CS 6360.001 DATABASE DESIGN

Uber

Nandhini Periasamy

Gowtham Jeyaprakash

Ziyad Amir Dhuka

The University of Texas, Dallas

800 W Campbell Rd, Richardson, TX 75080

PROJECT DESCRIPTION

Uber is a cab providing (car-for-hire) company that provides taxi service to the people, its customers. Unlike regular taxi services, Uber lets you join as a driver with your personally registered vehicle and make money when you complete a customer request.

Important Components of the System:

The main important actors in the domain are – Drivers and Customers, which form the basic foundation of the Database Model.

Driver – a person above the age of 18 having an authorised unexpired driver's license and SSN, possessing a registered vehicle under insurance.

Customer - a person having an account with Uber who needs to be picked up from his location and dropped at the drop-off point.

The working of Uber System

Uber is a real time application that allows a customer to request for a ride from the current

location of the customer to their desired destination.

- The customer requests for a ride and has the flexibility of choosing the type of ride (UberX, UberXL). The estimated price is visible to the customer based on the type of ride chosen. The customer gets allotted a ride according to the availability of the drivers in his current location.
- Once the trip request has been finalized, a driver arrives at the location of the customer to pick up the customer.
- After reaching the desired final destination, the driver ends the trip, and the customer can see the final fare of the trip.
- Once the trip has been completed, the fare gets reduced from the payment method option chosen by the customer at the beginning of the trip. The customer has the option of adding tip amount to the driver. Moreover, both the driver and the customer can rate each other based on the trip and provide comments/feedback if necessary.

PROJECT DATA REQUIREMENTS

A User entity has been created and the user type can be either a customer or a driver which are both registered with Uber and regarded as Uber Users.

The Uber system must store the personal information about the **User** such as name, gender, date of birth, address, email, phone number and rating. It must also identify the users uniquely by a system generated identification code (user_ID).

Additional information regarding the **driver** must also be stored such as Driver's License Number, DL Expiry date, SSN and his location.

A user in the system can either be a customer or a driver. A driver can also be a customer when he/she is not riding their vehicle.

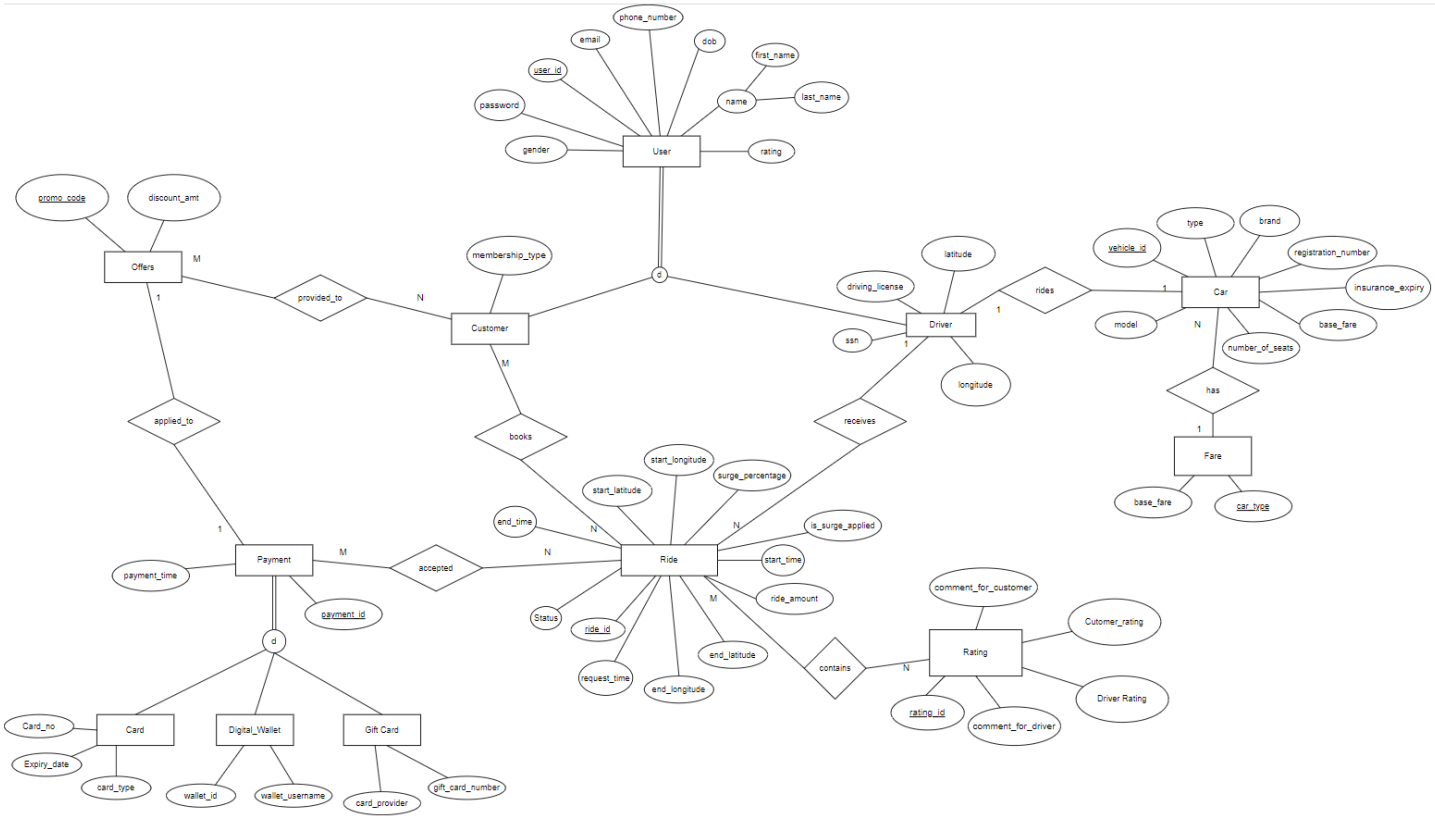
Car will contain the Vehicle identification number (VID), model of the car, the manufacturing year that when it was created, the capacity of the car that how many seats are available, the insurance number and the insurance expiry date which will be required for Insurance renewal.

