Βάσεις Δεδομένων

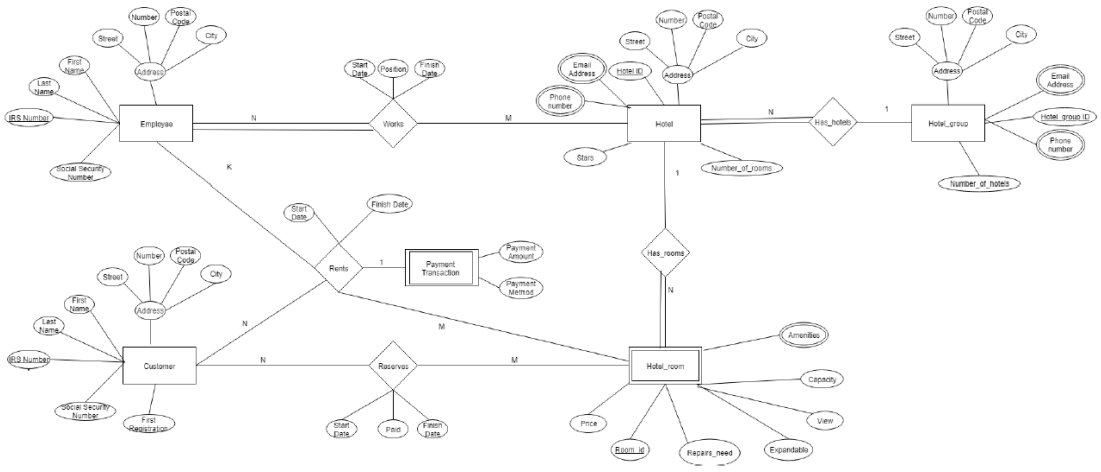
Εξαμηνιαίο Project με θέμα "e-Hotels"

ΣΧΟΛΗ: ΣΗΜΜΥ

Ονοματεπώνυμο	Αριθμός Μητρώου
Γιάννης Πιτόσκας	03115077
Γιώργος Χιονάς	03115132
Αντώνης Παπαοικονόμου	03115140

a. Σχεσιακό Διάγραμμα

Στο project μας δόθηκε το παρακάτω E-R διάγραμμα:

