**Azerbaijan State Oil and Industry University**

**Faculty:** Information Technology and Management

**Department:** Computer Engineering

**Group:** 682.20E

**Specialty:** Information Technologies

**Subject:** Database Management System

**Topic:** Database System

**COURSE WORK**

**Students:** Aphar Magarramova, Gulnar Mammadli, Nurgun Qanbarli, Suzan Alizada, Senan Huseynov, Natig Qasimli

**Teacher:** ass. lec. Kishiyeva N.S

**Department Director:** Rahimova N.A

BAKI – 2022

**Contents**

**Chapter 1. Food Service Database**

* 1. Introduction………………………………………..…............…...........3
  2. Research...................................................................................................3
  3. Business model…………………………………………………………5

**Chapter** **2. Creating tables in the Food Service database**

* 1. ER diagram……………………………………………..…....................7
  2. Entity description.....................................................................................9
  3. Alter, drop columns……………………………..….............................18
  4. SQL constraints…………….......………………..…............................19

**Chapter 3. SQL queries. Using SQL to Retrieve Information from tables.**

* 1. Simple SQL queries........……….......………………..…......................20
  2. Joining tables………….......………………..…....................................21
  3. SQL Procedure......................................................................................23
  4. SQL Trigger..........................................................................................27
  5. SQL Views………….......………………..….......................................29
  6. SQL Indexer..........................................................................................31

**Chapter 4. Conclusion**………….......……………..…..........................................32

**Food Service Database**

1. **Introduction**

The purpose of this project is to develop a Fast-food Restaurant management system. It is a system that will assist managers and administrators in managing restaurants effectively and a system that enabled customer to place their food order online at any time from any place. The reason to develop the system is to Reduce the workload in the present system and reduce time wasted in data processing. It provides a user-friendly database for displaying food menu and effective advertising of Paramount cuisine services products to the customers with cheaper cost. Food order management allows restaurants to provide takeout orders or delivery to patrons. Because of COVID-19, food order management has become even more important. With dining rooms closed, takeout and delivery services have become lifelines for many restaurants. The system was designed and implemented using MySQL.

* 1. **Research**

**COMPARE BETWEEN OTHER COMPANIES**

**Other company (Wolt)**

Order using the wolt.com website as well as the application for iOS and Android possible. Then choose between home delivery, takeaway, and dining at the table you can. Wolt also offered special discount coupons to Azerbaijanis. Wolt courier partners deliver food by bicycle, motorcycle, and car, depending on the area. The minimum order volume is 6 AZN, and the standard delivery price is 1.5 AZN. Standard prices are set according to the delivery distance. Loans and tokens are automatically applied. Credits - is money that you have in your Wolt account and can use to order. A token means a free standard delivery (up to 1.5 km).

Each customer has a unique promo code. You can find this code in your account in the application, "Earn loans" section. You can share this code with your friends and give them 2 AZN credits for each of the first 3 orders. In this way you can make a very good profit. You can change your promo code once and make it more personal. In order to eliminate delays in delivery, the approximate delivery time is adjusted. A support service is available to troubleshoot errors. Customers can see where the courier is on the map. You can use e-mail, Facebook, Instagram accounts and live chat in the Wolt application.