Ride will be identified with some specific id (Trip ID) and will be containing the information about the type of the trip, pickup location, drop-off location, start time, end time, request time, surge_percentage, is_SurgeApplied and estimated fare. It will also contain the driver ID who will be allotted a ride and the customer ID who will be booking the ride.

Payment will contain the transaction ID (TID), payment_type, payment_time, the billing address where the bill invoice is to be sent. It will also have the option to redeem a promo code or an offer by the customer.

In the end, the feedback is very important for Uber which helps the people to know about the review of the whole business. So, **Rating** table has been constructed which will contain the trip ID, the ratings given to the driver and the customer which will contain the number of stars out of 10 along the feedback for the driver and the customer.

ER DIAGRAM



ASSUMPTIONS

- A user can be a customer or driver at any point of time.
- A customer can book many rides and a ride can be pooled by a maximum of 4 customers
- A driver should have entry of driving_license_expiry which will be validated using triggers.
- A driver can receive many ride requests and a ride can be handled by only one ride
- Offer can be applied by many customers and a customer can apply many offers for payment
- Payment can be of any type -> card, wallet and gift_card
- A ride can have many ratings in case of uber pooling and a rating corresponds to only one ride
- A driver rides only one car and a car can be driven by only one driver.
- A ride can have m payments in case of uber pooling but a payment corresponds to only ride.

ONE-TO-ONE BINARY RELATIONSHIPS

- Each PAYMENT can use only one OFFER, each OFFER can be used only on one PAYMENT.
- Each DRIVER can ride only one RIDE, each RIDE can be done by only one DRIVER.
- Each DRIVER rides only one CAR, each CAR can be driven by only one DRIVER.