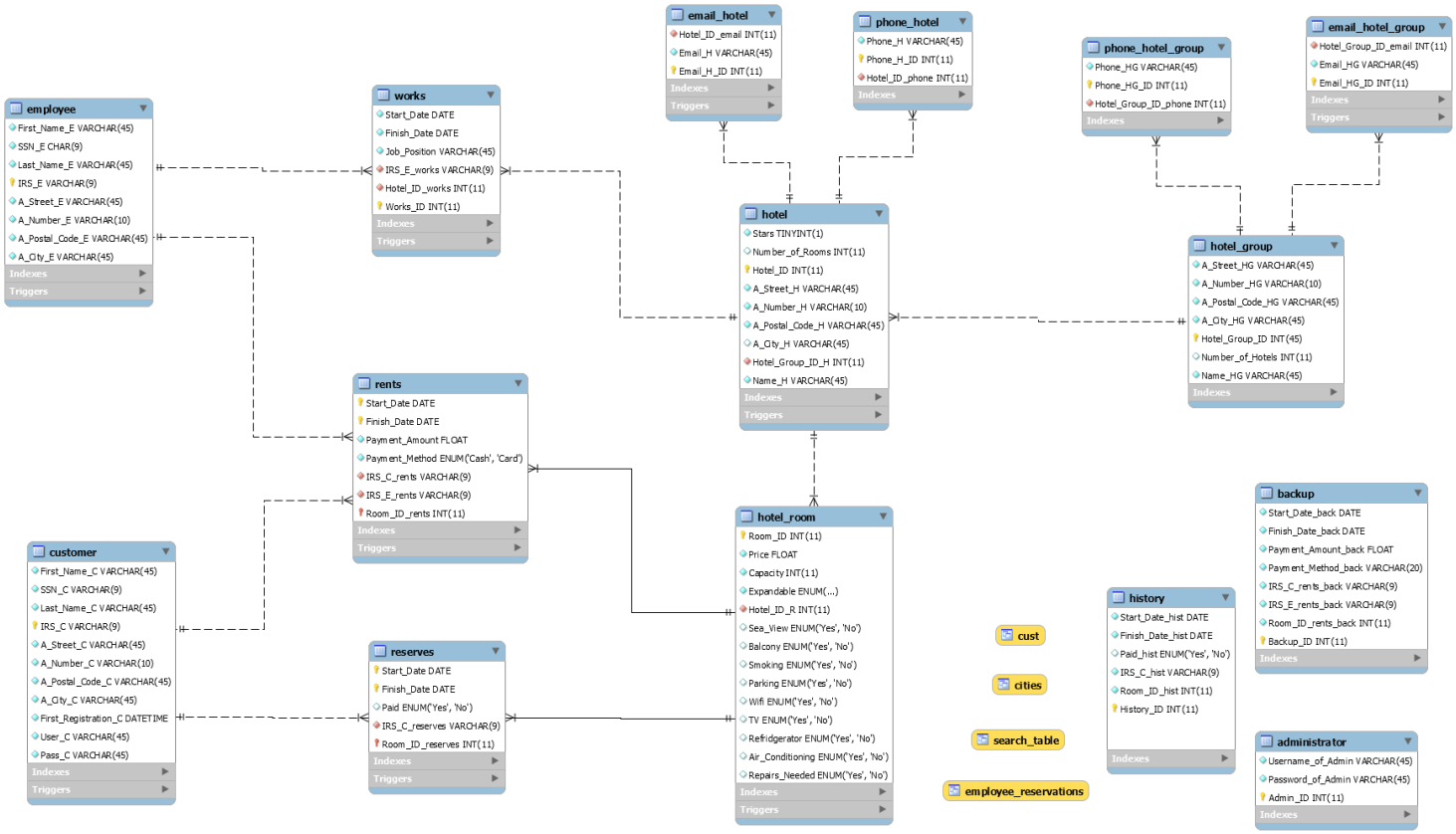


Πάνω σε αυτό το E-R διάγραμμα βασιστήκαμε για την κατασκευή της βάσης μας. Ωστόσο προσθέσαμε κάποια entities και κάποια attributes:

- προσθήκη Username, Password στον Customer
- προσθήκη Name στο Hotel Group
- προσθήκη Name στο Hotel
- επιλογή Amenities αυθαίρετα ως: Balcony, Smoking, Parking, WiFi, TV, Refrigerator, Air-conditioning
- δημιουργία του entity "History" που περιλαμβάνει το ιστορικό όλων των κρατήσεων δηλαδή τα περιεχόμενα του "Reserves" ανεξάρτητα από το αν διαγραφούν ύστερα (το "History" περιλαμβάνει attributes όμοια με αυτά του "Reserves")
- δημιουργία του entity "Backup" που περιλαμβάνει το ιστορικό όλων των ενοικιάσεων δηλαδή τα περιεχόμενα του "Rents" ανεξάρτητα από το αν διαγραφούν ύστερα (το "Backup" περιλαμβάνει attributes όμοια με αυτά του "Rents")
- στο "Payment Transaction" οι επιλογές είναι δύο "Cash", "Credit Card"

- προσθήκη του entity Administrator το οποίο θα περιέχει τα username και τα passwords των διαχειριστών οι οποίοι έχουν την δυνατότητα να εισάγουν νέες πληροφορίες, να αλλάζουν ήδη υπάρχουσες πληροφορίες και να διαγράφουν πληροφορίες σχετικά με τα hotel groups, τα hotel, τα hotel rooms, τους employees και τους customers καθώς επίσης έχουν πρόσβαση στο ιστορικό των κρατήσεων και των ενοικιάσεων ενοικιάσεων (αλλά δεν μπορούν να «πειράξουν» κανένα στοιχείο σε αυτά τα δύο πεδία)

Λαμβάνοντας όλα τα παραπάνω υπ' όψιν προκύπτει το ακόλουθο σχεσιακό διάγραμμα:



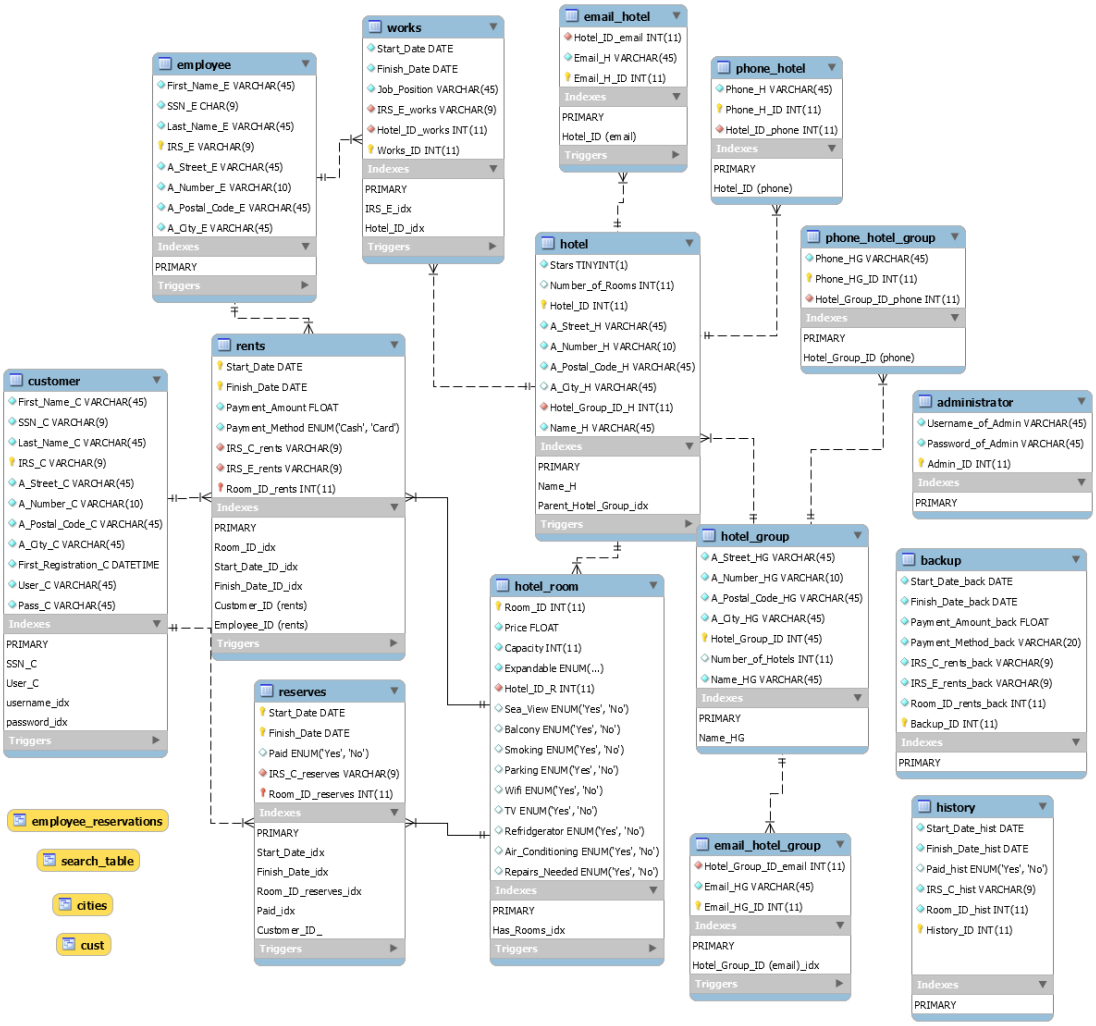
Παραπάνω φαίνονται και τέσσερα κίτρινα πλαίσια τα οποία αντιστοιχούν σε τέσσερα διαφορετικά views που έχουμε κατασκευάσει. Τα έχουμε χρησιμοποιήσει προκειμένου να κρύψουμε ευαίσθητες πληροφορίες (π.χ. το username και το password ενός customer).