**Our company (Food Lab)**

We also have website and app for IOS and Android, but our differences is everyone has own account in our website and there are many functions our apps and if they share our app with their friends, they get discount. Also, in our app has live tracking our customer can use it. We have also discount coupons for customer and us promo code for every customer they use it and can get discount. Also, our discount coupon amount is huge than Wolt. Delivery is by bicycle, car, and motorcycle. Our difference is that the delivery is as fast as possible, the orders are delivered accurately, on time and the food maintains its temperature. Reasonable prices are one of our main advantages. We offer quality food at the most affordable prices possible. The minimum order volume is 4 AZN. The standard delivery amount is 1.20 AZN. Delivery is free for distances less than 1km. Delivery price is determined according to the distance. The loans we arrange for customers are kept in their accounts. We lend the best amount possible. Our customers can also benefit from tokens. Delivery is free for distances less than 1km. Each customer has their own personal promo code. Customers can give them credit by sharing their promo codes with friends. They can use this promo code only once and get a loan of 3.50 AZN for an order of 50 AZN or more. We make it incredibly easy to discover and get great food, groceries, and anything you might need delivered in your city. Whatever you’re in the mood for, we’ll make sure it gets delivered to you. Our delivery services will deliver orders 15 minutes ahead of schedule. There are also small vehicles such as motorcycles and bicycles to avoid delays. If there is a delay in the arrival of any order more than 15 minute , our customer can take advantage of our company with a 10% discount. Also, incorrect orders are replaced free of charge. You’ll be kept in the loop on the status of the delivery in real-time, with a minute-by-minute countdown. If staring at a clock isn’t your thing, don’t worry – we'll also send you notifications, so you know when your order is about to arrive. The additional, you can follow server on the map easily at your home. Getting your food and goods should be as simple as possible. That’s why you can pay conveniently and securely with your credit card or Google Pay. Whatever works for you. Users can use official social website and live chat in mobile and web application.

* 1. **Business Model**

A business model is a framework for how a company will create value. Business model of the Food Service Management System distills the potential of a business down to its essence.

**Text

Description automatically generated with medium confidence**

Key Partners of our company are Restaurant and Delivery Service. Customer can use Restaurant's menu for ordering and Delivery Service provide ordering.

Key Activities include Taking Online Orders , Partnership with Delivery Service, Managing Logistics to Process Orders, Creating & Managing Technological Infrastructure, Managing Delivery and Payment process. These processes play an important role of creation and providing of company.

There are special Value Propositions for both drivers which provide ordering service and customers. Customer can order meals on the official social web site pages of company or Applications of company. There is 24/7 customer service. Customers can browse all menus of restaurant using our application. By using this way, they can define better pricing and efficiency. On the other hand, Drivers can provide exceptional customer base, Easy, Cheap, and a fast way of Delivering Orders. They can work with flexible time slot. That's why delays are rare.

Customer Relationship includes social media, 24/7 Customer Support. Our company has official social website pages and also Mobile app. This Mobile App is considered for both IOS and Android operation system.

Customer Segments are divided into 2 parts: User and Restaurant. Our company is considered for all these users that people who work a lot and don't have time for shopping, people who don't want to cook or want to eat from outside for example elderly people. Also, our company is good for restaurants which need many customers.

Technological set up & running cost include cost of creation application and websites. Salaries to permanent employees include wages of all employees. Some Marketing & Branding programmers include some sponsors of company.

1. **Creating tables in the Food Service database**

Tables are used to store data in the database. Creating a basic table involves naming the table and defining its columns and each column's data type. The SQL CREATE TABLE statement is used to create a new table. The SQL Server INSERT INTO statement is used to add new rows of data to a table in the database.

1. **ER diagram**

The ER (Entity Relationship) diagram stands for the model of Food Service Management Entity. The entity-relationship diagram of Food Service Management shows all the visual instrument of database tables and relation between restaurants, workers, customers, orders and etc. It used structure data and to define relationships between structured data groups of Food Service Management System functionalities. There are common features of Food Service Management System Database Design such as Managing Customers, Workers, Orders, Deliveries, and Payment’s information.

