Εθνικό Μετσόβιο Πολυτεχνείο

Βάσεις Δεδομένων

Εξαμηνιαία Εργασία

Ομάδα: 118

Γεώργιος Σουλιώτης (03114201)

Νικόλαος-Ιωάννης Τιτομιχελάκης (03114805)

Περιβάλλον Ανάπτυξης

Για την ανάπτυξη του User Interface χρησιμοποιήσαμε τον συνδυασμό των γλωσσών **PHP** και **HTML**, ενώ έγινε και σε μικρότερο βαθμό χρήση **CSS** μέσω **BootStrap**. Μέσω του προγράμματος XAMPP δημιουργήσαμε έναν διακομιστή που δέχεται αιτήματα html, ο οποίος συνδέεται με την βάση δεδομένων μας(**Mysql – MariaDB**) και επιτρέπει στον χρήστη να αλληλεπιδρά με αυτήν.
Συγκεκριμένα δίνει την δυνατότητα στον χρήστη να έχει πρόσβαση στα δεδομένα της βάσης, να τα διαχειρίζεται, ακόμα και να τα ενημερώνει, χωρίς να γνωρίζει απαραίτητα την δομή της ή να έχει γνώσεις SQL.

Πλεονεκτήματα:

1. Μέσω του Google Bootstrap που επιλέξαμε για την εμφάνιση στην html μας δίνονται πολλές επιλογές για τον τρόπο μορφοποίησης των σελίδων μας, έτσι ώστε η βάση μας να είναι όσο το δυνατόν πιο εύχρηστη, λειτουργική και ευχάριστη στον χρήστη.

2. Πληθώρα επιλογών σε templates, απεικόνισης στοιχείων σε πίνακα, μενού, φόρμες συμπλήρωσης, κουμπιά κ.α. ιδιαίτερα μέσω της χρήσης CSS.

3. Ο καθένας έχει την δυνατότητα να χρησιμοποιήσει την εφαρμογή από διαφορετικές συσκευές π.χ. μέσω κινητού τηλεφώνου, υπολογιστή, tablet χωρίς να απαιτείται κάτι παραπάνω από έναν απλό browser.

4. Μπορεί να γίνεται απομακρυσμένη σύνδεση στην εφαρμογή με κατάλληλη παραμετροποίηση του διακομιστή.

5. Εύκολη και άριστη σύνδεση με την Mysql χωρίς την απαίτηση από τον χρήστη να γνωρίζει CSS ώστε να αλλάζει από μόνος του το style της σελίδας.

6. Ευρεία χρήση στο Internet και συνεχής εξέλιξη και υποστήριξη των γλωσσών αυτών.

7. Δίνονται πολλές δυνατότητες στον χρήστη χωρίς να είναι απαραίτητες ιδιαίτερες γνώσεις προγραμματισμού είτε βάσεων δεδομένων αφού όλα γίνονται μέσω φιλικών προς τον χρήστη φορμών (forms).

Μειονέκτημα:

1. Η απομακρυσμένη σύνδεση καθιστά ευάλωτη της εφαρμογή και το σύστημα που τρέχει την ίδια την εφαρμογή σε επιθέσεις από τρίτους.

2. Δεν αποφεύγουμε επιθέσεις τύπου SQL Injection.

ΣΧΕΣΙΑΚΟ ΜΟΝΤΕΛΟ

Ανάλυση Περιορισμών και Ευρετηρίων:

Κατά την εισαγωγή οποιουδήποτε δεδομένου στην βάση απαιτούμε μέσω του user interface να συμπληρωθούν τα εν λόγω πεδία από τον χρήστη.

Για το όλα τα primary και foreign keys χρησιμοποιούμε **B TREE** ευρετήτιο.

stores: StoreID είναι το primary key της οντότητας και αποτελεί έναν αριθμό, ο οποίος είναι μοναδικός για κάθε κατάστημα. Αν κατά την εισαγωγή ενός νέου καταστήματος επιλέξουμε ίδιο StoreID η βάση δεν θα δεχτεί την καταχώρησή μας και θα μας ενημερώσει με μήνυμα error στο interface.

Κατά την διαγραφή ενός store θα πρέπει το primary key του να μην ταυτίζεται με κάποιο αντίστοιχο foreign key άλλης οντότητας, αλλιώς η βάση θα απορρίψει το αίτημα μας.

phone_store: StoreID, PhoneNumber primary key της οντότητας. Επιπλέον το StoreID είναι foreign key και αναφέρεται στο αντίστοιχο primary key της οντότητας stores. Κατά την εισαγωγή δεδομένων στην εν λόγω οντότητα, αν ο συνδυασμός που εισάγουμε υπάρχει ήδη, η βάση θα απορρίψει την ενέργειά μας και θα ενημερωθούμε αναλόγως στο interface.

e-mail.store: StoreID, E-mail primary key της οντότητας. Επιπλέον το StoreID είναι foreign key και αναφέρεται στο αντίστοιχο primary key της οντότητας stores. Κατά την εισαγωγή δεδομένων στην εν λόγω οντότητα, αν ο συνδυασμός που εισάγουμε υπάρχει ήδη, η βάση θα απορρίψει την ενέργειά μας και θα ενημερωθούμε αναλόγως στο interface.

employee: SSN_emp είναι το primary key της οντότητας. Η φυσική του σημασία είναι ο αριθμός κοινωνικής ασφάλισης (ΑΜΚΑ) του εργαζόμενου το οποίο είναι 11 νούμερα. Επιλέξαμε να χρησιμοποιήσουμε το ΑΜΚΑ επειδή για κάθε άνθρωπο είναι μοναδικό και χρησιμοποιείται σε παρόμοιες εφαρμογές.

Κατά την εισαγωγή ενός υπαλλήλου ελέγχουμε μέσω του user interface άμα το SSN_emp είναι 11 ακριβώς ψηφία ενώ σε αντίθετη περίπτωση εμφανίζεται κατάλληλο μήνυμα.

Επίσης αυτόματα από την βάση δεδομένων ελέγχεται άμα υπάρχει ήδη υπάλληλος με το ίδιο SSN_emp και απορρίπτει την καταχώρηση που επιχειρείται.

