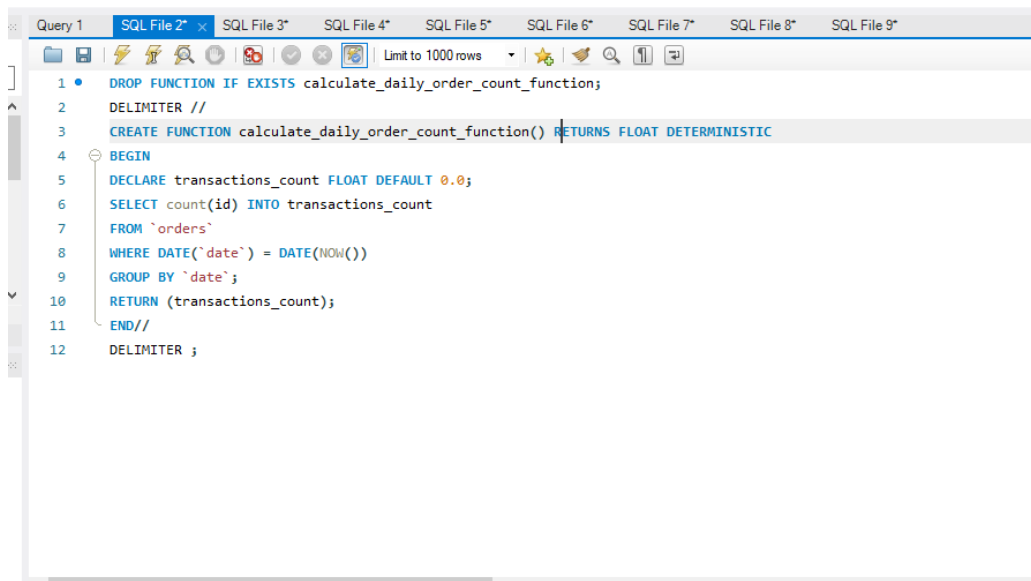


The screenshot shows a SQL IDE window with a toolbar at the top. The main text area contains the following SQL code:

```
1 • CREATE DEFINER=`root`@`localhost`
2 PROCEDURE `findAllBuyers`()
3 BEGIN
4 select * from user where type ="Buyer";
5 END
```

On the left side, there is a vertical toolbar with icons for file operations, and a sidebar with a tree view showing a database structure. The status bar at the bottom left indicates 'Limit to 1000 rows'.

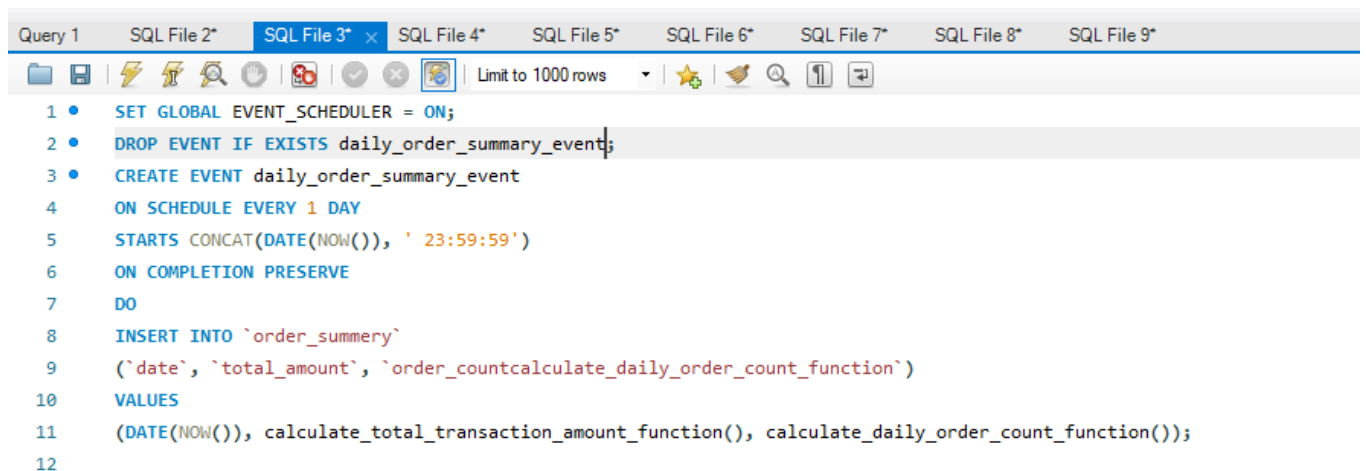
Figure 9: *Stored function*



```
1 DROP FUNCTION IF EXISTS calculate_daily_order_count_function;
2 DELIMITER //
3 CREATE FUNCTION calculate_daily_order_count_function() RETURNS FLOAT DETERMINISTIC
4 BEGIN
5 DECLARE transactions_count FLOAT DEFAULT 0.0;
6 SELECT count(id) INTO transactions_count
7 FROM `orders`
8 WHERE DATE(`date`) = DATE(NOW())
9 GROUP BY `date`;
10 RETURN (transactions_count);
11 END//
12 DELIMITER ;
```