b. Περιορισμοί

Στα περισσότερα πεδία έχουν εισαχθεί περιορισμοί NOT NULL ώστε να μην είναι δυνατό οι αντίστοιχες τιμές να τίθενται NULL. Στα Hotel_Group_ID, Hotel_ID, Room_ID, Phone_H_ID, Email_H_ID, Phone_HG_ID, Email_HG_ID έχουμε δώσει AUTO_INCREMENT έτσι ώστε κάθε φορά που εισάγεται νέο row στον εκάστοτε πίνακα το ID να αυξάνει κατά 1. Σε όποιο πεδίο ζητείται string έχουμε βάλει περιορισμό VARCHAR έως 45 χαρακτήρες. Δεν χρησιμοποιήσαμε περιορισμό CHAR διότι στις περιπτώσεις που το string είναι μικρότερο από το μέγιστο δυνατό γίνεται padding με whitespace πράγμα που δεν θέλουμε. Μια ειδική περίπτωση αποτελούν τα Street Numbers τα οποία μπορούν να είναι της μορφής "15A" και γι'αυτό ορίστηκαν ως VARCHAR. Ακόμη κωδικοί όπως IRS, Social Security Number, τηλέφωνο κλπ έχουμε δηλώσει ως VARCHAR και όχι ως INT καθώς ένας κωδικός μπορεί να αρχίζει με ψηφίο "0" και στην περίπτωση αυτή αν δινόταν ως INT τότε το "0" θα αγνοούταν π.χ. ο κωδικός "041" αν ήταν INT θα αναγνωριζόταν ως "41" ενώ ως VARCHAR συμβαίνει το επιθυμητό. Στις περιπτώσεις που δυο πεδία συνδέονται μεταξύ τους χρησιμοποιούμε τους περιορισμούς ακεραιότητας των keys. Μια τέτοια περίπτωση αποτελούν οι πίνακες hotel και hotel_room. Ο hotel έχει PRIMARY KEY το πεδίο Hotel_ID ενώ hotel_room έχει FOREIGN KEY το πεδίο αυτό ,με όνομα Hotel_ID_R. Όπου υπάρχει σχέση που περιέχει ξένο κλειδί υπάρχει και ο αντίστοιχος περιορισμός ξένου κλειδιού, έτσι ώστε να συμπληρώνεται υποχρεωτικά από την υπαρκτή τιμή του αντίστοιχου πίνακα στον οποίο είναι πρωτεύον κλειδί. Επίσης λάβαμε και κάποιους περιορισμούς με χρήση triggers. Αρχικά, για την εισαγωγή ή αλλαγή κάποιου email έχουμε ορίσει ότι πρέπει να έχει συγκεκριμένη μορφή τύπου '%_@__%.__%' αλλιώς να εμφανίζει ERROR. Επίσης, στην περίπτωση που ζητείται IRS ή Social Security Number έχουμε ορίσει να επιτρέπει μόνο 9-ψήφιους κωδικούς, αλλιώς να εμφανίζει ERROR. Επίσης, όπου βρίσκουμε Start Date και Finish Date έχουμε ορίσει να επιτρέπει μόνο στην περίπτωση που είναι $StartDate < FinishDate$. Επίσης, στο Paid της Reserves έχουμε βάλει Default Value = "No" καθώς μόλις γίνεται η κράτηση δεν γίνεται και η πληρωμή, και μόλις η κράτηση γίνει ενοικίαση και δηλωθεί το Payment Amount και το Payment Method τότε δίνεται η αντίστοιχη εντολή από τον employee μέσω ενός query στην php για αλλάγη της αντίστοιχης Paid από "No" σε "Yes".