Ταυτόχρονα, το IRS_emp (ΑΦΜ), αποτελεί υποψήφιο κλειδί για τους υπαλλήλους (οπότε ορίζεται ως unique), αφού είναι μοναδικό για κάθε άνθρωπο, ενώ επιλέγεται η τιμή του ως not null. Κατά την εισαγωγή ενός υπαλλήλου ελέγχουμε μέσω του user interface άμα το IRS_emp είναι 10 ακριβώς ψηφία ενώ σε αντίθετη περίπτωση εμφανίζεται κατάλληλο μήνυμα. Επίσης αυτόματα από την βάση δεδομένων ελέγχεται άμα υπάρχει ήδη υπάλληλος με το ίδιο IRS_emp και απορρίπτει την καταχώρηση που επιχειρείται. Το ίδιο θα είναι το αποτέλεσμα αν κατά την καταχώρηση η τιμή του IRS_emp παραμείνει null.

Όμοια με το IRS_emp, ισχύουν ακριβώς τα ίδια για το IdentityNumber και για το DriverLicense.

Τέλος έχουμε τα not null γνωρίσματα LastName και FisrtName.

Κατά την διαγραφή ενός employee θα πρέπει το primary key του να μην ταυτίζεται με κάποιο αντίστοιχο foreign key άλλης οντότητας, αλλιώς η βάση θα απορρίψει το αίτημα μας.

phone_employee: SSN_emp, PhoneNumber primary key της οντότητας. Επιπλέον το SSN_emp είναι foreign key και αναφέρεται στο αντίστοιχο primary key της οντότητας employee. Κατά την εισαγωγή δεδομένων στην εν λόγω οντότητα, αν ο συνδυασμός που εισάγουμε υπάρχει ήδη, η βάση θα απορρίψει την ενέργειά μας και θα ενημερωθούμε αναλόγως στο interface.

e-mail.employee: SSN_emp, E-mail primary key της οντότητας. Επιπλέον το SSN_emp είναι foreign key και αναφέρεται στο αντίστοιχο primary key της οντότητας employee. Κατά την εισαγωγή δεδομένων στην εν λόγω οντότητα, αν ο συνδυασμός που εισάγουμε υπάρχει ήδη, η βάση θα απορρίψει την ενέργειά μας και θα ενημερωθούμε αναλόγως στο interface.

works: SSN_emp, StoreID, StartDate είναι το primary key της σχέσης works. Επιπλέον το StoreID και το SSN_emp είναι foreign keys που αναφέρονται στα primary keys των οντοτήτων stores και employee αντίστοιχα. Salary, Position γνωρίσματα με default τιμή not null. Κατά την εισαγωγή δεδομένων στη βάση αν ο συνδυασμός SSN_emp, StoreID, StartDate είναι ίδιο με κάποιον άλλον ήδη καταχωρημένο, η βάση θα απορρίψει το αίτημα μας με ανάλογο μήνυμα στο user interface. Αν κατά την εισαγωγή Salary ή Position παραμείνουν με τιμή null, τότε έχουμε το ίδιο αποτέλεσμα.

vehicle: VehicleID είναι το primary key της οντότητας και αποτελεί έναν αριθμό, ο οποίος είναι μοναδικός για κάθε όχημα. Αν κατά την εισαγωγή ενός νέου οχήματος επιλέξουμε ίδιο VehicleID η βάση δεν θα δεχτεί την καταχώρησή μας και θα μας ενημερώσει με μήνυμα error στο interface. Επιπλέον το StoreID είναι foreign key και αναφέρεται στο αντίστοιχο primary key της οντότητας stores. Ταυτόχρονα, το LicensePlate (πινακίδα οχήματος), αποτελεί υποψήφιο κλειδί για τα οχήματα (οπότε ορίζεται ως unique), αφού είναι μοναδικό για κάθε όχημα, ενώ επιλέγεται η τιμή του ως not null. Αν κατά την εισαγωγή ενός νέου οχήματος επιλέξουμε ίδιο LicensePlate η βάση δεν θα δεχτεί την καταχώρησή μας και θα μας ενημερώσει με μήνυμα error στο interface.

Κατά την διαγραφή ενός vehicle θα πρέπει το primary key του να μην ταυτίζεται με κάποιο αντίστοιχο foreign key άλλης οντότητας, αλλιώς η βάση θα απορρίψει το αίτημα μας, διότι

υπάρχει εξάρτηση του συγκεκριμένου οχήματος με κάποια άλλη καταχώρηση σε κάποια άλλη οντότητα.

damages/malfunctions: VehicleID, Damages/Malfunctions primary key του πλειότιμου γνωρίσματος, αφού είναι μοναδικός για κάθε όχημα(δεν μπορεί το ίδιο όχημα να έχει 2 φορές την ίδια ζημιά). Επιπλέον το VehicleID είναι foreign key και αναφέρεται στο αντίστοιχο primary key της οντότητας vehicle. Κατά την εισαγωγή δεδομένων στην εν λόγω πλειότιμη, αν ο συνδυασμός που εισάγουμε υπάρχει ήδη, η βάση θα απορρίψει την ενέργειά μας και θα ενημερωθούμε αναλόγως στο interface.

customers: SSN_cust είναι το primary key της οντότητας. Η φυσική του σημασία είναι ο αριθμός κοινωνικής ασφάλισης (ΑΜΚΑ) του πελάτη το οποίο είναι 11 νούμερα. Επιλέξαμε να χρησιμοποιήσουμε το ΑΜΚΑ επειδή για κάθε άνθρωπο είναι μοναδικό και χρησιμοποιείται σε παρόμοιες εφαρμογές.

Κατά την εισαγωγή ενός πελάτη ελέγχουμε μέσω του user interface άμα το SSN_cust είναι 11 ακριβώς ψηφία ενώ σε αντίθετη περίπτωση εμφανίζεται κατάλληλο μήνυμα. Επίσης αυτόματα από την βάση δεδομένων ελέγχεται άμα υπάρχει ήδη πελάτης με το ίδιο SSN_cust και απορρίπτει την καταχώρηση που επιχειρείται.

Ταυτόχρονα, το IRS_cust (ΑΦΜ), αποτελεί υποψήφιο κλειδί για τους πελάτες (οπότε ορίζεται ως unique), αφού είναι μοναδικό για κάθε άνθρωπο, ενώ επιλέγεται η τιμή του ως not null. Κατά την εισαγωγή ενός πελάτη ελέγχουμε μέσω του user interface άμα το IRS_cust είναι 10 ακριβώς ψηφία ενώ σε αντίθετη περίπτωση εμφανίζεται κατάλληλο μήνυμα. Επίσης αυτόματα από την βάση δεδομένων ελέγχεται άμα υπάρχει ήδη πελάτης με το ίδιο IRS_cust και απορρίπτει την καταχώρηση που επιχειρείται. Το ίδιο θα είναι το αποτέλεσμα αν κατά την καταχώρηση η τιμή του IRS_cust παραμείνει null.