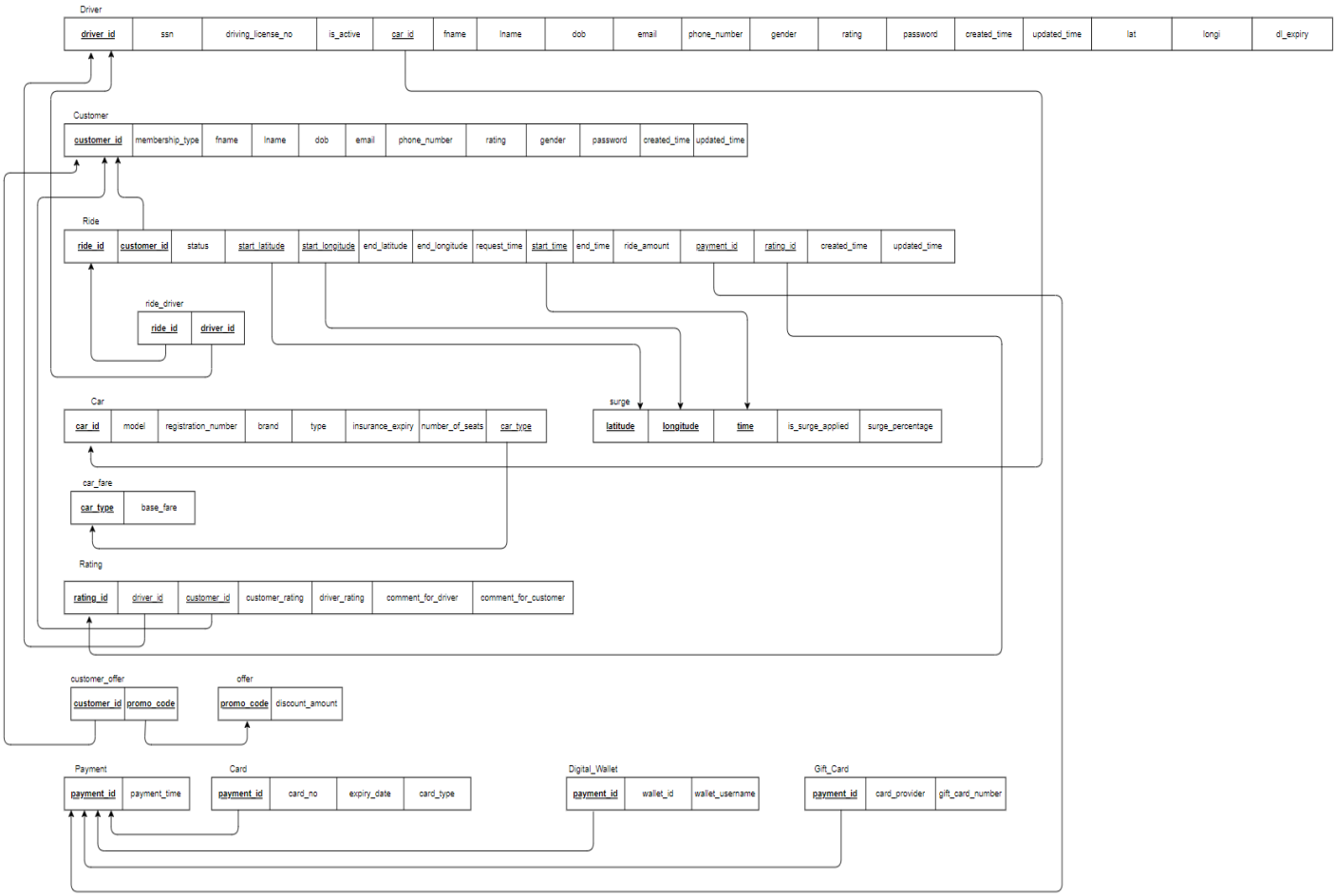
ONE-TO-MANY BINARY RELATIONSHIPS

- Each RIDE can have number of payments, but each PAYMENT can have only one RIDE.
- Each RIDE can have number of RATINGS, but each RATING will be only of one RIDE.
- Each DRIVER can do many RIDES, but each RIDE will be done by only one DRIVER.
- Each CAR will have only one FARE, but each FARE depends on the car chosen.

MANY-TO-MANY BINARY RELATIONSHIPS

- A CUSTOMER can book any number of rides, and a RIDE can be shared by any number of CUSTOMERS.
- A CUSTOMER can avail many OFFERS, and an OFFER can be availed by many CUSTOMERS.

RELATIONAL SCHEMA



DATABASE DEPENDENCIES

Functional Dependencies for the System

DRIVER

DRIVER_ID → SSN, DRIIVING_LICENSE_NO, IS_ACTIVE, CAR_ID, FNAME. LNAME, DOB, EMAIL, PHONE_NUMBER, GENDER, RATING, PASSWORD, CREATED_TIME, UPDATE_TIME

CUSTOMER

CUSTOMER_ID → MEMBERSHIP_TYPE, FNAME, LNAME, DOB, EMAIL, PHONE_NUMBER, RATING, GENDER, PASSWORD, CREATED_TIME, UPDATE_TIME