c. Ευρετήρια-Indexes

Στην υλοποίηση μας έγινε χρήση indexes. Τα indexes φαίνονται στο σχεσιακό μας ως εξής :



Ουσιαστικά έχουμε χρησιμοποιήσει indexes ώστε να πραγματοποιήσουμε επιτάχυνση σε κάποια SELECT Queries που πραγματοποιούνται συχνά. Με αυτή τη λογική έχουμε επιλέξει τα indexes που φαίνονται παραπάνω, δηλαδή με κριτήριο αν το attribute στο οποίο δείχνουν αποτελεί σύνηθες κριτήριο αναζήτησης.

d. Σύστημα και Γλώσσες Προγραμματισμού:

Για την υλοποίηση της εφαρμογής μας χρησιμοποιήθηκαν τα ακόλουθα :

- Κατασκευή βάσης: MySQL Workbench
- Επικοινωνία μεταξύ client-server: PHP
- Client side service και γραφικά: HTML και CSS
- Δημιουργία τοπικού server: Apache HTTP Server Project
- Λειτουργικό Σύστημα: Windows
- Επιπλέον χρησιμοποιήθηκε το phpMyAdmin για περαιτέρω διαχείριση της βάσης.

e. Οδηγίες για εκ νέου ανάπτυξη της εφαρμογής από κάποιον χρήστη

Για να εγκαταστήσει κάποιος την εφαρμογή που υλοποιήσαμε θα πρέπει να ακολουθήσει τα παρακάτω βήματα:
Κατεβάζουμε την έκδοση του Apache που επιθυμούμε και την αποθηκεύουμε στη βάση του υπολογιστή μας "C:\" . Έπειτα πρέπει να ληφθούν οι MySQL server και η PHP τα οποία απόθηκεύονται στο ίδο directory με τον Apache. Μέσα από το configuration file του apache, "Apache24\conf\httpd.conf" δημιουργούμε τον τοπικό server και κάνουμε τις απαραίτητες τροποποιήσεις ώστε να συγχρονιστεί με τον MySQL server και τον PHP. Οι τροποποιήσεις είναι οι εξής:

1. Προσθέτουμε τις παρακάτω εντολές:
   LoadModule php**7**_module "c:/php/php**7**apache**2_4**.dll"
   AddHandler application/x-httpd-php .php
   # configure the path to php.ini
   PHPIniDir "C:/php"

   Οι χαρακτήρες με έντονη γραφή σχετίζονται με την αντίστοιχη έκδοση των αρχείων.

2. Κάνουμε "uncomment" την εντολή: ServerName www.example.com:00 και προσθέτουμε το όνομα στον server π.χ. localhost.

Μόλις ολοκληρώσουμε τις αλλαγές στον configuration file εγκαθιστούμε τον Apache μέσω του cmd ως διαχειριστές με την εντολή: "C:\apache24\bin\httpd -k install" . Όλα τα αρχεία που έχουμε δημιουργήσει για την εφαρμογή πρέπει να βρίσκονται στον φάκελο htdocs του Apache ώστε να έχουμε πρόσβαση σε αυτά από τον τοπικό server. Στη συνέχεια απαιτείται να κάνουμε συγκεκριμένες προσθήκες στις μεταβλητές περιβάλλοντος του συστήματος για τον Apache και την PHP. Αυτές είναι:
C:\Apache24
C:\Apache24\bin
C:\php

Τέλος χρειάζεται να εγκαταστήσουμε το MySQL Workbench για την διαχείριση των σχεσιακών βάσεων δεδομένων.
Η σύνδεση της βάσης με την PHP πραγματοποιείται μέσω του αρχείου "connect.php" που έχουμε υλοποιήσει. Εκεί εισάγεται ο εκάστοτε συνδυασμός (όνομα τοπικού server, όνομα user, κωδικός, βάση που χρειαζόμαστε)
Έχοντας πραγματοποιήσει τα παραπάνω βήματα ένας χρήστης είναι πλέον σε θέση να χρησιμοποιήσει την εφαρμογή μας.

