

# Project Travel Agency Final

**-Κατσαρός Ανδρέας (1084522)**  
**-Ποταμιάνος Άγγελος Νικόλαος**  
**(1084537)**  
**-Τζωρτζάκης Γρηγόρης (1084538)**

ΑΚΑΔΗΜΑΙΚΟ ΕΤΟΣ: 2022-2023

—

ΕΡΓΑΣΤΗΡΙΟ ΒΑΣΕΩΝ  
ΔΕΔΟΜΕΝΩΝ

—

Κα ΒΟΓΙΑΤΖΑΚΗ,  
Κος ΒΑΣΙΛΟΠΟΥΛΟΣ

---

**ΕΡΩΤΗΜΑΤΑ:**

**3.1.1 Προπαρασκευαστική Φάση:**

Αρχικά, παραθέτουμε μαζί με την αναφορά για την ευκολία στην ανάγνωση την βάση δεδομένων επισυναπτόμενη με όνομα: travel\_agency.sql .Η βάση αυτή, περιέχει τα αρχικά tables(branch,worker,guide κ.λ.π.) που απαιτούνται για την υλοποίηση του project σε συνδυασμό με τα επόμενα tables που έχουν δημιουργηθεί λόγω των ερωτημάτων της επόμενης φάσης.

Εξηγούμε ενδεικτικά το table: trip για να παραθέσουμε την λογική με την οποία λειτουργήσαμε και με αντίστοιχο τρόπο σκεφτήκαμε για τα υπόλοιπα tables:

```

create table if not exists trip(
tr_id int(11) not null auto_increment ,
tr_departure date not null,
tr_return date not null,
tr_maxseats tinyint(4) default '0' not null,
tr_cost float(7,2) not null,
tr_br_code int (11) not null,
tr_gui_AT char(10) not null,
tr_drv_AT char(10) not null,
primary key (tr_id),
CONSTRAINT tripcode FOREIGN KEY (tr_br_code) REFERENCES branch(br_code)
ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT guided FOREIGN KEY (tr_gui_AT) REFERENCES guide(gui_AT)
ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT driven FOREIGN KEY (tr_drv_AT) REFERENCES driver(drv_AT)
ON UPDATE CASCADE ON DELETE CASCADE
) ;

```

Έχει προηγηθεί η δημιουργία των κατάλληλων table και μεταβλητών που απαιτούνται για την ορθή λειτουργία του trip.

Το σχεσιακό μοντέλο που μας δόθηκε, μας καθοδηγεί ώστε να δημιουργήσουμε τις κατάλληλες μεταβλητές καθώς και τι είδος είναι η κάθε μια απ' αυτές.

Χρησιμοποιούμε **ως πρωτεύων κλειδί το tr\_id** το οποίο προσδιορίζει τον κωδικό μέσω του οποίου βρίσκουμε το κάθε ταξίδι.

Παράλληλα, θέτουμε **ως δευτερεύοντα κλειδιά** τα **tr\_br\_code, tr\_gui\_AT, tr\_drv\_AT** , με σκοπό να συνδέσουμε το trip με τους αντίστοιχους πίνακες (branch,guide,driver) και να μπορούμε να αντλήσουμε από αυτούς τα κατάλληλα στοιχεία.

Τέλος δημιουργούμε τις μεταβλητές :

**tr\_departure** (μεταβλητή αποθήκευσης της αναχώρησης)

**tr\_return** (μεταβλητή αποθήκευσης της επιστροφής)

**tr\_maxseats** (μεταβλητή αποθήκευσης των μέγιστων θέσεων κάθε ταξιδιού)

**tr\_cost**(μεταβλητή αποθήκευσης κόστους ταξιδιού)

Με αντίστοιχη λογική δημιουργούμε και τα υπόλοιπα tables που απαιτούνται για την δημιουργία της βάσης δεδομένων.