Όμοια με το IRS_cust, ισχύουν ακριβώς τα ίδια για το DriverLicense.

Τέλος έχουμε τα not null γνωρίσματα FirstRegistration, Country και cust_Type.

Κατά την διαγραφή ενός customer θα πρέπει το primary key του να μην ταυτίζεται με κάποιο αντίστοιχο foreign key άλλης οντότητας, αλλιώς η βάση θα απορρίψει το αίτημα μας.

cust_companies: SSN_cust, Name primary key της εξειδίκευσης, αφού μια εταιρία μπορεί να έχει μόνο ένα όνομα. Επιπλέον το SSN_cust είναι foreign key και αναφέρεται στο αντίστοιχο primary key της οντότητας customers. Κατά την εισαγωγή δεδομένων στην εν λόγω εξειδίκευση, αν ο συνδυασμός που εισάγουμε υπάρχει ήδη, η βάση θα απορρίψει την ενέργειά μας και θα ενημερωθούμε αναλόγως στο interface.

residents_abroad_docs: SSN_cust, Documents primary key της εξειδίκευσης, αφού ένας πελάτης εξωτερικού μπορεί να υποβάλλει μόνο μια φορά το κάθε έγγραφο. Επιπλέον το SSN_cust είναι foreign key και αναφέρεται στο αντίστοιχο primary key της οντότητας customers. Κατά την εισαγωγή δεδομένων στην εν λόγω εξειδίκευση, αν ο συνδυασμός που εισάγουμε υπάρχει ήδη, η βάση θα απορρίψει την ενέργειά μας και θα ενημερωθούμε αναλόγως στο interface.

rents: RentID είναι το primary key της οντότητας και αποτελεί έναν αριθμό, ο οποίος είναι μοναδικός για κάθε ενοικίαση. Αν κατά την εισαγωγή μιας νέας ενοικίασης επιλέξουμε ίδιο RentID η βάση δεν θα δεχτεί την καταχώρησή μας και θα μας ενημερώσει με μήνυμα error

στο interface. Επιπλέον το SSN_emp, SSN_cust και VehicleID είναι foreign keys και αναφέρονται στα primary keys των οντοτήτων employee, customers και vehicle αντίστοιχα. Ταυτόχρονα έχουμε τα not null γνωρίσματα StartDate, StartLocation και EmployeeHand, για τα οποία αν κατά την εισαγωγή μιας νέας ενοικίασης παραμείνουν null, η βάση θα απορρίψει την εισαγωγή, ενημερώνοντας μας στο interface με error.

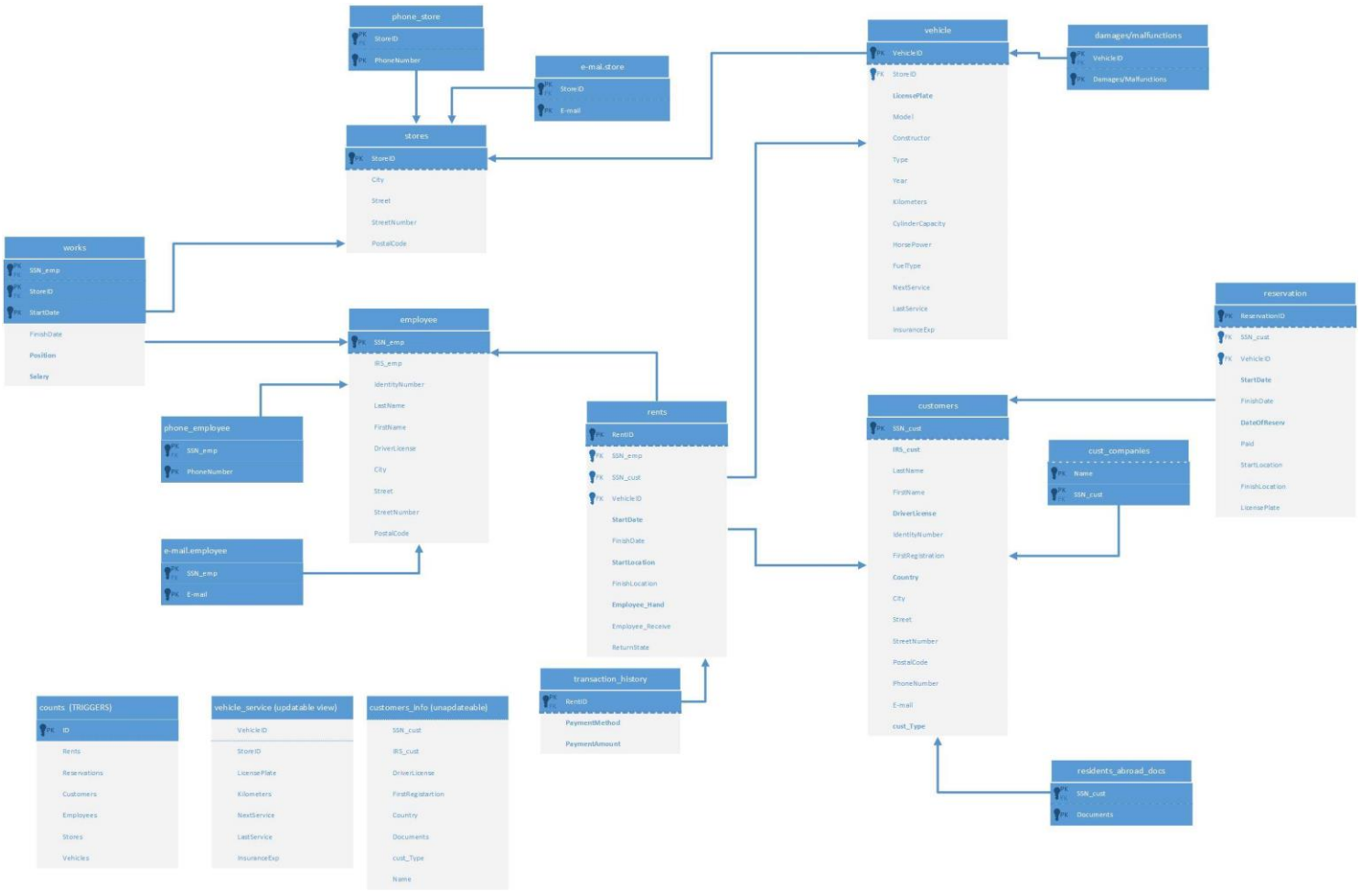
Κατά την διαγραφή ενός rent θα πρέπει το primary key του να μην ταυτίζεται με κάποιο αντίστοιχο foreign key άλλης οντότητας, αλλιώς η βάση θα απορρίψει το αίτημα μας.