PAYMENT

PAYMENT_ID → PAYMENT_TIME

CARD

PAYMENT_ID → CARD_NO, EXPIRY_DATE, CARD_TYPE

DIGITAL WALLET

PAYMENT_ID → WALLET_ID, WALLET_USERNAME

GIFT CARD

PAYMENT_ID → CARD_PROVIDER, GIFTCARD_NUMBER

RATING

RATING_ID → DRIVER_ID, CUSTOMER_ID, CUSTOMER_RATING, DRIVER_RATING, COMMENT-FOR_DRIVER, COMMENT_FOR_CUSTOMER

OFFER

PROMO_CODE → DISCOUNT_AMOUNT

CAR

CAR_ID → MODEL, REGISTRATION_NUMBER, BRAND, TYPE, NUMBER_OFSEATS, PLATE_NUMBER, INSURANCE_EXPIRY, CAR_TYPE,

CAR FARE

CAR_TYPE → BASE_FARE

RIDE DRIVER

RIDE_ID → DRIVER_ID

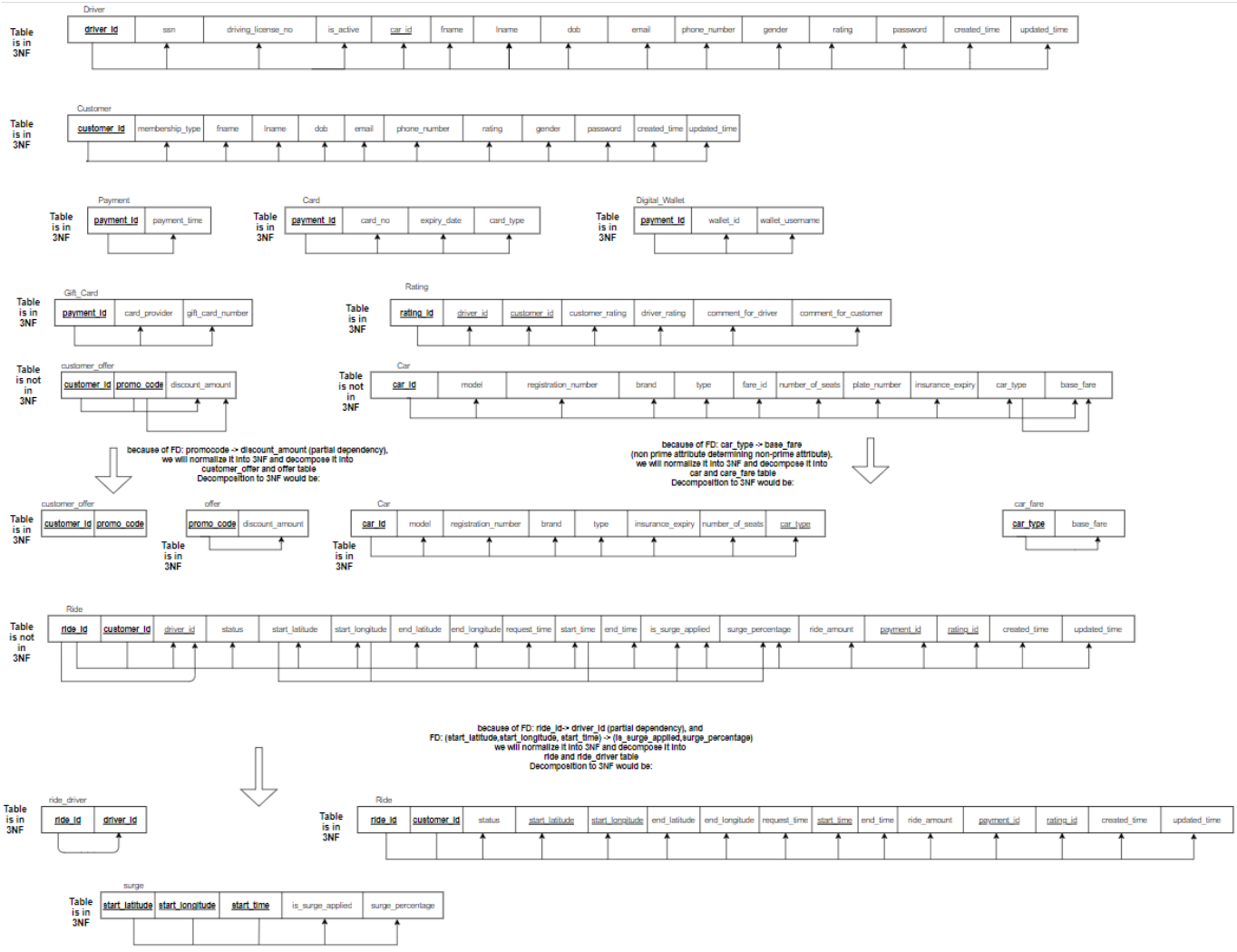
RIDE

RIDE_ID, CUSTOMER_ID → STATUS, START_LATITUDE, START_LONGITUDE, END_LATITUDE, END_LOGITUDE, REQUEST_TIME, START_TIME, END_TIME, RIDE_AMOUNT, PAYMENT_ID, RATING_ID, CREATED_TIME, UPDATED_TIME

SURGE

START_LATITUDE, START_LONGITUDE, START_TIME →IS_SURGE_APPLIED, SURGE_PERCENTAGE

NORMALIZATION DIAGRAM



PROCEDURES

```
-- ----------------------------------------------------

-- procedure find_nearby_drivers

-- ----------------------------------------------------


DELIMITER $$
USE `uber`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `find_nearby_drivers`(IN ride_id INT)
BEGIN
        DECLARE finished INTEGER DEFAULT 0;
        declare latitude,longitude VARCHAR(45);
   declare driver_lat,driver_long VARCHAR(45);
   declare driver_id int;
   declare distance varchar(10);
   declare driver cursor for select lat,longi,id from driver where is_active=1;
```

```
    declare CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
    drop table nearby_drivers;
    create temporary table nearby_drivers(id int, distance varchar(10));
        SELECT start_latitude, start_longitude into latitude,longitude from ride where id=ride_id;
        OPEN driver;
        getDriverInfo: LOOP
            FETCH driver INTO driver_lat, driver_long, driver_id;
            IF finished = 1 THEN

                LEAVE getDriverInfo;

            END IF;
            -- build email list
      set distance = getDistance(latitude,longitude,driver_lat,driver_long);
      if(distance < 25.0)
      then
                LEAVE getDriverInfo;
            end if;
        END LOOP getDriverInfo;
        CLOSE driver;
    select * from nearby_drivers;
END$$

DELIMITER ;
```