**Diagram, timeline

Description automatically generated**

* 1. **Entity description:**

1. Opening date:

**Create** **table** **OpeningDate** (

opening\_id **int** **primary** **key** **auto\_increment** **not** **null**,

days **enum**('weekdays', 'weekend'),

opening\_time **datetime**,

closing\_time **datetime**

);

* opening\_id – this is a unique ID for an opening date
* days – this is the name of a days
* opening\_time – this is the opening time of the restaurant
* closing\_date – this is the closing time of the restaurant

2)Restaurant:

**create** **table** **Restaurant**(

resId **int** **primary** **key** **auto\_increment** **not** **null**,

resName **varchar**(**150**),

res\_address **varchar**(**250**),

res\_phone **varchar**(**20**),

res\_mail **varchar**(**150**),

res\_image **binary**,

res\_isActive **bool**,

opening\_id **int**,

**constraint** fk\_opening

**foreign** **key** (opening\_id)

**references** **OpeningDate**(opening\_id)

);

* resId – this is a unique ID for a restaurant
* resName – this is a name of the restaurant
* res\_address – this is an address of the restaurant
* res\_phone – this is a phone of the restaurant
* res\_mail – this is a mail of the restaurant
* res\_image – this is an image of the restaurant
* res\_isActive – this is shows that restaurant is open or not.
* opening\_id – this is ID of the opening date which referring to the opening\_id column in the Opening Date table.

3) Workers

**Create** **table** **Workers** (

workerId **int** **primary** **key** **auto\_increment** **not** **null**,

FirstName **varchar**(**50**),

LastName **varchar**(**50**),

Address **varchar**(**60**),

Phone **varchar**(**50**),

Email **varchar**(**100**),

JobTitle **varchar**(**100**),

Salary **int**,

ReportsTo **int**,

resId **int**,

**constraint** fk\_res

**foreign** **key**(resId)

**references** **Restaurant**(resId)

);

* workerId – this is a unique ID for worker
* FirstName – this is a First name of the worker
* LastName – this is a Last name of the worker
* Address – this is an address of the worker
* Phone – this is a phone of the worker
* Email – this is a mail of the worker
* JobTitle – this shows job of the worker
* Salary – this is the salary of the worker
* reportsTo – this shows the manager of the worker
* resId - this is ID of the restaurant which referring to the resId column in the Restaurant table.

4) Customers

**Create** **table** **Customers** (

customerId **int** **primary** **key** **auto\_increment** **not** **null**,

FirstName **varchar**(**50**),

LastName **varchar**(**50**),

Address **varchar**(**100**),

Phone **varchar**(**60**),

Email **varchar**(**100**),

Balance **int**,

Card **varchar**(**300**),

Promocode **varchar**(**100**)

);

* customerId - this is a unique ID for customer.
* FirstName – this shows customer’s first name.
* LastName - this shows customer’s last name.
* Address – this shows the address where the order will be delivered .
* Phone - this shows customer’s phone number.
* Email - this shows customer’s email.
* Balance - this shows the money in the customer’s account.
* Card - this shows card number of the customer.
* Promocode - this shows customer’s promo code.

5) Menu

**create** **table** **Menu**(

menuId **int** **primary** **key** **auto\_increment** **NOT** **NULL**,

menu\_name **varchar**(**150**),

resId **int**,

**constraint** fk\_rest

**foreign** **key** (resId)

**references** **Restaurant**(resId )

);

* menuId – this is a unique ID for menu.
* menu\_name – this shows the name of the menu.
* resId - this is id of restaurant which referring to the resId column in the Restaurant table.

6) MenuGroup

**Create** **table** **MenuGroup**(

groupId **int** **primary** **key** **auto\_increment** **not** **null**,

groupName **varchar**(**50**),

groupImage **binary**,

groupSize **enum**('small', 'medium', 'large'),

menuId **int** **not** **null**,

**constraint** fk\_menu

**foreign** **key** (menuId)

**references** **Menu**(menuId)

);

* groupId – this is a unique ID for menu group.
* groupName – this shows the name of the menu group
* groupImage – this is image of the menu group
* groupSize – this determines the size of menu group.