transaction_history: RentID είναι primary key και ταυτόχρονα foreign key που αναφέρεται στο αντίστοιχο primary key της οντότητας rents. Κατά την εισαγωγή δεδομένων στην εν λόγω οντότητα, αν εισάγουμε transaction με το ίδιο RentID, η βάση θα απορρίψει την ενέργειά μας και θα ενημερωθούμε αναλόγως στο interface. Επιπλέον έχουμε τα by deafault not null γνωρίσματα PaymentMethod και PaymentAmount, για τα οποία αν κατά την εισαγωγή παραμείνουν null, θα έχουμε το ίδιο αποτέλεσμα.

reservation: ReservationID είναι το primary key της οντότητας και αποτελεί έναν αριθμό, ο οποίος είναι μοναδικός για κάθε κράτηση. Αν κατά την εισαγωγή μιας νέας κράτησης επιλέξουμε ίδιο ReservationID η βάση δεν θα δεχτεί την καταχώρησή μας και θα μας ενημερώσει με μήνυμα error στο interface. Επιπλέον το SSN_cust και VehicleID είναι foreign keys και αναφέρονται στα primary keys των οντοτήτων customers και vehicle αντίστοιχα. Ταυτόχρονα έχουμε τα not null γνωρίσματα StartDate και DateOfReserv, για τα οποία αν κατά την εισαγωγή μιας νέας κράτησης παραμείνουν null, η βάση θα απορρίψει την εισαγωγή, ενημερώνοντας μας στο interface με error.

Κατά την διαγραφή μιας κράτησης θα πρέπει το primary key της να μην ταυτίζεται με κάποιο αντίστοιχο foreign key άλλης οντότητας, αλλιώς η βάση θα απορρίψει το αίτημα μας.

Το UML που χρησιμοποιήθηκε για τη βάση φαίνεται παρακάτω:



Κατασκευή Βάσης:

```
CREATE DATABASE  IF NOT EXISTS `new_car_system1fixed` DEFAULT CHARACTER SET utf8
USE `new_car_system1fixed`;
```

Δημιουργία stores (καταστημάτων):

```
CREATE TABLE `stores` (
  `StoreID` int(2) NOT NULL,
  `City` varchar(60) DEFAULT NULL,
  `Street` varchar(60) DEFAULT NULL,
  `StreetNumber` int(3) DEFAULT NULL,
  `PostalCode` int(10) DEFAULT NULL,
  PRIMARY KEY (`StoreID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Δημιουργία phone_store (τηλέφωνο καταστήματος-πλειότιμο):

```
CREATE TABLE `phone_store` (
  `StoreID` int(2) NOT NULL,
  `PhoneNumber` bigint(14) NOT NULL,
  PRIMARY KEY (`StoreID`,`PhoneNumber`),
  CONSTRAINT `con2` FOREIGN KEY (`StoreID`) REFERENCES `stores` (`StoreID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Δημιουργία e-mail.store (e-mail καταστήματος-πλειότιμο):

```
CREATE TABLE `e-mail.store` (
  `StoreID` int(2) NOT NULL AUTO_INCREMENT,
  `E-mail` varchar(80) NOT NULL,
  PRIMARY KEY (`StoreID`,`E-mail`),
  CONSTRAINT `con1` FOREIGN KEY (`StoreID`) REFERENCES `stores` (`StoreID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;
```

Δημιουργία employee (εργαζόμενοι):

```
CREATE TABLE `employee` (
  `SSN_emp` bigint(11) NOT NULL,
  `IRS_emp` int(10) NOT NULL,
  `IdentityNumber` varchar(45) NOT NULL,
  `LastName` varchar(45) NOT NULL,
  `FirstName` varchar(45) NOT NULL,
  `DriverLicense` int(10) NOT NULL,
  `City` varchar(60) DEFAULT NULL,
  `Street` varchar(60) DEFAULT NULL,
  `StreetNumber` int(3) DEFAULT NULL,
  `PostalCode` int(10) DEFAULT NULL,
  PRIMARY KEY (`SSN_emp`),
  UNIQUE KEY `IRS_emp_UNIQUE` (`IRS_emp`),
  UNIQUE KEY `IdentityNumber_UNIQUE` (`IdentityNumber`),
  UNIQUE KEY `DriverLicense_UNIQUE` (`DriverLicense`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Δημιουργία phone_employee (τηλέφωνο υπαλλήλου-πλειότιμο):

```
CREATE TABLE `phone_employee` (
  `SSN_emp` bigint(11) NOT NULL,
  `PhoneNumber` bigint(10) NOT NULL,
  PRIMARY KEY (`SSN_emp`,`PhoneNumber`),
  CONSTRAINT `con13` FOREIGN KEY (`SSN_emp`) REFERENCES `employee` (`SSN_emp`) ON
DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Δημιουργία e-mail.employee (e-mail υπαλλήλου-πλειότιμο):

```
CREATE TABLE `e-mail.employee` (
  `SSN_emp` bigint(11) NOT NULL,
```

`E-mail` varchar(80) **NOT NULL**,

PRIMARY KEY (`SSN_emp`,`E-mail`),

CONSTRAINT `con15` FOREIGN KEY (`SSN_emp`) **REFERENCES** `employee` (`SSN_emp`) **ON DELETE NO ACTION ON UPDATE NO ACTION**