Figure 10: Stored function

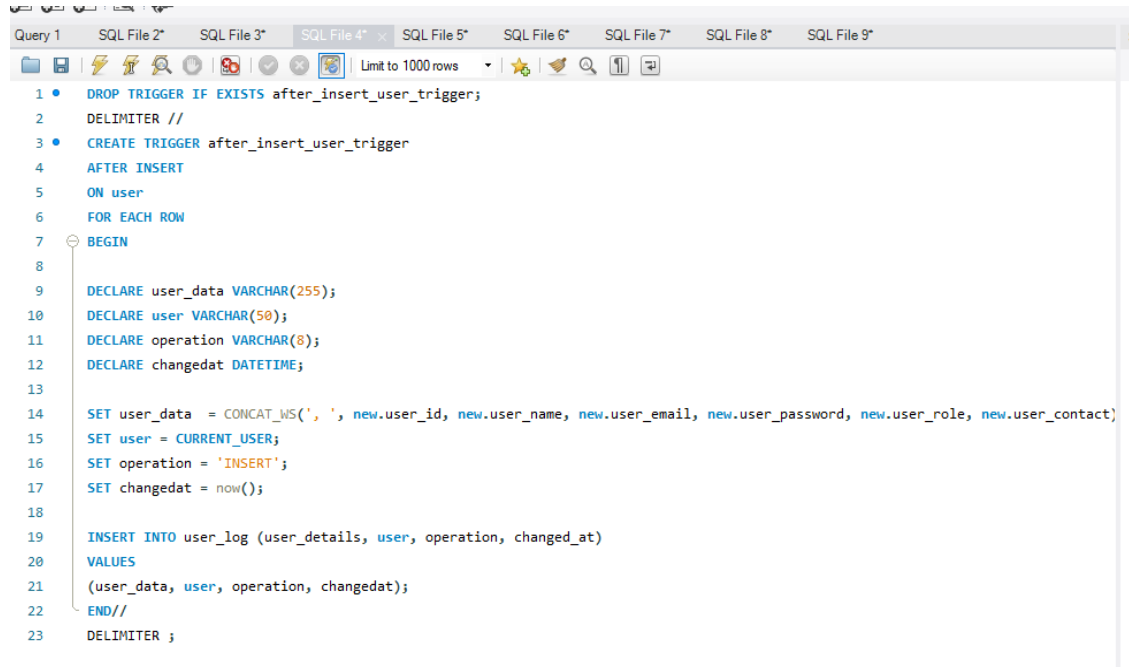
EVENT



```
1 SET GLOBAL EVENT_SCHEDULER = ON;
2 DROP EVENT IF EXISTS daily_order_summary_event;
3 CREATE EVENT daily_order_summary_event
4 ON SCHEDULE EVERY 1 DAY
5 STARTS CONCAT(DATE(NOW()), ' 23:59:59')
6 ON COMPLETION PRESERVE
7 DO
8 INSERT INTO `order_summary`
9 (`date`, `total_amount`, `order_count`, calculate_daily_order_count_function())
10 VALUES
11 (DATE(NOW()), calculate_total_transaction_amount_function(), calculate_daily_order_count_function());
12
```