DDL κώδικας (περιέχεται και σε άλλο αρχείο):

```
CREATE SCHEMA IF NOT EXISTS `hotels`;
USE `hotels` ;


DROP TABLE IF EXISTS `hotels`.`administrator`;

CREATE TABLE `hotels`.`administrator` (
  `Username_of_Admin` VARCHAR(45) NOT NULL,
  `Password_of_Admin` VARCHAR(45) NOT NULL,
  `Admin_ID` INT(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`Admin_ID`));


DROP TABLE IF EXISTS `hotels`.`customer`;

CREATE TABLE `hotels`.`customer` (
  `First_Name_C` VARCHAR(45) NOT NULL,
  `SSN_C` VARCHAR(9) NOT NULL UNIQUE,
  `Last_Name_C` VARCHAR(45) NOT NULL,
  `IRS_C` VARCHAR(9) NOT NULL,
  `A_Street_C` VARCHAR(45) NOT NULL,
  `A_Number_C` VARCHAR(10) NOT NULL,
  `A_Postal_Code_C` VARCHAR(45) NOT NULL,
  `A_City_C` VARCHAR(45) NOT NULL,
  `First_Registration_C` DATETIME NOT NULL DEFAULT LOCALTIME(),
  `User_C` VARCHAR(45) UNIQUE NOT NULL,
  `Pass_C` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`IRS_C`),
  INDEX `username_idx` (`User_C` ASC),
  INDEX `password_idx` (`Pass_C` ASC));


DROP VIEW IF EXISTS `hotels`.`cust`;

CREATE VIEW `hotels`.`cust`
AS SELECT First_Name_C, Last_Name_C, IRS_C, SSN_C, A_Street_C,
A_Number_C, A_Postal_Code_C, A_City_C, First_Registration_C
FROM `hotels`.`customer`;
```

```
DROP TABLE IF EXISTS `hotels`.`hotel_group`;

CREATE TABLE `hotels`.`hotel_group` (
  `A_Street_HG` VARCHAR(45) NOT NULL,
  `A_Number_HG` VARCHAR(10) NOT NULL,
  `A_Postal_Code_HG` VARCHAR(45) NOT NULL,
  `A_City_HG` VARCHAR(45) NOT NULL,
  `Hotel_Group_ID` INT(45) NOT NULL AUTO_INCREMENT,
  `Number_of_Hotels` INT(11) DEFAULT 0,
  `Name_HG` VARCHAR(45) NOT NULL UNIQUE,
  PRIMARY KEY (`Hotel_Group_ID`));


DROP TABLE IF EXISTS `hotels`.`hotel`;

CREATE TABLE `hotels`.`hotel` (
  `Stars` TINYINT(1) NOT NULL,
  `Number_of_Rooms` INT(11) DEFAULT 0,
  `Hotel_ID` INT(11) NOT NULL AUTO_INCREMENT,
  `A_Street_H` VARCHAR(45) NOT NULL,
  `A_Number_H` VARCHAR(10) NOT NULL,
  `A_Postal_Code_H` VARCHAR(45) NOT NULL,
  `A_City_H` VARCHAR(45) NULL DEFAULT NULL,
  `Hotel_Group_ID_H` INT(11) NOT NULL,
  `Name_H` VARCHAR(45) NOT NULL UNIQUE,
  PRIMARY KEY (`Hotel_ID`),
  INDEX `Parent_Hotel_Group_idx` (`Hotel_Group_ID_H` ASC),
  CONSTRAINT `Parent_Hotel_Group`
    FOREIGN KEY (`Hotel_Group_ID_H`)
    REFERENCES `hotels`.`hotel_group` (`Hotel_Group_ID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE
  );


DROP TABLE IF EXISTS `hotels`.`email_hotel`;

CREATE TABLE `hotels`.`email_hotel` (
  `Hotel_ID_email` INT(11) NOT NULL,
  `Email_H` VARCHAR(45) NOT NULL,
  `Email_H_ID` INT(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`Email_H_ID`),
  INDEX `Hotel_ID (email)` (`Hotel_ID_email` ASC),
  CONSTRAINT `Hotel_ID (email)`
    FOREIGN KEY (`Hotel_ID_email`)
    REFERENCES `hotels`.`hotel` (`Hotel_ID`)
      ON DELETE CASCADE
    ON UPDATE CASCADE
```

```
  );


DROP TABLE IF EXISTS `hotels`.`email_hotel_group`;

CREATE TABLE `hotels`.`email_hotel_group` (
  `Hotel_Group_ID_email` INT(11) NOT NULL,
  `Email_HG` VARCHAR(45) NOT NULL,
  `Email_HG_ID` INT(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`Email_HG_ID`),
  INDEX `Hotel_Group_ID (email)_idx` (`Hotel_Group_ID_email` ASC),
  CONSTRAINT `Hotel_Group_ID (email)`
    FOREIGN KEY (`Hotel_Group_ID_email`)
    REFERENCES `hotels`.`hotel_group` (`Hotel_Group_ID`)
      ON DELETE CASCADE
    ON UPDATE CASCADE
  );


DROP TABLE IF EXISTS `hotels`.`employee`;

CREATE TABLE `hotels`.`employee` (
  `First_Name_E` VARCHAR(45) NOT NULL,
  `SSN_E` CHAR(9) NOT NULL,
  `Last_Name_E` VARCHAR(45) NOT NULL,
  `IRS_E` VARCHAR(9) NOT NULL,
  `A_Street_E` VARCHAR(45) NOT NULL,
  `A_Number_E` VARCHAR(10) NOT NULL,
  `A_Postal_Code_E` VARCHAR(45) NOT NULL,
  `A_City_E` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`IRS_E`)
);


DROP TABLE IF EXISTS `hotels`.`hotel_room`;

CREATE TABLE `hotels`.`hotel_room` (
  `Room_ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Price` FLOAT NOT NULL,
  `Capacity` INT(11) NOT NULL,