7) MenuItem

**Create** **table** **MenuItem** (

itemId **int** **primary** **key** **auto\_increment** **not** **null**,

itemName **varchar**(**50**),

itemImage **binary**,

itemDescription **varchar**(**100**),

price **double** ,

depoCount **int**,

itemType **enum**('main', 'extra'),

itemSize **enum**('small', 'medium', 'large'),

groupId **int** **not** **null**,

**constraint** fk\_menuGroup

**foreign** **key** (groupId)

**references** **MenuGroup**(groupId)

);

* itemId – this is a unique ID for menu group item.
* itemName – this is the name of the menu group item.
* itemIamge – this is the image of the menu group item.
* itemDescription – this describes the menu group item.
* price – this shows the price of the menu group item.
* depoCount – this shows how many items exist now in the depo.
* itemType – this determines the type of item.
* itemSize – this determines the size of menu group item.

8) Rating

**Create** **table** **Rating** (

ratingId **int** **primary** **key** **auto\_increment** **not** **null**,

score **int**,

date\_recorded **date**,

itemId **int**,

**constraint** fk\_menuItem

**foreign** **key**(itemId)

**references** **MenuItem**(itemId),

customerId **int**,

**constraint** fk\_customer

**foreign** **key**(customerId)

**references** **Customers**(customerId)

);

* ratingId – unique id for rating
* score – rating given by user
* date\_recorded – shows the date that score is given
* itemId- item that is reviewed
* customerId- the id of customer that rates

9) Delivery

**create** **table** **Delivery**(

deliveryId **int** **primary** **key** **auto\_increment** **not** **null**,

vehicleType **varchar**(**150**),

supportServiceName **varchar**(**150**),

workerId **int**,

**constraint** fk\_work

**foreign** **key**(workerId)

**references** **Workers**(workerId)

);

* ratingId – unique id for rating
* score – rating given by user
* date\_recorded – shows the date that score is given
* itemId- item that is reviewed
* customerId- the id of customer that rates

10) Orders

**create** **table** **Orders**(

orderId **int** **primary** **key** **auto\_increment** **not** **null**,

ordered\_date **datetime**,

takenOver\_date **datetime**,

order\_status **enum**('draft', 'ordered', 'preparing',

'checking', 'prepared', 'delivering', 'taken over', 'canceled'),

address **varchar**(**200**),

customerId **int**,

**constraint** fk\_cust

**foreign** **key**(customerId)

**references** **Customers**(customerId),

workerId **int**,

**constraint** fk\_worker

**foreign** **key**(workerId)

**references** **Workers**(workerId),

deliveryId **int**,

**constraint** fk\_delivery

**foreign** **key**(deliveryId)

**references** **Delivery**(deliveryId)

);

* orderId – this is a unique ID for order
* ordered\_date – this shows when customer ordered food
* takenOver\_date – this shows when order delivered to customer
* order\_status – this shows the status of the order
* customerId – this is id of customer which referring to the customerId column in the Customers table.
* workerId – this is id of worker which referring to the workerId column in the Worker table.
* deliveryId – this is id of delivery which referring to the deliveryId column in the Delivery table.

11) Order\_Item

**create** **table** **Order\_Item**(

oItemId **int** **primary** **key** **auto\_increment** **not** **null**,

quantity **int**,

orderId **int**,

**constraint** fk\_order

**foreign** **key** (orderId)

**references** **Orders**(orderId),

itemId **int**,

**constraint** fk\_item

**foreign** **key** (itemId)

**references** **MenuItem**(itemId)

);

* oItemtId – this is a unique ID for Order Item
* quantity – this shows the quantity of the ordered item
* orderId – this is id of order which referring to the orderId column in the Order table.
* itemId – this is id of menu item which referring to the itemId column in the MenuItem table.

12) Coupon

**create** **table** **Coupon**(

couponId **int** **primary** **key** **auto\_increment** **not** **null**,

coupon\_Name **varchar**(**30**),

coupon\_Code **varchar**(**30**),

discount **double**,

maxUse **int**,

resId **int**,

**constraint** fk\_restaurantId

**foreign** **key**(resId)

**references** **Restaurant**(resId)

);

* couponId – this is a unique ID for coupon
* coupon\_Name – this is a name of the coupon
* coupon\_Code – this is a code of the coupon
* discount – this shows how much user got discount
* maxUse – this shows how many times coupon can be use.
* resId – this is id of restaurant which referring to the resId column in the Restaurant table.