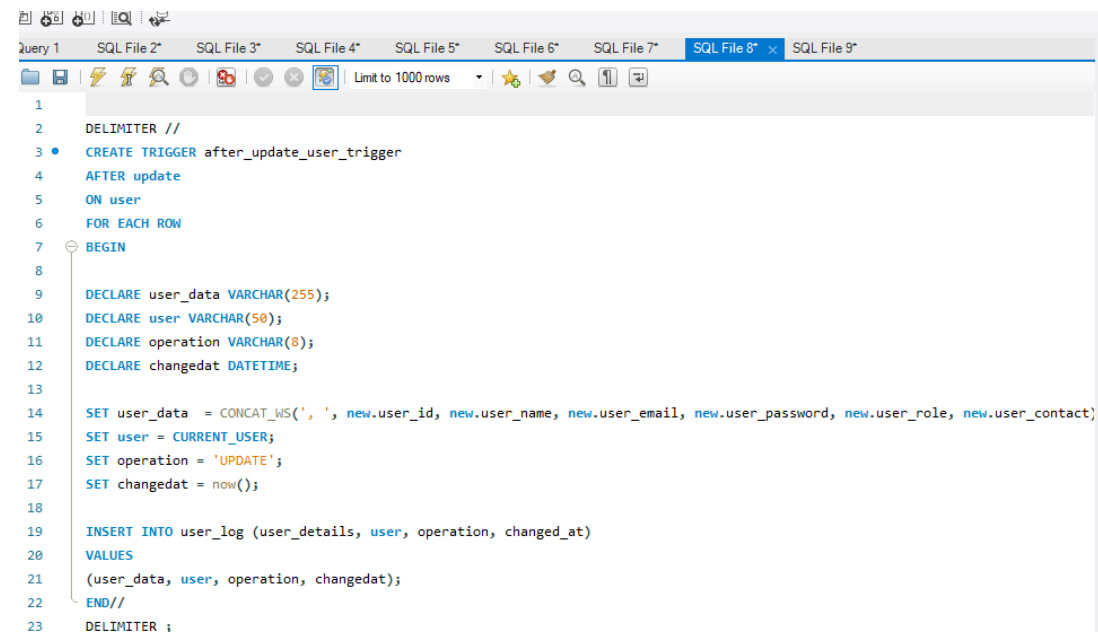
Figure 11: Event

TRIGGERS



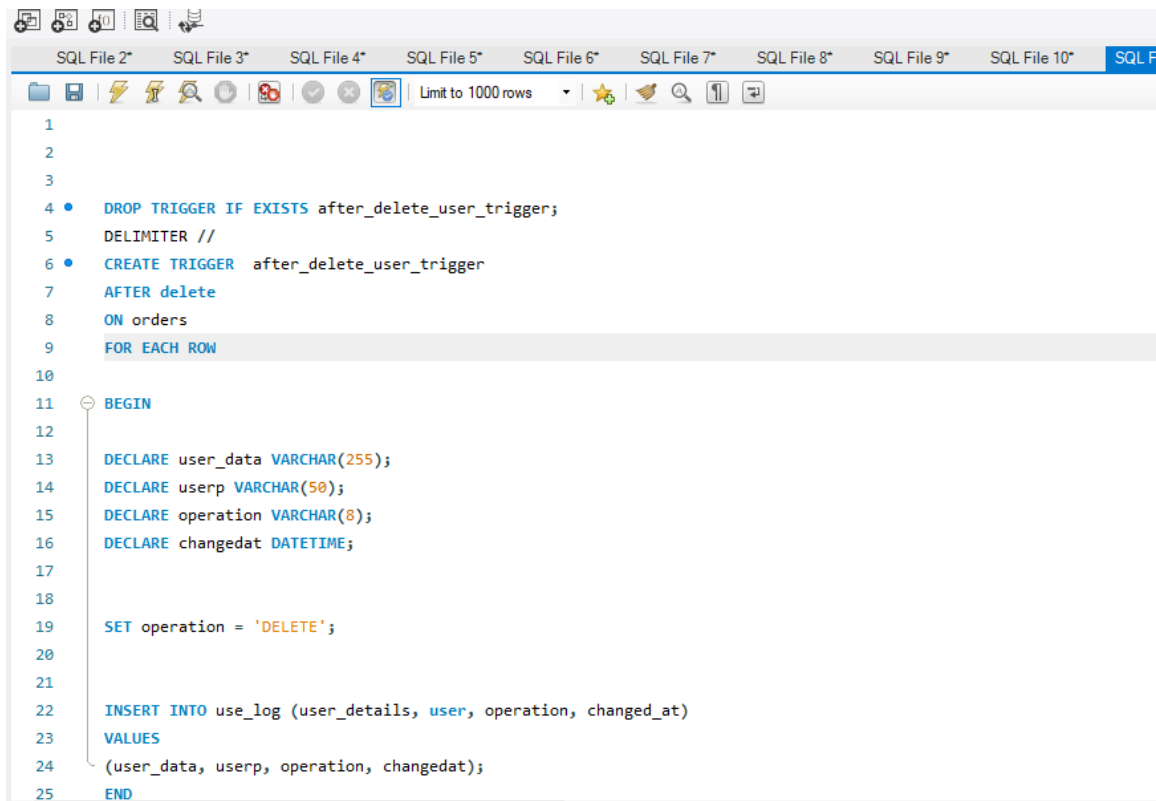
```
1 DROP TRIGGER IF EXISTS after_insert_user_trigger;
2 DELIMITER //
3 CREATE TRIGGER after_insert_user_trigger
4 AFTER INSERT
5 ON user
6 FOR EACH ROW
7 BEGIN
8
9     DECLARE user_data VARCHAR(255);
10    DECLARE user VARCHAR(50);
11    DECLARE operation VARCHAR(8);
12    DECLARE changedat DATETIME;
13
14    SET user_data = CONCAT_WS(' ', new.user_id, new.user_name, new.user_email, new.user_password, new.user_role, new.user_contact);
15    SET user = CURRENT_USER();
16    SET operation = 'INSERT';
17    SET changedat = now();
18
19    INSERT INTO user_log (user_details, user, operation, changed_at)
20    VALUES
21    (user_data, user, operation, changedat);
22 END//
23 DELIMITER ;
```