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

<u>Δημιουργία works (σχέση εργασίας μεταξύ υπαλλήλου και καταστήματος):</u>

CREATE TABLE `works` (

`SSN_emp` bigint(11) **NOT NULL**,

`StoreID` int(2) **NOT NULL**,

`StartDate` date **NOT NULL**,

`FinishDate` date **DEFAULT NULL**,

`Position` varchar(45) **NOT NULL**,

`Salary` double **NOT NULL**,

PRIMARY KEY (`SSN_emp`,`StoreID`,`StartDate`),

KEY `con16_idx` (`StoreID`),

CONSTRAINT `con16` FOREIGN KEY (`StoreID`) **REFERENCES** `stores` (`StoreID`) **ON DELETE NO ACTION ON UPDATE NO ACTION**,

CONSTRAINT `con17` FOREIGN KEY (`SSN_emp`) **REFERENCES** `employee` (`SSN_emp`) **ON DELETE NO ACTION ON UPDATE NO ACTION**

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

<u>Δημιουργία vehicle (οχήματα):</u>

CREATE TABLE `vehicle` (

`VehicleID` int(4) **NOT NULL**,

`StoreID` int(2) **NOT NULL**,

`LicensePlate` varchar(10) **NOT NULL**,

`Model` varchar(45) **DEFAULT NULL**,

`Constructor` varchar(45) **DEFAULT NULL**,

`Type` varchar(45) **DEFAULT NULL**,

`Year` year(4) **DEFAULT NULL**,

`Kilometers` int(6) **DEFAULT NULL**,

```
  `CylinderCapacity` int(5) DEFAULT NULL,
  `HorsePower` int(5) DEFAULT NULL,
  `FuelType` varchar(15) DEFAULT NULL,
  `NextService` date DEFAULT NULL,
  `LastService` date DEFAULT NULL,
  `InsuranceExp` date DEFAULT NULL,
  PRIMARY KEY (`VehicleID`),
  UNIQUE KEY `LicensePlate_UNIQUE` (`LicensePlate`),
  KEY `con4_idx` (`StoreID`),
  CONSTRAINT `con4` FOREIGN KEY (`StoreID`) REFERENCES `stores` (`StoreID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Δημιουργία damages/malfunctions (ζημιές-δυσλειτουργίες οχημάτων):

```
CREATE TABLE `damages/malfunctions` (
  `VehicleID` int(4) NOT NULL,
  `Damages/Malfunctions` varchar(100) NOT NULL,
  PRIMARY KEY (`VehicleID`,`Damages/Malfunctions`),
  CONSTRAINT `con5` FOREIGN KEY (`VehicleID`) REFERENCES `vehicle` (`VehicleID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Δημιουργία customers (πελάτες):

```
CREATE TABLE `customers` (
  `SSN_cust` bigint(11) NOT NULL,
  `IRS_cust` int(10) NOT NULL,
  `LastName` varchar(45) DEFAULT NULL,
  `FirstName` varchar(45) DEFAULT NULL,
```

```
  `DriverLicense` int(11) NOT NULL,
  `IdentityNumber` varchar(10) DEFAULT NULL,
  `FirstRegistration` date NOT NULL,
  `Country` varchar(45) NOT NULL,
  `City` varchar(60) DEFAULT NULL,
  `Street` varchar(60) DEFAULT NULL,
  `StreetNumber` int(3) DEFAULT NULL,
  `PostalCode` int(10) DEFAULT NULL,
  `PhoneNumber` bigint(10) DEFAULT NULL,
  `E-mail` varchar(80) DEFAULT NULL,
  `cust_Type` varchar(15) NOT NULL,
  PRIMARY KEY (`SSN_cust`),
  UNIQUE KEY `IRS_cust_UNIQUE` (`IRS_cust`),
  UNIQUE KEY `DriverLicense_UNIQUE` (`DriverLicense`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Δημιουργία residents_abroad_docs (πελάτες εξωτερικού με τα απαραίτητα έγγραφα-πλειότιμο):

```
CREATE TABLE `residents_abroad_docs` (
  `SSN_cust` bigint(11) NOT NULL,
  `Documents` varchar(45) NOT NULL,
  PRIMARY KEY (`SSN_cust`,`Documents`),
  CONSTRAINT `con8` FOREIGN KEY (`SSN_cust`) REFERENCES `customers` (`SSN_cust`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Δημιουργία cust_companies (οι πελάτες που νοικιάζουν στο όνομα της εταιρίας και τα ονόματα των εταιριών):

```
CREATE TABLE `cust_companies` (
  `SSN_cust` bigint(11) NOT NULL,
  `Name` varchar(45) NOT NULL,
  PRIMARY KEY (`SSN_cust`,`Name`),
  CONSTRAINT `con7` FOREIGN KEY (`SSN_cust`) REFERENCES `customers` (`SSN_cust`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Δημιουργία reservation (κρατήσεις):

```
CREATE TABLE `reservation` (
  `ReservationID` int(10) NOT NULL,
  `SSN_cust` bigint(11) NOT NULL,
  `VehicleID` int(4) NOT NULL,
  `StartDate` date NOT NULL,
  `FinishDate` date DEFAULT NULL,
  `DateOfReserv` date NOT NULL,
  `Paid` varchar(3) DEFAULT NULL,
  `StartLocation` varchar(45) DEFAULT NULL,
  `FinishLocation` varchar(45) DEFAULT NULL,
  `LicensePlate` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`ReservationID`),
  KEY `con10_idx` (`SSN_cust`),
  KEY `con11_idx` (`VehicleID`),
  CONSTRAINT `con10` FOREIGN KEY (`SSN_cust`) REFERENCES `customers` (`SSN_cust`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `con11` FOREIGN KEY (`VehicleID`) REFERENCES `vehicle` (`VehicleID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Δημιουργία rents (ενοικιάσεις):

```
CREATE TABLE `rents` (
  `RentID` int(10) NOT NULL,
  `SSN_emp` bigint(11) NOT NULL,
  `SSN_cust` bigint(11) NOT NULL,
  `VehicleID` int(4) NOT NULL,
  `StartDate` date NOT NULL,
  `FinishDate` date DEFAULT NULL,
  `StartLocation` varchar(45) NOT NULL,
  `FinishLocation` varchar(45) DEFAULT NULL,
  `Employee_Hand` bigint(11) NOT NULL,
  `Employee_Receive` bigint(11) DEFAULT NULL,
  `ReturnState` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`RentID`),
  KEY `con20_idx` (`SSN_cust`),
  KEY `con21_idx` (`VehicleID`),
  KEY `con22_idx` (`SSN_emp`),
  CONSTRAINT `con20` FOREIGN KEY (`SSN_cust`) REFERENCES `customers` (`SSN_cust`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `con21` FOREIGN KEY (`VehicleID`) REFERENCES `vehicle` (`VehicleID`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `con22` FOREIGN KEY (`SSN_emp`) REFERENCES `employee` (`SSN_emp`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Δημιουργία transaction_history (ιστορικό συναλλαγών):