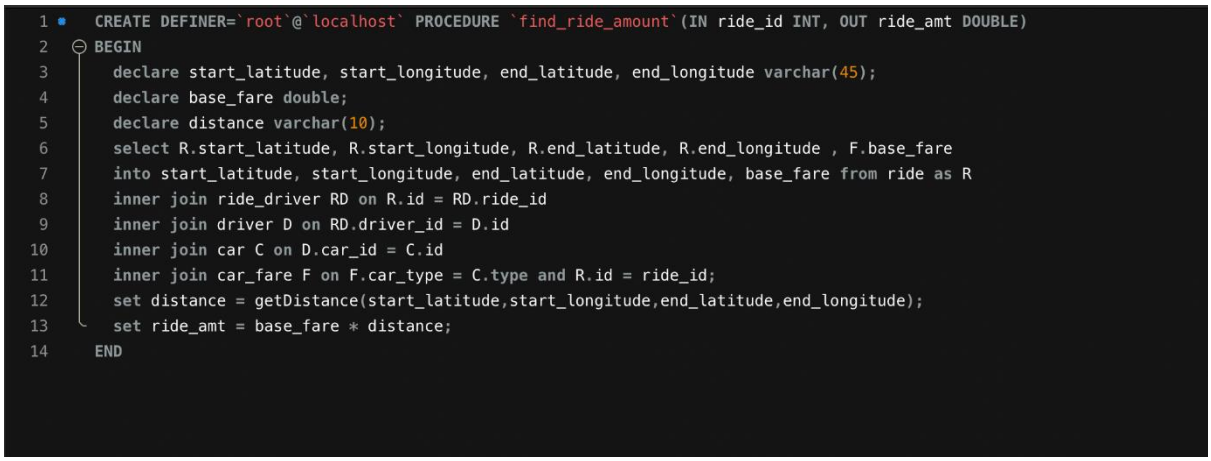
```
CREATE DEFINER=`root`@`localhost` PROCEDURE `find_nearby_drivers`(IN ride_id INT)
BEGIN
    DECLARE finished INTEGER DEFAULT 0;
    declare latitude,longitude VARCHAR(45);
    declare driver_lat,driver_long VARCHAR(45);
    declare driver_id int;
    declare distance varchar(10);
    declare driver cursor for select lat,longi,id from driver where is_active=1;
    declare CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
    drop table nearby_drivers;
    create temporary table nearby_drivers(id int, distance varchar(10));
    SELECT start_latitude, start_longitude into latitude,longitude from ride where id=ride_id;
    OPEN driver;
    getDriverInfo: LOOP
        FETCH driver INTO driver_lat, driver_long, driver_id;
        IF finished = 1 THEN
            LEAVE getDriverInfo;
        END IF;
        -- build email list
        set distance = getDistance(latitude,longitude,driver_lat,driver_long);
        if(distance < 25.0)
        then
            insert into nearby_drivers values(driver_id,distance);
        end if;
    END LOOP getDriverInfo;
    CLOSE driver;
    select * from nearby_drivers;
END
```

-- ----------

-- procedure find_ride_amount

-- ----------

```
DELIMITER $$
USE `uber`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `find_ride_amount`(IN ride_id INT, OUT ride_amt
DOUBLE)
BEGIN
 declare start_latitude, start_longitude, end_latitude, end_longitude varchar(45);
 declare is_surge boolean;
 declare surge_per double;
 declare base_fare double;
 declare distance varchar(10);
 select R.start_latitude, R.start_longitude, R.end_latitude, R.end_longitude , R.is_surge_applied,
R.surge_percentage , F.base_fare
 into start_latitude, start_longitude, end_latitude, end_longitude, is_surge, surge_per, base_fare
from ride as R
 inner join driver D on R.driver_id = D.id
 inner join car C on D.car_id = C.id
 inner join Fare F on F.car_type = C.type and R.id = ride_id;
 set distance = getDistance(start_latitude,start_longitude,end_latitude,end_longitude);
 if(is_surge)
 then
        set ride_amt = base_fare * distance * surge_per;
 else
        set ride_amt = base_fare * distance;
 end if;
END$$

DELIMITER ;
```



```
CREATE DEFINER=`root`@`localhost` PROCEDURE `find_ride_amount`(IN ride_id INT, OUT ride_amt DOUBLE)
BEGIN
  declare start_latitude, start_longitude, end_latitude, end_longitude varchar(45);
  declare base_fare double;
  declare distance varchar(10);
  select R.start_latitude, R.start_longitude, R.end_latitude, R.end_longitude , F.base_fare
  into start_latitude, start_longitude, end_latitude, end_longitude, base_fare from ride as R
  inner join ride_driver RD on R.id = RD.ride_id
  inner join driver D on RD.driver_id = D.id
  inner join car C on D.car_id = C.id
  inner join car_fare F on F.car_type = C.type and R.id = ride_id;
  set distance = getDistance(start_latitude,start_longitude,end_latitude,end_longitude);
  set ride_amt = base_fare * distance;
END
```