Figure 12: Triggers



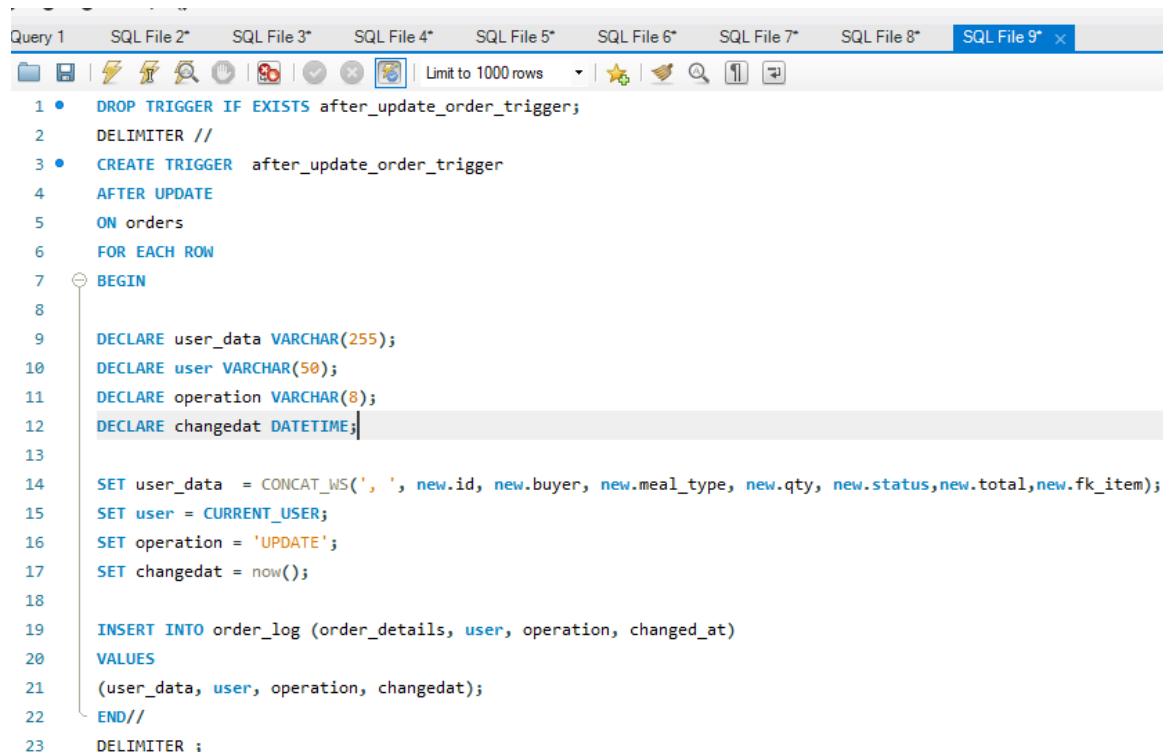
```
1
2 DELIMITER //
3 CREATE TRIGGER after_update_user_trigger
4 AFTER update
5 ON user
6 FOR EACH ROW
7 BEGIN
8
9     DECLARE user_data VARCHAR(255);
10    DECLARE user VARCHAR(50);
11    DECLARE operation VARCHAR(8);
12    DECLARE changedat DATETIME;
13
14    SET user_data = CONCAT_WS(' ', new.user_id, new.user_name, new.user_email, new.user_password, new.user_role, new.user_contact);
15    SET user = CURRENT_USER();
16    SET operation = 'UPDATE';
17    SET changedat = now();
18
19    INSERT INTO user_log (user_details, user, operation, changed_at)
20    VALUES
21    (user_data, user, operation, changedat);
22 END//
23 DELIMITER ;
```