  `Expandable` ENUM('None', 'Bed', 'Room', 'Both') NOT NULL,
  `Hotel_ID_R` INT(11) NOT NULL,
  `Sea_View` ENUM('Yes','No') DEFAULT 'No',
  `Balcony` ENUM('Yes','No') DEFAULT 'No',
  `Smoking` ENUM('Yes','No') DEFAULT 'No',
  `Parking` ENUM('Yes','No') DEFAULT 'No',
  `Wifi` ENUM('Yes','No') DEFAULT 'No',
  `TV` ENUM('Yes','No') DEFAULT 'No',
  `Refridgerator` ENUM('Yes','No') DEFAULT 'No',
  `Air_Conditioning` ENUM('Yes','No') DEFAULT 'No',
```

```
  `Repairs_Needed` ENUM('Yes','No') DEFAULT 'No',
  PRIMARY KEY (`Room_ID`),
  INDEX `Has_Rooms_idx` (`Hotel_ID_R` ASC),
  CONSTRAINT `Parent_Hotel`
    FOREIGN KEY (`Hotel_ID_R`)
    REFERENCES `hotels`.`hotel` (`Hotel_ID`)
      ON DELETE CASCADE
    ON UPDATE CASCADE
  );

ALTER TABLE hotel_room AUTO_INCREMENT=100;


DROP TABLE IF EXISTS `hotels`.`phone_hotel`;

CREATE TABLE `hotels`.`phone_hotel` (
  `Phone_H` VARCHAR(45) NOT NULL,
  `Phone_H_ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Hotel_ID_phone` INT(11) NOT NULL,
  PRIMARY KEY (`Phone_H_ID`),
  INDEX `Hotel_ID (phone)` (`Hotel_ID_phone` ASC),
  CONSTRAINT `Hotel_ID (phone)`
    FOREIGN KEY (`Hotel_ID_phone`)
    REFERENCES `hotels`.`hotel` (`Hotel_ID`)
      ON DELETE CASCADE
    ON UPDATE CASCADE
  );


DROP TABLE IF EXISTS `hotels`.`phone_hotel_group`;

CREATE TABLE `hotels`.`phone_hotel_group` (
  `Phone_HG` VARCHAR(45) NOT NULL,
  `Phone_HG_ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Hotel_Group_ID_phone` INT(11) NOT NULL,
  PRIMARY KEY (`Phone_HG_ID`),
  INDEX `Hotel_Group_ID (phone)` (`Hotel_Group_ID_phone` ASC),
  CONSTRAINT `Hotel_Group_ID (phone)`
    FOREIGN KEY (`Hotel_Group_ID_phone`)
    REFERENCES `hotels`.`hotel_group` (`Hotel_Group_ID`)
      ON DELETE CASCADE
    ON UPDATE CASCADE
  );


DROP TABLE IF EXISTS `hotels`.`rents`;

CREATE TABLE `hotels`.`rents` (
  `Start_Date` DATE NOT NULL,
  `Finish_Date` DATE NOT NULL,
```

```
  `Payment_Amount` FLOAT NOT NULL,
  `Payment_Method` ENUM('Cash','Card') NOT NULL,
  `IRS_C_rents` VARCHAR(9) NOT NULL,
  `IRS_E_rents` VARCHAR(9) NOT NULL,
  `Room_ID_rents` INT(11) NOT NULL,
  PRIMARY KEY (`Room_ID_rents`, `Start_Date`, `Finish_Date`),
  INDEX `Room_ID_idx` (`Room_ID_rents` ASC),
  INDEX `Start_Date_ID_idx` (`Start_Date` ASC),
  INDEX `Finish_Date_ID_idx` (`Finish_Date` ASC),
  CONSTRAINT `Customer_ID (rents)`
    FOREIGN KEY (`IRS_C_rents`)
    REFERENCES `hotels`.`customer` (`IRS_C`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `Employee_ID (rents)`
    FOREIGN KEY (`IRS_E_rents`)
    REFERENCES `hotels`.`employee` (`IRS_E`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `Room_ID (rents)`
    FOREIGN KEY (`Room_ID_rents`)
    REFERENCES `hotels`.`hotel_room` (`Room_ID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE
  );

DROP TABLE IF EXISTS `hotels`.`backup`;

CREATE TABLE `hotels`.`backup` (
  `Start_Date_back` DATE NOT NULL,
  `Finish_Date_back` DATE NOT NULL,
  `Payment_Amount_back` FLOAT NOT NULL,
  `Payment_Method_back` VARCHAR(20) NOT NULL,
  `IRS_C_rents_back` VARCHAR(9) NOT NULL,
  `IRS_E_rents_back` VARCHAR(9) NOT NULL,
  `Room_ID_rents_back` INT(11) NOT NULL,
  `Backup_ID` INT(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`Backup_ID`));


DROP TABLE IF EXISTS `hotels`.`reserves`;

CREATE TABLE `hotels`.`reserves` (
  `Start_Date` DATE NOT NULL,
  `Finish_Date` DATE NOT NULL,
  `Paid` ENUM('Yes','No') DEFAULT 'No',
  `IRS_C_reserves` VARCHAR(9) NOT NULL,
  `Room_ID_reserves` INT(11) NOT NULL,
  PRIMARY KEY (`Room_ID_reserves`,`Start_Date`,`Finish_Date`),
  INDEX `Start_Date_idx` (`Start_Date` ASC),
```

```
  INDEX `Finish_Date_idx` (`Finish_Date` ASC),
  INDEX `Room_ID_reserves_idx` (`Room_ID_reserves` ASC),
  INDEX `Paid_idx` (`Paid` ASC),
  CONSTRAINT `Customer_ID_`
    FOREIGN KEY (`IRS_C_reserves`)
    REFERENCES `hotels`.`customer` (`IRS_C`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `Room_ID_`
    FOREIGN KEY (`Room_ID_reserves`)