13) Payment

**create** **table** **Payment**(

paymentId **int** **primary** **key** **auto\_increment** **not** **null**,

totalAmount **int**,

paymentType **varchar**(**30**),

payment\_date **date**,

orderId **int**,

**constraint** fk\_orders

**foreign** **key**(orderId)

**references** **Orders**(orderId),

couponId **int**,

**constraint** fk\_coupon

**foreign** **key**(couponId)

**references** **Coupon**(couponId)

);

* paymentId – this is a unique ID for payment
* totalAmount – this shows the price of the customer’s order
* paymentType – this shows our payment is online or offline
* payment\_date – this shows when customer paid for the order
* orderId – this is id of order which referring to the orderId column in the Order table.
* couponId – this is id of order which referring to the couponId column in the Coupon table.
  1. **Alter, drop columns**

**alter** **table** Customers

**add** gender **varchar**(**30**)

ADD is used to add columns into the existing table. Sometimes we may require to add additional information, in that case we do not require to create the whole database again, ADD comes to our rescue.

**alter** **table** Customers

**drop** **column** gender

DROP COLUMN is used to drop column in a table. Deleting the unwanted columns from the table.

* 1. **SQL constraints**

SQL constraints are used to specify rules for the data in a table. Constraints are used to limit the type of data that can go into a table.

There is a primary key on the orderId column of the Orders table. It means we cannot have two records having the same value of that field.

There is a foreign key on the orderId column of the Payment table. It means values match the orderId column of the Orders table.

With the following statement we add constraint to the cost column of the Payment table. It checks a total amount for an order if value greater than 100.

**alter** **table** Payment

**add** **constraint** pk\_cost

**check** (totalAmount>=**100**)

We have many restaurant branches and each of them has a different opening date. For example, restaurants have been opening at 8 am on weekdays, but at 10 am on weekends. That is why our “Restaurant” table has an “Opening date id” foreign key link to the "Opening date" table and has one-to-many relation.

We have many restaurant branches and each of them has different worker. That is why worker table has “restaurant.id” foreign key link to the Restaurant table. With the “restaurant.id” we can identify which Branche our worker is working.

Workers have delivery and One worker can do one delivery that’s why ‘delivery service’ table has “worker.id” foreign key link to the Worker table.

Workers have order for delivery. One worker may have many orders. That’s why Order table has “worker.id” foreign key link to the Worker table.

1. **SQL queries. Using SQL to Retrieve Information from tables.**
   1. **Simple SQL queries**
2. This query is to obtain the name of the workers who are getting salary more than 1500 and working in the restaurant which id is 2. Here we used ASC. The ASC command is used to sort the data returned in ascending order. The following SQL statement selects some columns from the "workers" and "restaurant" table, sorted by the "FirstName" column:

**select** **concat**(w.FirstName, w.LastName) **as** worker,

r.resName **as** restaurant **from** Workers w, Restaurant r

**where** r.resId=**2** **and** w.Salary>**1500**

**order** **by** FirstName **asc**

1. The UPDATE statement is used to modify the existing records in a table. The WHERE clause specifies which record(s) that should be updated.

**update** Workers

**set** reportsTo = **null**

**where** workerId = **107**

1. The DELETE statement is used to delete existing records in a table. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted.