Figure 13: Triggers



```
1
2
3
4 • DROP TRIGGER IF EXISTS after_delete_user_trigger;
5 DELIMITER //
6 • CREATE TRIGGER after_delete_user_trigger
7 AFTER delete
8 ON orders
9 FOR EACH ROW
10
11 BEGIN
12
13 DECLARE user_data VARCHAR(255);
14 DECLARE userp VARCHAR(50);
15 DECLARE operation VARCHAR(8);
16 DECLARE changedat DATETIME;
17
18
19 SET operation = 'DELETE';
20
21
22 INSERT INTO use_log (user_details, user, operation, changed_at)
23 VALUES
24 (user_data, userp, operation, changedat);
25 END
```

Figure 14: Triggers



```
Query 1 SQL File 2* SQL File 3* SQL File 4* SQL File 5* SQL File 6* SQL File 7* SQL File 8* SQL File 9* x
1 • DROP TRIGGER IF EXISTS after_update_order_trigger;
2 DELIMITER //
3 • CREATE TRIGGER after_update_order_trigger
4 AFTER UPDATE
5 ON orders
6 FOR EACH ROW
7 BEGIN
8
9 DECLARE user_data VARCHAR(255);
10 DECLARE user VARCHAR(50);
11 DECLARE operation VARCHAR(8);
12 DECLARE changedat DATETIME;
13
14 SET user_data = CONCAT_WS(' ', new.id, new.buyer, new.meal_type, new.qty, new.status, new.total, new.fk_item);
15 SET user = CURRENT_USER;
16 SET operation = 'UPDATE';
17 SET changedat = now();
18
19 INSERT INTO order_log (order_details, user, operation, changed_at)
20 VALUES
21 (user_data, user, operation, changedat);
22 END//
23 DELIMITER ;
```

Figure 15: Triggers

```

1
2 • DROP TRIGGER IF EXISTS after_insert_order_trigger;
3 DELIMITER //
4 • CREATE TRIGGER after_insert_order_trigger
5 AFTER INSERT
6 ON orders
7 FOR EACH ROW
8 BEGIN
9
10 DECLARE user_data VARCHAR(255);
11 DECLARE user VARCHAR(50);
12 DECLARE operation VARCHAR(8);
13 DECLARE changedat DATETIME;
14
15 SET user_data = CONCAT_WS(' ', new.id, new.buyer, new.meal_type, new.qty, new.status, new.total, new.fk_item);
16 SET user = CURRENT_USER;
17 SET operation = 'INSERT';
18 SET changedat = now();
19
20 INSERT INTO order_log (order_details, user, operation, changed_at)
21 VALUES
22 (user_data, user, operation, changedat);
23 END