    REFERENCES `hotels`.`hotel_room` (`Room_ID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE
    );


DROP TABLE IF EXISTS `hotels`.`history`;

CREATE TABLE `hotels`.`history` (
  `Start_Date_hist` DATE NOT NULL,
  `Finish_Date_hist` DATE NOT NULL,
  `Paid_hist` ENUM('Yes','No') DEFAULT 'No',
  `IRS_C_hist` VARCHAR(9) NOT NULL,
  `Room_ID_hist` INT(11) NOT NULL,
  `History_ID` INT(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`History_ID`));


DROP TABLE IF EXISTS `hotels`.`works`;

CREATE TABLE `hotels`.`works` (
  `Start_Date` DATE NOT NULL,
  `Finish_Date` DATE NOT NULL,
  `Job_Position` VARCHAR(45) NOT NULL,
  `IRS_E_works` VARCHAR(9) NOT NULL,
  `Hotel_ID_works` INT(11) NOT NULL,
  `Works_ID` INT(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`Works_ID`),
  INDEX `IRS_E_idx` (`IRS_E_works` ASC),
  INDEX `Hotel_ID_idx` (`Hotel_ID_works` ASC),
  CONSTRAINT `Hotel_ID`
    FOREIGN KEY (`Hotel_ID_works`)
    REFERENCES `hotels`.`hotel` (`Hotel_ID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `IRS_E`
    FOREIGN KEY (`IRS_E_works`)
    REFERENCES `hotels`.`employee` (`IRS_E`)
    ON DELETE CASCADE
    ON UPDATE CASCADE
```

```
    );


DROP VIEW IF EXISTS hotels.`search_table`;

CREATE VIEW hotels.`search_table`
AS SELECT Room_ID, Price, Capacity, Expandable, Hotel_ID_R,
Sea_View, Balcony, Smoking, Parking, Wifi, TV, Refridgerator,
Air_Conditioning, Repairs_Needed, Stars, Hotel_ID, A_City_H, Name_H,
Hotel_Group_ID_H, Hotel_Group_ID, Name_HG FROM hotel_room
JOIN hotel ON hotel.Hotel_ID = hotel_room.Hotel_ID_R
JOIN hotel_group ON hotel_group.Hotel_Group_ID =
hotel.Hotel_Group_ID_H;


DROP VIEW IF EXISTS hotels.`employee_reservations`;

CREATE VIEW hotels.`employee_reservations`
AS SELECT Start_Date, Finish_Date, Paid, IRS_C_reserves,
Room_ID_reserves, Room_ID, Hotel_ID_R FROM reserves
JOIN hotel_room ON Room_ID = Room_ID_reserves;


DROP VIEW IF EXISTS hotels.`cities`;

CREATE VIEW hotels.`cities`
AS SELECT DISTINCT A_City_H FROM hotel;


DELIMITER |
CREATE TRIGGER `reserveshistory` AFTER INSERT ON reserves FOR EACH
ROW BEGIN
INSERT INTO history (Start_Date_hist, Finish_Date_hist, Paid_hist,
IRS_C_hist, Room_ID_hist)
VALUES (NEW.Start_Date, NEW.Finish_Date, NEW.Paid,
NEW.IRS_C_reserves, NEW.Room_ID_reserves);
END;
|
delimiter ;

delimiter |
CREATE TRIGGER `rentsbackup` AFTER INSERT ON rents FOR EACH ROW
BEGIN
INSERT INTO backup (Start_Date_back, Finish_Date_back,
IRS_C_rents_back, Room_ID_rents_back, Payment_Amount_back,
Payment_Method_back, IRS_E_rents_back)
VALUES (NEW.Start_Date, NEW.Finish_Date, NEW.IRS_C_rents,
NEW.Room_ID_rents, NEW.Payment_Amount, NEW.Payment_Method,
NEW.IRS_E_rents);
END;
```

```
|
DELIMITER ;

DELIMITER |
CREATE TRIGGER `incrementhotels` BEFORE INSERT ON hotel FOR EACH ROW
BEGIN
UPDATE hotel_group SET Number_of_Hotels = Number_of_Hotels+1 WHERE
Hotel_Group_ID=NEW.Hotel_Group_ID_H;
END;
|
DELIMITER ;

DELIMITER |
CREATE TRIGGER `decrementhotels` BEFORE DELETE ON hotel FOR EACH ROW
BEGIN
UPDATE hotel_group SET Number_of_Hotels = Number_of_Hotels-1 WHERE
Hotel_Group_ID=OLD.Hotel_Group_ID_H;
END;
|
DELIMITER ;

DELIMITER |
CREATE TRIGGER `incrementrooms` BEFORE INSERT ON hotel_room FOR EACH
ROW BEGIN
UPDATE hotel SET Number_of_Rooms = Number_of_Rooms+1 WHERE
Hotel_ID=NEW.Hotel_ID_R;
END;
|
DELIMITER ;

DELIMITER |
CREATE TRIGGER `decrementrooms` BEFORE DELETE ON hotel_room FOR EACH
ROW BEGIN
UPDATE hotel SET Number_of_Rooms = Number_of_Rooms-1 WHERE
Hotel_ID=OLD.Hotel_ID_R;
END;
|
DELIMITER ;


DELIMITER ;;
/* IRS OF EACH EMPLOYEE MUST BE A 9-DIGIT CODE ELSE ERROR*/
CREATE DEFINER=`root`@`localhost` TRIGGER `check_irs_e` BEFORE
INSERT ON `employee` FOR EACH ROW BEGIN
          IF length(NEW.IRS_E) != 9
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid IRS of employee';
          END IF;
     END ;;
```

```
CREATE  DEFINER=`root`@`localhost`  TRIGGER `check_update_irs_e`
BEFORE UPDATE ON `employee` FOR EACH ROW BEGIN
          IF length(NEW.IRS_E) != 9