```
CREATE TABLE `transaction_history` (
  `RentID` int(10) NOT NULL,
  `PaymentMethod` varchar(45) NOT NULL,
  `PaymentAmount` double NOT NULL,
  PRIMARY KEY (`RentID`),
```

```
  CONSTRAINT `con24` FOREIGN KEY (`RentID`) REFERENCES `rents` (`RentID`) ON DELETE
NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

QUERIES

1.
```
SELECT  `vehicle`.`StoreID` AS `StoreID`, `stores`.`City` AS `City`, `vehicle`.`Type` AS `Type`,
    COUNT(`vehicle`.`VehicleID`) AS `vcount1`
  FROM
    (`vehicle`
    LEFT JOIN `stores` ON ((`stores`.`StoreID` = `vehicle`.`StoreID`)))
  GROUP BY `vehicle`.`Type` , `stores`.`City`
```

Το παραπάνω ερώτημα μας ενημερώνει για τον αριθμό των οχημάτων ανά τύπο οχήματος που ανήκει σε κάθε ένα από τα 5 καταστήματα της επιχείρισης. Αποτέλει μια χρήσιμη ενέργεια, ώστε να υπάρχει καλύτερος έλεγχος των οχημάτων.

(aggregate, join, group by)

2.
```
SELECT
    `rents`.`RentID` AS `RentID`,
    `customers`.`Country` AS `Country`,
    `customers`.`SSN_cust` AS `SSN_cust`,
    `transaction_history`.`PaymentMethod` AS `PaymentMethod`,
    `transaction_history`.`PaymentAmount` AS `PaymentAmount`
  FROM
    ((`rents`
    LEFT JOIN `customers` ON ((`rents`.`SSN_cust` = `customers`.`SSN_cust`)))
    LEFT JOIN `transaction_history` ON ((`rents`.`RentID` = `transaction_history`.`RentID`)))
  WHERE
    (NOT ((`customers`.`Country` LIKE 'Greece')))
```

Το παραπάνω ερώτημα μας ενημερώνει για τις συναλλαγές (transactions) που έγιναν με πελάτες του εξωτερικού και μας παρέχει χρήσιμες πληροφορίες γι’ αυτές (χώρα πελάτη, τρόπος πληρωμής, ποσό).

(join, where, not, like)

3. **SELECT**

`stores`.`StoreID` **AS** `StoreID`,

`rents`.`StartLocation` **AS** `store_city`,

SUM(`transaction_history`.`PaymentAmount`) **AS** `income`

FROM

((`rents`

LEFT JOIN `stores` **ON** ((`rents`.`StartLocation` = `stores`.`City`)))

LEFT JOIN `transaction_history` **ON** ((`rents`.`RentID` = `transaction_history`.`RentID`)))

GROUP BY `rents`.`StartLocation`

ORDER BY `income` **DESC**

Συνολικές εισπράξεις του κάθε καταστήματος από την έναρξη λειτουργίας του ως τώρα, ταξινομημένες σε φθίνουσα σειρά. Μας παρέχει πληροφορίες για την επιτυχία του κάθε καταστήματος, για το κατάστημα που παρέχει τις μεγαλύτερες εισπράξεις. Ερώτημα που βοηθά στην διαχείριση των οικονομικών.

(join, group by, order by…desc)

4. **SELECT**

`employee`.`SSN_emp` **AS** `SSN_emp`,

`works`.`StoreID` **AS** `StoreID`,

`employee`.`LastName` **AS** `LastName`,

`employee`.`FirstName` **AS** `FirstName`,

`works`.`Position` **AS** `Position`,

`works`.`Salary` **AS** `Salary`,

`works`.`FinishDate` **AS** `FinishDate`

FROM

(`employee`

LEFT JOIN `works` **ON** ((`employee`.`SSN_emp` = `works`.`SSN_emp`)))

WHERE

ISNULL(`works`.`FinishDate`)

HAVING (`works`.`Salary` > 1000)

Μας ενημερώνει για τους υπαλλήλους που εργάζονται τώρα στην επιχείριση και έχουν μισθό παραπάνω από 1000 ευρώ, δηλαδή εξάγουμε πληροφορίες για τους θεωρητικά καλοπληρωμένους υπαλλήλους της επιχείρισης. Η σημασία του ερωτήματος είναι οικονομικής φύσεως και εξυπηρετεί την καλύτερη διαχείριση των οικονομικών,

(join, where, having)

5. **SELECT**

    `rents`.`VehicleID` **AS** `VehicleID`,

    COUNT(`rents`.`RentID`) **AS** `COUNT`

  FROM

    `rents`

  GROUP BY `rents`.`VehicleID`

  HAVING (**COUNT**(`COUNT`) <= 3)

Το παρόν ερώτημα μας πληροφορεί για τα οχήματα που δεν έχουν ζήτηση, δηλαδή για αυτά που έχουν νοικιαστεί το πολύ 3 φορές. Τις παρούσες πληροφορίες μπορεί να λάβει υπόψη της η διοίκηση της επιχείρισης, όταν θα αγοράσει το επόμενο όχημα, ώστε στατιστικά να αποφύγει την αγορά ενός μη κερδοφόρου οχήματος.

(aggregate, group by..having)

6. **SELECT**

    `customers`.`SSN_cust` **AS** `black_list`,

    `customers`.`LastName` **AS** `LastName`,

    `customers`.`FirstName` **AS** `FirstName`

  FROM

    `customers`

  WHERE

    `customers`.`SSN_cust` **IN** (**SELECT**

        `rents`.`SSN_cust`

      FROM

        `rents`

      WHERE

        (`rents`.`ReturnState` **LIKE** 'BAD'))

Η παραπάνω ενέργεια μας παρέχει τους πελάτες που επέστρεψαν το όχημα που ενοικίασαν σε κακή κατάσταση, δηλαδή μας παρέχει την black list των πελατών, ώστε η επιχείριση να αποφύγει μελλοντική συνεργασία με τους ίδιους πελάτες και να αποφύγει τυχόν ζημιές στα οχήματα.