-- ----------

-- function getDistance

-- ----------

```
DELIMITER $$
USE `uber`$$
CREATE DEFINER=`root`@`localhost` FUNCTION `getDistance`(`lat1` VARCHAR(45), `lng1`
VARCHAR(45), `lat2` VARCHAR(45), `lng2` VARCHAR(45)) RETURNS varchar(10) CHARSET utf8
    READS SQL DATA
    DETERMINISTIC
begin
declare distance varchar(10);
set distance = (select (6371 * acos(
             cos( radians(lat2) )
           * cos( radians( lat1 ) )
           * cos( radians( lng1 ) - radians(lng2) )
           + sin( radians(lat2) )
           * sin( radians( lat1 ) )) ) as distance);
if(distance is null)
then
 return '';
else
return distance;
end if;
end$$

DELIMITER ;
USE `uber`;

DELIMITER $$
USE `uber`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `uber`.`car_BEFORE_INSERT`
BEFORE INSERT ON `uber`.`car`
FOR EACH ROW
BEGIN
            if(NEW.insurance_expiry < sysdate())
    then
                        signal sqlstate '45000' set message_text = 'Expiry date of driving license should be
greater than current date';
    end if;
END$$


USE `uber`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `uber`.`driver_BEFORE_INSERT`
BEFORE INSERT ON `uber`.`driver`
FOR EACH ROW
BEGIN
            if(NEW.dl_expiry < sysdate())
    then
```

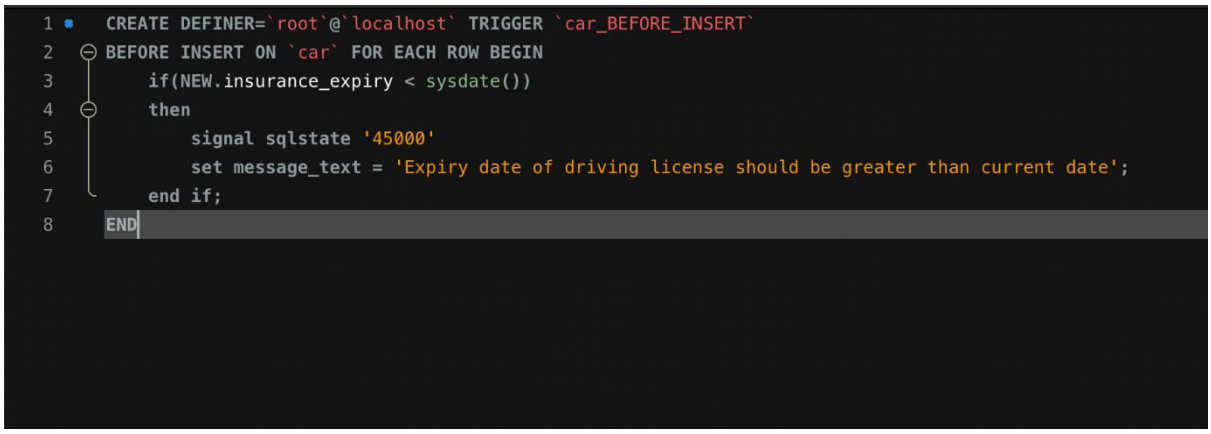
```
                signal sqlstate '45000' set message_text = 'Expiry date of driving license should be
greater than current date';
   end if;
END$$

USE `uber`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `uber`.`driver_BEFORE_UPDATE`
BEFORE UPDATE ON `uber`.`driver`
FOR EACH ROW
BEGIN
        if(NEW.dl_expiry < sysdate)
   then
                signal sqlstate '45000' set message_text = 'Expiry date of driving license should be
greater than current date';
   end if;
END$$

USE `uber`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `uber`.`ride_BEFORE_INSERT`
BEFORE INSERT ON `uber`.`ride`
FOR EACH ROW
BEGIN
        SET @ride_amt = 0.0;
   call find_ride_amount(NEW.id,@ride_amt);

   SET NEW.ride_amount = @ride_amt;

END$$

DELIMITER ;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

TRIGGERS

- Trigger for checking the expiry of the driver license