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid IRS of employee';
          END IF;
     END ;;
DELIMITER ;

DELIMITER ;;
/* IRS OF EACH EMPLOYEE MUST BE A 9-DIGIT CODE ELSE ERROR*/
CREATE DEFINER=`root`@`localhost` TRIGGER `check_ssn_e` BEFORE
INSERT ON `employee` FOR EACH ROW BEGIN
          IF length(NEW.SSN_E) != 9
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid SSN of employee';
          END IF;
     END ;;
CREATE  DEFINER=`root`@`localhost`  TRIGGER `check_update_ssn_e`
BEFORE UPDATE ON `employee` FOR EACH ROW BEGIN
          IF length(NEW.SSN_E) != 9
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid SSN of employee';
          END IF;
     END ;;
DELIMITER ;


DELIMITER ;;
/* IRS OF EACH CUSTOMER MUST BE A 9-DIGIT CODE ELSE ERROR*/
CREATE DEFINER=`root`@`localhost` TRIGGER `check_irs_c` BEFORE
INSERT ON `customer` FOR EACH ROW BEGIN
          IF length(NEW.IRS_C) != 9
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid IRS of customer';
          END IF;
     END ;;
CREATE  DEFINER=`root`@`localhost`  TRIGGER `check_update_irs_c`
BEFORE UPDATE ON `customer` FOR EACH ROW BEGIN
          IF length(NEW.IRS_C) != 9
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid IRS of customer';
          END IF;
     END ;;
DELIMITER ;
```

```
DELIMITER ;;
/* IRS OF EACH CUSTOMER MUST BE A 9-DIGIT CODE ELSE ERROR*/
CREATE DEFINER=`root`@`localhost` TRIGGER `check_ssn_c` BEFORE
INSERT ON `customer` FOR EACH ROW BEGIN
          IF length(NEW.SSN_C) != 9
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid IRS of customer';
          END IF;
     END ;;
CREATE  DEFINER=`root`@`localhost`  TRIGGER `check_update_ssn_c`
BEFORE UPDATE ON `customer` FOR EACH ROW BEGIN
          IF length(NEW.SSN_C) != 9
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid IRS of customer';
          END IF;
     END ;;
DELIMITER ;


DELIMITER ;;
/* EMAIL OF EACH HOTEL GROUP MUST BE IN THE CORRECT FORM ELSE
ERROR*/
CREATE  DEFINER=`root`@`localhost`  TRIGGER `check_mail_hg` BEFORE
INSERT ON `email_hotel_group` FOR EACH ROW BEGIN
          IF NEW.Email_HG NOT LIKE '%_@__%.__%'
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid mail';
          END IF;
     END ;;

CREATE DEFINER=`root`@`localhost`  TRIGGER `check_update_mail_hg`
BEFORE UPDATE ON `email_hotel_group` FOR EACH ROW BEGIN
          IF NEW.Email_HG NOT LIKE '%_@__%.__%'
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid mail';
          END IF;
     END ;;
DELIMITER ;


DELIMITER ;;
/* EMAIL OF EACH HOTEL MUST BE IN THE CORRECT FORM ELSE ERROR*/
 CREATE DEFINER=`root`@`localhost`TRIGGER `check_mail_h` BEFORE
INSERT ON `email_hotel` FOR EACH ROW BEGIN
          IF NEW.Email_H NOT LIKE '%_@__%.__%'
```

```
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid mail';
          END IF;
     END ;;
SET SESSION
SQL_MODE="ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_
ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO";;
CREATE DEFINER=`root`@`localhost` TRIGGER `check_update_mail_h`
BEFORE UPDATE ON `email_hotel` FOR EACH ROW BEGIN
          IF NEW.Email_H NOT LIKE '%_@__%.__%'
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Invalid mail';
          END IF;
     END ;;
DELIMITER ;


DELIMITER ;;
/* START DATE < FINISH DATE ELSE ERROR */
CREATE DEFINER=`root`@`localhost` TRIGGER
`validate__insert_end_date` BEFORE INSERT ON `works` FOR EACH ROW
BEGIN
          IF (NEW.Start_Date IS NOT NULL AND NEW.Finish_date <
NEW.Start_date)
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'End date can not be prior
start date';
          END IF;
     END ;;
CREATE DEFINER=`root`@`localhost` TRIGGER
`validate__updated_end_date` BEFORE UPDATE ON `works` FOR EACH ROW
BEGIN
          IF (NEW.Finish_Date IS NOT NULL AND NEW.Finish_Date <
NEW.Start_Date)
          THEN
               SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'End date can not be prior
start date';
          END IF;
     END ;;
DELIMITER ;
```