```

Figure 16: Triggers

VIEWS

outline GetItem - Routine view_user - Routine **order_details - View** x

Name: The name of the view is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:

```

1 • CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = `root`@`localhost`
4     SQL SECURITY DEFINER
5     VIEW `fot-canteen`.`order_details` AS
6     SELECT
7         `fot-canteen`.`orders`.`buyer` AS `buyer`,
8         `fot-canteen`.`orders`.`total` AS `total`
9     FROM
10        `fot-canteen`.`orders`
11     ORDER BY `fot-canteen`.`orders`.`total` DESC

```

Figure 17: Caption

08. WHERE HOST YOUR BACKEND AND REASONS FOR THE SELECTION

We use AWS (Amazon Web Service) to host the backend.

Reasons

Broad platform support- With AWS, you can use whatever CMS you like, including WordPress, Drupal, Joomla, and more. AWS also supports and provides SDKs for popular platforms like Java, Ruby, PHP, Node.js, and .Net.

Datacenters worldwide - our customers can be anywhere in the world. With AWS you can have a datacenter or CDN hosting your website in any geography you choose with just a few mouse clicks.

Scalable from day one- Website traffic can fluctuate a lot. From quiet times in the middle of the night, to campaign driven, social media sharing traffic spikes, AWS infrastructure that can grow and shrink to meet your needs.

09. SECURITY MEASURES THAT TAKEN TO PROTECT DB

- Secure database user access - should aim for the least number of people possible to have access to the database. Administrators should have only the bare minimum privileges they need to do their job, and only during periods while they need access. For smaller organizations, this may not be practical, but at the very least permissions should be managed using groups or roles rather than granted directly.
- Encrypt data and backups - should also regularly backup your database and ensure that any backups are encrypted and stored separately from the decryption keys.
- Separate database servers and web servers- keeping our database server in a secure, locked environment with access controls in place to keep unauthorized people out

10. BRIEF DESCRIPTION ABOUT DB ACCOUNTS/USERS AND THE REASONS FOR CREATING SUCH ACCOUNTS/USERS

In the hosting environment, they also have MySQL environment in their computers they use shared hosting then they have other hosted websites also then we need to create our hosted database to relevant DB name and the separate user for access to the DB. Particular database user only can access the relevant database it protects the database from unauthorized access, and it reduce the risks. It protects our database and restricts other users from access.

11. CODE OF THE CONNECTION CLASS/ES OR FRAMEWORKS

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver spring.datasource.url =  
jdbc:mysql://localhost:3306/canteen_ms?useUnicode=true&useJDBCCompliantTimezoneS  
hift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
```