(nested query)

7. **SELECT**

    `rents`.`RentID` **AS** `RentID`,

    `rents`.`SSN_cust` **AS** `SSN_cust`,

    `transaction_history`.`PaymentMethod` **AS** `PaymentMethod`,

    `transaction_history`.`PaymentAmount` **AS** `PaymentAmount`

  FROM

    (`rents`

    LEFT JOIN `transaction_history` **ON** ((`rents`.`RentID` = `transaction_history`.`RentID`)))

  WHERE

    ISNULL(`rents`.`Employee_Receive`)

  ORDER BY `transaction_history`.`PaymentAmount` **DESC**

Το συγκεκριμένο ερώτημα μας ενημερώνει για τα transactions(σε φθίνουσα σειρά-από το περισσότερο στο λιγότερο σημαντικό) των πελατών που αντιστοιχούν σε ενοικιάσεις οι οποίες ακόμα τρέχουν, δηλαδή το όχημα δεν έχει επιστραφεί ακόμα. Χρήσιμο query για καλύτερη εποπτεία των τρεχόντων συναλλαγών, σε ζητήματα οικονομικά και νομικά.

(join, order by…desc)

8. **SELECT**

    `reservation`.`VehicleID` **AS** `VehicleID`,

    `reservation`.`StartDate` **AS** `StartDate`,

    `reservation`.`FinishDate` **AS** `FinishDate`

  FROM

    `reservation`

  WHERE

    (`reservation`.`StartDate` > **NOW**())

  UNION SELECT

    `rents`.`VehicleID` **AS** `VehicleID`,

    `rents`.`StartDate` **AS** `StartDate`,

```
    `rents`.`FinishDate` AS `FinishDate`
  FROM
    `rents`
  WHERE
    (ISNULL(`rents`.`FinishDate`)
      OR (`rents`.`FinishDate` > NOW()))
```

Το ερώτημα αυτό μας πληροφορεί για τα οχήματα που είναι νοικιασμένα τώρα και για τα οχήματα που έχουν δεσμευτεί από κρατήσεις στο μέλλον, σε πραγματικό χρόνο, λόγω της συνάρτησης NOW() που υποστηρίζει η sql και η οποία επιστρέφει την παρούσα ημερομηνία δυναμικά. Το ερώτημα αυτό είναι ζωτικής σημασίας για την επιχείριση, διότι πριν γίνει οποιαδήποτε νέα ενοικίαση ή κράτηση πρέπει να ελέγχεται αν αυτές έρχονται σε σύγκρουση με άλλες, από την στιγμή που το όχημα θα έχει δεσμευτεί. Το παρόν ερώτημα λοιπόν έχει να κάνει με την αποτελεσματική και υπεύθυνη παροχή υπηρεσιών των καταστημάτων.

(WHERE, UNION)

9.
```
SELECT
    `customers`.`SSN_cust` AS `SSN_cust`,
    `customers`.`LastName` AS `LastName`,
    `customers`.`FirstName` AS `FirstName`,
    COUNT(`rents`.`RentID`) AS `sum_rents`
  FROM
    (`customers`
    LEFT JOIN `rents` ON ((`customers`.`SSN_cust` = `rents`.`SSN_cust`)))
  WHERE
    (NOT ((`rents`.`ReturnState` LIKE 'BAD')))
  GROUP BY `rents`.`SSN_cust`
  HAVING (COUNT(`sum_rents`) >= 2)
```

Το παρόν ερώτημα μας παρέχει τους «καλούς» πελάτες της επιχείρισης, δηλαδή αυτούς που έχουν νοικιάσει όχημα τουλάχιστον 2 φορές από οποιοδήποτε κατάστημα της επιχείρισης και ποτέ δεν το επέστρεψαν σε κακή κατάσταση. Το παρόν ερώτημα είναι χρήσιμο, διότι βοηθά την επιχείριση να γνώριζει πώς θα πρέπει να αντιμετωπίσει έναν πελάτη, αν για παράδειγμα θέλει να κάνει μια φιλική έκπτωση σε έναν καλό πελάτη ως προτροπή για συνέχιση της συνεργασίας τους και στο μέλλον.

(join, like, group by…having)

10. **SELECT**

    `vehicle`.`Type` **AS** `Type`,

    AVG(`vehicle`.`Kilometers`) **AS** `avr_kilometers`

  FROM

    `vehicle`

  GROUP BY `vehicle`.`Type`

Μας παρέχει τον μέσο όρο των χιλιομέτρων ανά τύπο οχήματος. Βοηθά την επιχείριση να εξάγει συμπεράσματα για τους διάφορους τύπους οχημάτων που κατέχει, για παράδειγμα ανά πόσα χιλιόμετρα ένας τύπος οχήματος χρειάζεται service.

(aggregate, group by)

11. **SELECT**

    `stores`.`StoreID` **AS** `StoreID`,

    `rents`.`StartLocation` **AS** `City`,

    SUM(`transaction_history`.`PaymentAmount`) **AS** `anual_income`,

    YEAR(`rents`.`StartDate`) **AS** `YEAR`

  FROM

    ((`rents`

    LEFT JOIN `stores` **ON** ((`rents`.`StartLocation` = `stores`.`City`)))

    LEFT JOIN `transaction_history` **ON** ((`rents`.`RentID` = `transaction_history`.`RentID`)))

  GROUP BY `rents`.`StartLocation` , **YEAR**(`rents`.`StartDate`)

Πληροφορεί τον χρήστη για το ετήσιο εισόδημα κάθε καταστήματος από την έναρξη λειτουργίας του. Χρήσιμο ερώτημα από οικονομικής άποψης, για εξαγωγή ανάλογων συμπερασμάτων από την διοίκηση. Τα δεδομένα που επιστρέφει το παρόν ερώτημα είναι ομαδοποιημένα με βάση το κάθε κατάστημα.