**delete** **from** Orders

**where** orderId=**1345**

1. This query shows the workers who get salary more than 1500 and works in Restaurant which ID is 2:

**select** concat(w.FirstName, w.LastName) **as** worker,

r.resName **as** restaurant

**from** Workers w, Restaurant r

**where** r.resId=2 **and** w.Salary>1500

**order** **by** worker **asc**

* 1. **Joining table**

1. The INNER JOIN keyword selects records that have matching values in both tables. The following query is to obtain the payment of the orders which is canceled in last week:

**select** totalAmount **from** Payment

**inner** **join** Orders **using** (orderId)

**where** order\_status = "canceled"

**and** ordered\_date>= day(**current\_date**()) - 7

1. The LEFT JOIN keyword returns all records from the left table, and the matching records from the right table . The result is 0 records from the right side if there is no match. The following query is to obtain the manager of the workers in the restaurant:

**select** ifnull(concat(w.FirstName, w.LastName),

'Top Manager') **as** 'Manager',

Concat(e.FirstName, e.LastName) **as** 'Worker'

**from** Workers e

**left** **join** Workers w **on** w.workerId=e.ReportsTo

**order** **by** manager **desc**

1. The RIGHT JOIN keyword returns all records from the right table , and the matching records from the left table . The following query is to find the name of the workers who prepare the order of the customer:

**select** Concat(w.FirstName, w.LastName) **as** ‘Worker Name’,

Concat(c.FirstName, c.LastName) **as** ‘Customer Name’ **from** Customers c

**inner** **join** Orders **using**(customerId)

**right** **join** Workers w **using**(workerId);

1. This query shows the delivery worker of the orders:

**select** concat(w.FirstName, w.LastName) **as** ‘Delivery Worker’,

o.orderId **as** ‘Order’ **from** Workers w

**inner** **join** Delivery d **using**(workerId)

**inner** **join** Orders o **using**(deliveryId)

1. This query shows the income of the food which is ordered more:

**select** m.itemName, m.price, o.quantity, m.itemId,

count(o.oItemId) **as** count **from** MenuItem m

**inner** **join** Order\_Item o **using**(itemId)

**inner** **join** Orders **using**(orderId)

**where** order\_status=’taken over’

**group** **by** itemId

* 1. SQL Procedure

A stored procedure is a segment of declarative SQL statements stored inside the MySQL Server.

1. The following procedure returns the order count of the food which name is sent as input:

DELIMITER $$

**create** **procedure** GetCountbyFood(

**in** foodId INT,

**out** orderCount INT

)

BEGIN

**declare** x decimal **default** 0;

**select** count(oItemId) **into** x **from** Order\_Item

**where** itemId= foodId;

set orderCount=x;

END $$

DELIMITER ;

1. The following procedure shows the food is ordered or not.

DELIMITER $$

**create** **procedure** GetOrderbyName(

**in** foodName varchar(200),

**out** orderStatus varchar(200))

Begin

**declare** iname varchar(200);

**select** itemName

**into** iname **from** MenuItem

**inner** **join** Order\_Item **using**(itemId)

**inner** **join** Orders **using**(orderId);

**if** foodName=iname **then**

set orderStatus='ordered';

**else**

set orderStatus='draft';

end **if**;

END $$

DELIMITER ;

1. The following procedure returns the count of customer’s orders:

DELIMITER $$

**create** **procedure** GetCountbyUser(

**in** customer INT,

**out** orderCount INT

)

BEGIN

**declare** countO int **default** 0;

**select** count(orderId) **into** countO **from** Orders

**inner** **join** Customers **using**(customerId)

**where** customerId=customer;

set orderCount=countO;

END $$

DELIMITER ;

1. The following procedure to return the worker who prepared the ordered food:

DELIMITER $$

**Create** **procedure** GetsWorkerbyFood(

**IN** foodName varchar(100),

**OUT** workerName varchar(100))

BEGIN

**declare** worker varchar(200);

**select** concat(w.FirstName, w.LastName) **into** worker **from** Workers w

**inner** **join** Orders **using**(workerId)

**inner** **join** Order\_item **using**(orderId)

**inner** **join** MenuItem m **using**(itemId)

**where** itemName=foodName;

set workerName=worker;

END $$

DELIMITER ;

1. The following procedure to return the income per days:

DELIMITER $$

**create** **procedure** GetTotalGainbyDay(

**in** Days varchar(150),

**out** Gain int

)