```
spring.datasource.username = root
```

```
spring.datasource.password =
```

```
#server.port=9090
```

```
spring.jpa.show-sql = true
```

```
spring.jpa.hibernate.ddl-auto = update
```

12. PROBLEMS THAT YOU FACED DURING THE DEVELOPMENT OF THE BACKEND .

- When we are doing the GitHub committing display so many errors.
- A couple of bugs and error were encountered during the implementation process

13. SOLUTIONS/HOW YOU HAVE OVERCOME THE ABOVE IDENTIFIED PROBLEMS

- The problems encountered during the implementation process were resolved with the help of stack overflow and cross checking the codes properly.

14. NEW DATABASE TECHNOLOGIES/TRENDS THAT YOU HAVE USED TO DEVELOP THE BACKEND

- We create database in our framework, so no need to share the database and we create the tables as entities we didn't already create the tables, while running the system it creates the tables.

15. CHANGES THAT DONE IN OUR BACKEND WHEN HOSTING IN A CLOUD ENVIRONMENT.

Change the properties file.- application.properties file

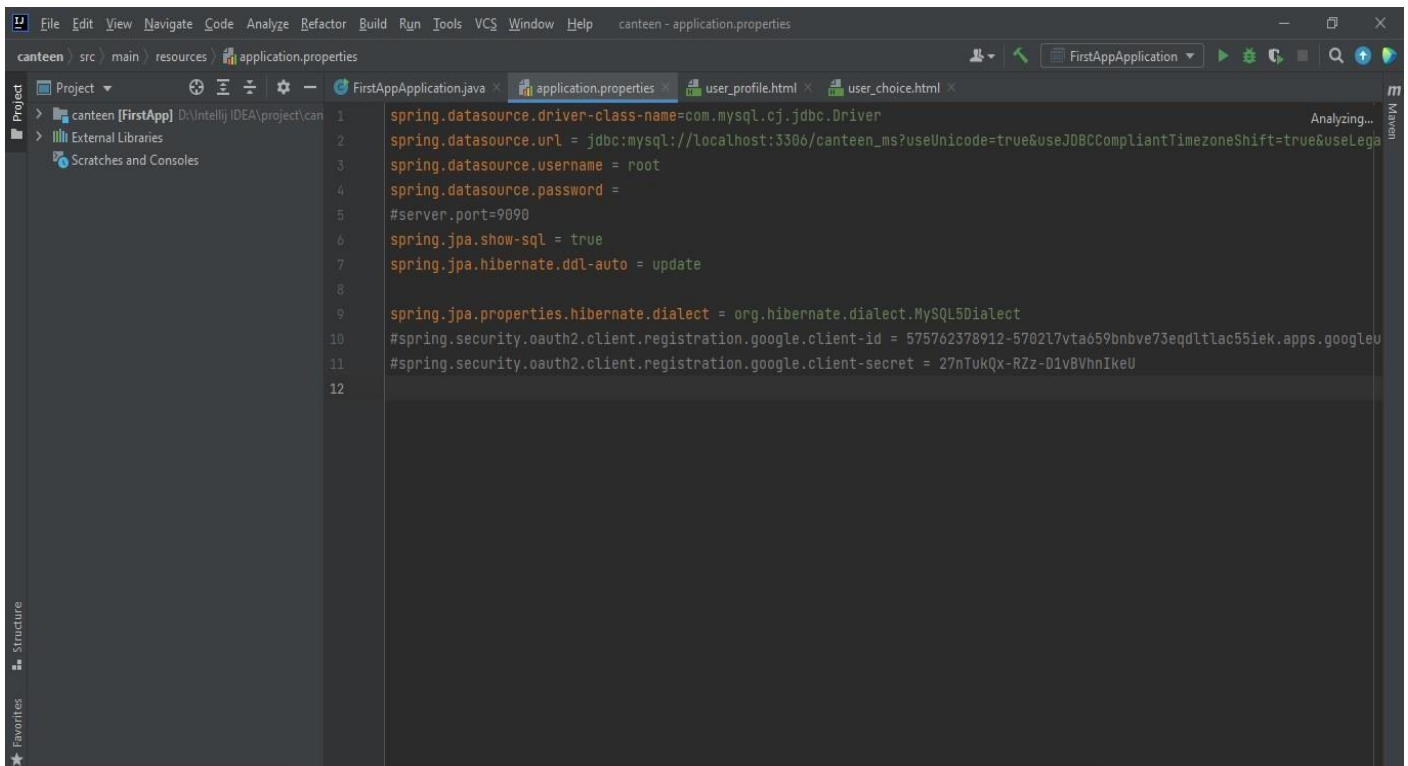


Figure 18: Application properties file

16. WHAT ARE THE POSSIBILITIES FOR YOU TO REPLACE YOUR RELATIONAL DB BACKEND WITH A NON RELATIONAL DB TECHNOLOGY

A relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as invented by E. F. Codd, of IBM's San Jose Research Laboratory. Many popular databases currently in use are based on the relational database model.

RDBMSs have become a predominant choice for the storage of information in new databases used for financial records, manufacturing and logistical information, personnel data, and much more since the 1980s. Relational databases have often replaced legacy hierarchical databases and network databases because they are easier to understand and use. However, relational databases have been challenged by object databases, which were introduced in an attempt to address the object-relational impedance mismatch in relational database, and XML databases.

Non-relational/NoSQL databases provide following advantages,

- Handle large volumes of data at high speed with a scale-out architecture
- Store unstructured, semi-structured, or structured data
- Enable easy updates to schemas and fields
- Be developer-friendly
- Take full advantage of the cloud to deliver zero downtime

17. Individual contribution to the backend development

Group Members	Individual Contribution
TG/2017/279- M.S.R.Kularathna	Admin
TG/2017/254 -R.N.H.Vitharana	Login, Register ,user
TG/2017/260 - S.Tharmiya	Inventory