```
CREATE DEFINER=`root`@`localhost` TRIGGER `car_BEFORE_INSERT`
BEFORE INSERT ON `car` FOR EACH ROW BEGIN
    if(NEW.insurance_expiry < sysdate())
    then
        signal sqlstate '45000'
        set message_text = 'Expiry date of driving license should be greater than current date';
    end if;
END
```

- Trigger for checking the insurance of the car



APPENDIX

-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE SCHEMA IF NOT EXISTS `uber` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci ;
USE `uber` ;

-- ---
-- Table `uber`.`car`

```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`car` (
  `id` INT NOT NULL,
  `model` VARCHAR(45) NULL DEFAULT NULL,
  `registration_number` VARCHAR(45) NULL DEFAULT NULL,
  `brand` VARCHAR(45) NULL DEFAULT NULL,
  `type` ENUM('UBER_GO', 'UBER_XL', 'UBER_XXL') NULL DEFAULT NULL,
  `insurance_expiry` DATE NULL DEFAULT '2015-09-13',
  PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `uber`.`car_fare`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`car_fare` (
  `car_type` ENUM('UBER_GO', 'UBER_XL', 'UBER_XXL') NOT NULL,
  `base_fare` DOUBLE NULL DEFAULT NULL,
  PRIMARY KEY (`car_type`),
  UNIQUE INDEX `car_type_UNIQUE` (`car_type` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `uber`.`offer`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`offer` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `promo_code` VARCHAR(45) NOT NULL,
  `discount_amt` DOUBLE NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 3
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `uber`.`payment`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`payment` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `offer_id` INT NULL DEFAULT NULL,
  `ride_amount` VARCHAR(45) NULL DEFAULT NULL,
  `payment_amount` VARCHAR(45) NULL DEFAULT NULL,
  `payment_time` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
```

```
  INDEX `payment_offer_id_fk_idx` (`offer_id` ASC) VISIBLE,
  CONSTRAINT `payment_offer_id_fk`
    FOREIGN KEY (`offer_id`)
    REFERENCES `uber`.`offer` (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 3
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `uber`.`card`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`card` (
  `payment_id` INT NOT NULL,
  `card_no` VARCHAR(16) NULL DEFAULT NULL,
  `expiry_date` VARCHAR(16) NULL DEFAULT NULL,
  `card_type` ENUM('AMEX', 'MASTER_CARD', 'VISA') NULL DEFAULT NULL,
  PRIMARY KEY (`payment_id`),
  CONSTRAINT `card_id_fk`
    FOREIGN KEY (`payment_id`)
    REFERENCES `uber`.`payment` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `uber`.`user`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`user` (
  `user_id` INT NOT NULL AUTO_INCREMENT,
  `fname` VARCHAR(50) NOT NULL,
  `lname` VARCHAR(50) NOT NULL,
  `dob` DATE NOT NULL,
  `email` VARCHAR(100) NOT NULL,
  `phone_number` VARCHAR(10) NOT NULL,
  `rating` DOUBLE NULL DEFAULT NULL,
  `gender` ENUM('MALE', 'FEMALE', 'TRANSGENDER') NOT NULL,
  `created_time` DATETIME NOT NULL,
  `updated_time` DATETIME NULL DEFAULT NULL,
  `password` VARCHAR(45) NOT NULL DEFAULT 'PASSWORD',
  PRIMARY KEY (`user_id`),
  UNIQUE INDEX `user_id_UNIQUE` (`user_id` ASC) VISIBLE)
ENGINE = InnoDB
AUTO_INCREMENT = 9
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
```

```
-- Table `uber`.`customer`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`customer` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `membership_type` ENUM('GOLD', 'PREMIUM', 'ORDINARY') NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `customer_user_fk`
    FOREIGN KEY (`id`)
    REFERENCES `uber`.`user` (`user_id`)
    ON DELETE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 5
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `uber`.`customer_offer`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`customer_offer` (
  `customer_id` INT NOT NULL,
  `offer_id` INT NOT NULL,
  PRIMARY KEY (`customer_id`, `offer_id`),
  INDEX `co_offer_fk_idx` (`offer_id` ASC) VISIBLE,
  CONSTRAINT `co_cust_fk`
    FOREIGN KEY (`customer_id`)
    REFERENCES `uber`.`customer` (`id`),
  CONSTRAINT `co_offer_fk`
    FOREIGN KEY (`offer_id`)
    REFERENCES `uber`.`offer` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `uber`.`didgital_wallet`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`didgital_wallet` (
  `payment_id` INT NOT NULL,
  `wallet_id` INT NULL DEFAULT NULL,
  `wallet_username` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`payment_id`),
  CONSTRAINT `wallet_payment_id_fk`
    FOREIGN KEY (`payment_id`)
    REFERENCES `uber`.`payment` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `uber`.`driver`
```