BEGIN

**declare** TotalGain decimal **default** 0;

**select** count(totalAmount) **into** TotalGain **from** Payment

**where** payment\_date=Days;

set Gain=TotalGain;

END $$

DELIMITER ;

1. The following procedure to return the delivery service of every order:

DELIMITER $$

**Create** **procedure** GetsDeliveryService(

**IN** orderName int,

**OUT** DeliveryService int

)

BEGIN

**declare** service int;

**select** deliveryId **into** service **from** Delivery

**inner** **join** Orders **using**(deliveryId)

**where** orderId=orderName;

set DeliveryService=service;

END $$

DELIMITER ;

1. The following procedure to return the rating of every menu item:

DELIMITER $$

**Create** **procedure** GetRating(

**IN** MenuName int,

**OUT** rating int

)

BEGIN

**declare** scoreR int;

**select** score **into** scoreR **from** Rating

**inner** **join** MenuItem **using**(itemId)

**where** itemId=MenuName;

set rating=scoreR;

END $$

DELIMITER ;

1. The following procedure to return the cancelled count of every menu item:

DELIMITER $$

**create** **procedure** GetCancellbyFood(

**in** foodName varchar(100),

**out** canceledCount int

)

BEGIN

**declare** counts int;

**select** count(orderId) **into** counts **from** Orders

**inner** **join** Order\_Item **using**(orderId)

**inner** **join** MenuItem **using**(itemId)

**where** order\_status='canceled' **and** foodName=itemName;

set canceledCount=counts;

END$$

DELIMITER ;

1. The following procedure to return the count of used coupon of every customer:

DELIMITER $$

**create** **procedure** GetCouponbyCustomer(

**in** customerName varchar(100),

**out** couponCount int

)

BEGIN

**declare** counts int;

**select** count(couponId) **into** counts **from** Coupon

**inner** **join** Payment **using**(couponId)

**inner** **join** Orders **using**(orderId)

**inner** **join** Customers c **using**(customerId)

**where** concat(c.FirstName, ' ', c.LastName)=customername;

set couponCount= counts;

END$$

DELIMITER ;

* 1. SQL Trigger

A trigger is a special type of stored procedure that automatically runs when an event occurs in the database server. DML triggers run when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view.

1. The following AFTER UPDATE ON trigger inserts a new row into the customer\_change table after a row from the Workers table is updated.

**CREATE** **TABLE** customer\_change (

id INT AUTO\_INCREMENT **PRIMARY** **KEY**,

customerId int **not** null,

FirstName varchar(50),

LastName varchar(50),

changeDate DATETIME **DEFAULT** NULL,

action VARCHAR(50) **DEFAULT** NULL

);

**CREATE** **TRIGGER** after\_customer\_update

after **UPDATE** **ON** Customers

**FOR** **EACH** ROW

**INSERT** **INTO** customer\_change

SET action = 'update',

customerId=OLD.customerId,

FirstName = OLD.FirstName,

LastName = OLD.LastName,

changeDate = NOW();

**insert** **into** Customers(FirstName, LastName)

**values**('Ruhan', 'Musayev'),

('Kemale', 'Agayev'),

('Ayten', 'Babayeva');

**update** Customers

set FirstName='Nezaket'

**where** customerId=2;

**select**\* **from** customer\_change;

A picture containing logo

Description automatically generated

1. The following BEFORE DELETE trigger inserts a new row into the WorkerArchives table before a row from the Workers table is deleted.