(aggregate, join, group by)

12. **SELECT**

    `vehicle`.`VehicleID` **AS** `VehicleID`,

    `vehicle`.`LicensePlate` **AS** `LicensePlate`,

```
    `vehicle`.`NextService` AS `NextService`,
    `vehicle`.`InsuranceExp` AS `InsuranceExp`
FROM
    `vehicle`
HAVING (((`vehicle`.`NextService` > NOW())
    AND (`vehicle`.`NextService` < (NOW() + INTERVAL 30 DAY)))
    OR ((`vehicle`.`InsuranceExp` > NOW())
    AND (`vehicle`.`InsuranceExp` < (NOW() + INTERVAL 30 DAY))))
```

Το ερώτημα αύτο πληροφορεί τον χρήστη για τα οχήματα που θα χρειαστούν service ή ανανέωση της ασφάλειάς τους, μέσα στις επόμενες 30 μέρες σε σχέση πάντα με την παρούσα χρονική στιγμή (σε πραγματικό χρόνο). Η χρησιμότητα του ερωτήματος έγγειται στο γεγόνος ότι βοηθά την επιχείριση στον προγραμματισμό των εκκρεμοτήτων που αφορούν τα οχήματα στο άμεσο μέλλον.

(HAVING)

VIEWS

Not updateable view

```
CREATE VIEW `customers_info` AS
ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
  SELECT
    `customers`.`SSN_cust` AS `SSN_cust`,
    `customers`.`IRS_cust` AS `IRS_cust`,
    `customers`.`DriverLicense` AS `DriverLicense`,
    `customers`.`FirstRegistration` AS `FirstRegistration`,
    `customers`.`Country` AS `Country`,
    `residents_abroad_docs`.`Documents` AS `Documents`,
    `customers`.`cust_Type` AS `cust_Type`,
    `cust_companies`.`Name` AS `Name`
  FROM
    ((`customers`
    LEFT JOIN `residents_abroad_docs` ON ((`customers`.`SSN_cust` =
`residents_abroad_docs`.`SSN_cust`)))
    LEFT JOIN `cust_companies` ON ((`customers`.`SSN_cust` =
`cust_companies`.`SSN_cust`)))
  ORDER BY `customers`.`SSN_cust`
```

Η συγκεκριμένη όψη μπορεί να χρησιμοποιηθεί για μια πιο περιορισμένη διαχείριση της βάσης, αφού δεν δίνεται η δυνατότητα στον χρήστη να επεξεργαστεί δεδομένα είτε να δει πιο προσωπικές πληροφορίες(πχ την διεύθυνση και τον αριθμό κατοικίας) ενός πελάτη. Επιπλέον γίνεται μια καλύτερη παρουσίαση χρήσιμων δεδομένων σε **ένα** table, χωρίς να είναι απαραίτητο ο υπάλληλος να ανοίγει διαφορετικά ευρετήρια για να πάρει τις πληροφορίες που χρειάζεται (πχ όνομα εταιρίας αν ο πελάτης είναι εταιρία ή έγγραφα που έχει υποβάλλει ο πελάτης αν είναι κάτοικος εξωτερικού).

Updateable view

```
CREATE VIEW `vehicle_service` AS
ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
  SELECT
    `vehicle`.`VehicleID` AS `VehicleID`,
    `vehicle`.`StoreID` AS `StoreID`,
    `vehicle`.`LicensePlate` AS `LicensePlate`,
    `vehicle`.`Kilometers` AS `Kilometers`,
    `vehicle`.`NextService` AS `NextService`,
    `vehicle`.`LastService` AS `LastService`,
    `vehicle`.`InsuranceExp` AS `InsuranceExp`
  FROM
    `vehicle`
```

Η συγκεκριμένη όψη χρησιμοποιείται για περιορισμένη επεξεργασία δεδομένων ενός οχήματος (πχ ενημέρωση χιλιομέτρων, ημερομηνία λήξης ασφάλειας κλπ). Αυτό εξυπηρετεί τον χρήστη, ο οποίος θέλει να ενημερώσει μόνο τα συγκεκριμένα δεδομένα ή να πληροφορηθεί γι' αυτά. Φυσικά, τα πεδία που μπορούν να ενημερώνονται στην παραπάνω όψη έχουν επιλεχτεί έτσι ώστε να μην παραβιάζεται η αναφορική ακεραιότητα της βάσης

(πχ όταν εισάγουμε νέο όχημα στην όψη καμία τιμή NOT NULL του table των οχημάτων δεν παραμένει null, ενώ ταυτόχρονα τηρούνται οι περιορισμοί που αφορούν τα foreign keys).

TRIGGERS

Για τα triggers δημιουργήσαμε μια καινούργια οντότητα η οποία περιέχει τον αριθμό των στοιχείων που περιέχουν οι βασικές μας οντότητες στην βάση. Ουσιαστικά δημιουργήσαμε μία οντότητα η οποία κάνει συνεχές count στις άλλες βασικές οντότητες. Για την δημιουργία της οντότητας:

```
CREATE TABLE `counts` (
  `ID` int(11) NOT NULL,
  `Rents` int(11) DEFAULT NULL,
```

```
  `Reservations` int(11) DEFAULT NULL,
  `Customers` int(11) DEFAULT NULL,
  `Employees` int(11) DEFAULT NULL,
  `Stores` int(11) DEFAULT NULL,
  `Vehicles` int(11) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Για κάθε βασική οντότητα της βάσης μας έχουμε δημιουργήσει ένα trigger που παρακολουθεί την είσοδο και την διαγραφή κάθε στοιχείου ώστε να ενημερώνεται άμεσα ο αντίστοιχο count. Επομένως έχουμε δημιουργήσει 5x2 triggers που είναι όμως είναι ίδια μεταξύ τους. Ένα σετ από αυτά είναι:

```
CREATE DEFINER=`root`@`localhost` TRIGGER `addStore` BEFORE INSERT ON `stores` FOR EACH ROW
BEGIN
UPDATE counts SET Stores = Stores+1 WHERE ID=1;
END

CREATE DEFINER=`root`@`localhost` TRIGGER `deleteStore` AFTER DELETE ON `stores` FOR EACH ROW
BEGIN
UPDATE counts SET Stores = Stores-1 WHERE ID=1;
END
```

Τα συγκεκριμένα triggers μας ενημερώνουν για τον ακριβή αριθμό των στοιχείων που έχουμε στην βάση μας ώστε να έχουμε μια πιο γενική εικόνα του μεγέθους της πληροφορίας που διαθέτουμε στην βάση μας.

Η σελίδα με τα triggers της βάσης φαίνεται παρακάτω:

Τέλος, αναφέρεται ότι τα ενδεικτικά στοιχεία που έχουν καταχωρηθεί στην βάση είναι «έξυπνα», δηλαδή έχουν λογική σημασία, ώστε να εξυπηρετούνται όλα τα ερωτήματα κατά το τεστάρισμα της βάσης.