```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`driver` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `ssn` VARCHAR(10) NOT NULL,
  `driving_license_no` VARCHAR(45) NOT NULL,
  `is_active` TINYINT NULL DEFAULT NULL,
  `lat` VARCHAR(45) NULL DEFAULT NULL,
  `longi` VARCHAR(45) NULL DEFAULT NULL,
  `car_id` INT NULL DEFAULT NULL,
  `dl_expiry` DATE NULL DEFAULT '2025-09-13',
  PRIMARY KEY (`id`),
  INDEX `driver_car_fk_idx` (`car_id` ASC) VISIBLE,
  CONSTRAINT `driver_car_fk`
    FOREIGN KEY (`car_id`)
    REFERENCES `uber`.`car` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `driver_user_fk`
    FOREIGN KEY (`id`)
    REFERENCES `uber`.`user` (`user_id`)
    ON DELETE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 9
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `uber`.`gift_card`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`gift_card` (
  `payment_id` INT NOT NULL,
  `card_provider` VARCHAR(45) NULL DEFAULT NULL,
  `gift_card_number` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`payment_id`),
  CONSTRAINT `gift_payment_id_fk`
    FOREIGN KEY (`payment_id`)
    REFERENCES `uber`.`payment` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;




-- -----------------------------------------------------
-- Table `uber`.`rating`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`rating` (
```

```
  `id` INT NOT NULL AUTO_INCREMENT,
  `customer_rating` DOUBLE NULL DEFAULT NULL,
  `driver_rating` DOUBLE NULL DEFAULT NULL,
  `comment_for_driver` VARCHAR(200) NULL DEFAULT NULL,
  `comment_for_customer` VARCHAR(200) NULL DEFAULT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 3
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `uber`.`ride`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`ride` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `status` ENUM('CANCELLED', 'BOOKED', 'DRIVER_ASSIGNED', 'DRIVER_REQUESTING',
'RIDE_STARTED', 'RIDE_COMPLETED') NOT NULL,
  `customer_id` INT NOT NULL,
  `start_latitude` VARCHAR(45) NOT NULL,
  `start_longitude` VARCHAR(45) NOT NULL,
  `end_latitude` VARCHAR(45) NOT NULL,
  `end_longitude` VARCHAR(45) NOT NULL,
  `start_time` DATETIME NULL DEFAULT NULL,
  `end_time` DATETIME NULL DEFAULT NULL,
  `created_time` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `updated_time` DATETIME NULL DEFAULT NULL,
  `is_surge_applied` TINYINT NULL DEFAULT NULL,
  `surge_percentage` DOUBLE NULL DEFAULT NULL,
  `ride_amount` DOUBLE NULL DEFAULT NULL,
  `payment_id` INT NULL DEFAULT NULL,
  `rating_id` INT NULL DEFAULT NULL,
  `ride_type` ENUM('NORMAL', 'UBER_POOL') NOT NULL DEFAULT 'NORMAL',
  PRIMARY KEY (`id`, `customer_id`),
  INDEX `ride_customer_id_foreign` (`customer_id` ASC) VISIBLE,
  INDEX `ride_payment_id_foreign_idx` (`payment_id` ASC) VISIBLE,
  INDEX `ride_payment_id_foreign_idx1` (`payment_id` ASC) VISIBLE,
  INDEX `ride_rating_id_foreign` (`rating_id` ASC) VISIBLE,
  CONSTRAINT `ride_customer_id_foreign`
    FOREIGN KEY (`customer_id`)
    REFERENCES `uber`.`customer` (`id`),
  CONSTRAINT `ride_payment_id_fk`
    FOREIGN KEY (`payment_id`)
    REFERENCES `uber`.`payment` (`id`)
    ON UPDATE CASCADE,
  CONSTRAINT `ride_rating_id_foreign`
    FOREIGN KEY (`rating_id`)
    REFERENCES `uber`.`rating` (`id`)
    ON DELETE SET NULL
    ON UPDATE CASCADE)
```

```
ENGINE = InnoDB
AUTO_INCREMENT = 3
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `uber`.`ride_driver`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`ride_driver` (
  `ride_id` INT NOT NULL,
  `driver_id` INT NOT NULL,
  PRIMARY KEY (`ride_id`, `driver_id`),
  INDEX `ride_driver_fk_idx` (`driver_id` ASC) VISIBLE,
  CONSTRAINT `ride_driver_fk`
    FOREIGN KEY (`driver_id`)
    REFERENCES `uber`.`driver` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `ride_id_fk`
    FOREIGN KEY (`ride_id`)
    REFERENCES `uber`.`ride` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `uber`.`surge`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `uber`.`surge` (
  `latitude` VARCHAR(50) NOT NULL,
  `longitude` VARCHAR(45) NOT NULL,
  `time` VARCHAR(45) NOT NULL,
  `is_surge_applied` TINYINT NULL DEFAULT NULL,
  `surge_percentage` DOUBLE NULL DEFAULT NULL,
  PRIMARY KEY (`latitude`, `longitude`, `time`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```