**CREATE** **TABLE** WorkerArchives (

id INT **PRIMARY** **KEY** AUTO\_INCREMENT,

workerId INT,

FirstName varchar(100),

LastName varchar(100),

Salary int **not** null **default** 0,

deletedAt TIMESTAMP **DEFAULT** NOW()

);

**insert** **into** Workers(FirstName, LastName, Salary)

**values**('Nuray', 'Qasimova', 1000),

('Leman', 'Hesenli', 800),

('Elgiz', 'Bayramov', 900);

DELIMITER $$

**create** **trigger** before\_worker\_delete

**before** **delete** **on** Workers **for** **each** row

begin

**insert** **into** WorkerArchives(workerId, FirstName, LastName, Salary)

**values**(old.workerId, old.FirstName, old.LastName, old.Salary);

END$$

DELIMITER ;

**delete** **from** Workers

**where** workerId=2;

**select** \* **from** WorkerArchives;

Graphical user interface, application, Teams

Description automatically generated

* 1. SQL Views

A view is a virtual table based on the result-set of an SQL statement.

1. The following view shows the manager worker relation:

**CREATE** VIEW ManagerWorker **AS**

**select** ifnull(Concat(m.FirstName, m.LastName), 'Top Manager')

**as** Manager,

Concat(w.FirstName, w.LastName) **as** Worker

**from** Workers m

**left** **join** Workers w **on**

m.workerId=w.ReportsTo

**order** **by** manager **desc**;

**SELECT** \* **FROM** ManagerWorker;

Graphical user interface, table

Description automatically generated with medium confidence

1. The following CustomerOrders view to show every customer’s order and their total:

**CREATE** VIEW customerOrder **AS**

**SELECT**

concat(FirstName,LastName),

oItemId,

sum(quantity\*price) **as** total

**FROM**

Order\_Item

**INNER** **JOIN** Orders **USING** (orderId)

**INNER** **JOIN** MenuItem **USING** (itemId)

**INNER** **JOIN** Customers **USING** (customerId)

**GROUP** **BY** oItemId;

**select** \* **from** customerOrders;

1. The following view to show customer and worker who serve them:

**Create** VIEW workerCustomer **AS**

**SELECT** Concat(w.FirstName,w.LastName) **as** 'Worker Name',

Concat(c.FirstName,c.LastName) **as** 'Customer Name'

**FROM** Customers c

**INNER** **JOIN** Orders **USING**(customerId)

**right** **join** Workers w **using**(workerId)

**GROUP** **BY** workerId;

**select** \* **from** workerCustomer;

**Table

Description automatically generated**

1. The following view to show workers who get salary more than 1500 and work on restaurant ID=2:

**CREATE** VIEW WorkerRestaurant **AS**

**select** concat(w.FirstName, w.LastName) **as** worker,

r.resName **as** restaurant

**from** Workers w, Restaurant r

**where** r.resId=2 **and** w.Salary>1500

**order** **by** worker **asc**;

**SELECT** \* **FROM** WorkerRestaurant;

* 1. SQL Indexer

An index is a data structure such as B-Tree that improves the speed of data retrieval on a table at the cost of additional writes and storage to maintain it.

1. The following indexer created for the jobTitle column on the Workers table:

**CREATE** **INDEX** jobTitle **ON** Workers(jobTitle);

**EXPLAIN** **SELECT**

workerId,

FirstName,

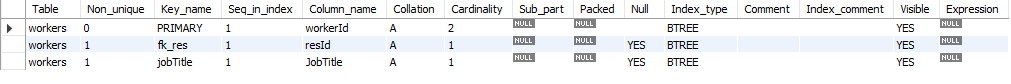
LastName

**FROM**

Workers

**WHERE**

jobTitle = 'Delivery';



1. The following indexer created for FirstName column on the Customer table:

**CREATE** **INDEX** CustomersName **ON** Customers (FirstName(50));

**EXPLAIN** **SELECT**

FirsName,

LastName

**FROM**

Customers

**WHERE**

FirstName = 'Ruhan';

**SHOW** INDEXES **FROM** Customers;

**Chapter 4. Conclusion**

All large enterprises need the database systems for handling the information. One kind of those enterprises is the Food Service System. Here in this course work we have created SQL Database Management System.

Because of large number of orders, customers and other staff in Food Service, data processing becomes more crucial. Data Management in Food Service System can be used for achieving the worker’s information, arranging the orders preparing and accounting business.