Ταυτόχρονα ,εξηγούμε την λογική πάνω σε ένα από τα insert. Με αντίστοιχο τρόπο χειριζόμαστε και τα υπόλοιπα tables.

### **DESTINATION:**

```
insert into destination values
```

```
(null,'Epidavros','Northeastern Side Of Peloponnese ','local','GREEK',1),  
(null,'Acropolis','Center of Athens ','local','GREEK',2),  
(null,'Kalavryta','Mountainous Town in Peloponnese','local','GREEK',3),  
(null,'Mykhnes','Ancient City of Argolida','local','GREEK',4),  
(null,'Amsterdam','Capital city of Netherlands ','abroad','ENGLISH',5),  
(null,'London','Capital city of England ','abroad','ENGLISH',6),  
(null,'Berlin','Capital city of Germany ','abroad','GERMAN',7),  
(null,'Barcelona','Beautiful city in Spain','abroad','SPANISH',8),  
(null,'New York','Beautiful city in Spain ','abroad','ENGLISH',9),  
(null,'Tokyo','Capital city of Japan ','abroad','JAPANESE',10);
```

Με βάση το table destination εισάγουμε ως παραδείγματα τους 10 παραπάνω προορισμούς για την υλοποίηση της βάσης. Χρησιμοποιούμε το όνομα του προορισμού, μια περιγραφή για αυτόν , enum για το εάν είναι στη χώρα ή εκτός , την γλώσσα και το id της τοποθεσίας ( βοηθητικό του dst\_id).Τέλος το dst\_id το χρησιμοποιούμε ως **null** διότι έχουμε δημιουργήσει μια auto increment μεταβλητή και έτσι αυξάνεται αυτόματα την κάθε φορά.

Παρακάτω δημιουργούμε την select η οποία εκτυπώνει τα παραπάνω στοιχεία του destination:

```
select *  
from destination;
```

### **Screenshot output:**

	dst_id	dst_name	dst_desc	dst_rtype	dst_languages	dst_location
▶	1	Epidavros	Northeastern Side Of Peloponnese	LOCAL	GREEK	1
	2	Acropolis	Center of Athens	LOCAL	GREEK	2
	3	Kalavryta	Mountainous Town in Peloponnese	LOCAL	GREEK	3
	4	Mykhnes	Ancient City of Argolida	LOCAL	GREEK	4
	5	Amsterdam	Capital city of Netherlands	ABROAD	ENGLISH	5
	6	London	Capital city of England	ABROAD	ENGLISH	6
	7	Berlin	Capital city of Germany	ABROAD	GERMAN	7
	8	Barcelona	Beautiful city in Spain	ABROAD	SPANISH	8
	9	New York	Beautiful city in Spain	ABROAD	ENGLISH	9
	10	Tokyo	Capital city of Japan	ABROAD	JAPANESE	10



## ΣΗΜΕΙΩΣΗ:

Για την ορθή λειτουργία του παραπάνω πίνακα καθώς και της υπόλοιπης βάσης δεδομένων, απαιτείται η εισαγωγή στοιχείων (insert) σε όλα tables και μάλιστα με σωστή σειρά, όπως παρουσιάζεται.

Αυτό συμβαίνει, διότι προκειμένου κάποια tables να λειτουργήσουν, χρειάζονται πληροφορίες οι οποίες βρίσκονται σε κάποιο άλλο table λόγω των πρωτεύοντων και δευτερεύοντων κλειδιών (πχ. ο driver από τον worker).

## 3.1.2 Νέες απαιτήσεις:

### 3.1.2.1

Δημιουργία table IT\_manager. Με βάση τις νέες απαιτήσεις χρησιμοποιούμε τις μεταβλητές που παρατέθηκαν (drv\_AT, drv\_license, drv\_route, drv\_experience) με σκοπό την αποθήκευση των στοιχείων στο table.

```
create table if not exists it_manager (  
    it_AT char(10) not null,  
    password char(100) default 'password' not null,  
    start_date date not null,  
    end_date date,  
    primary key (it_AT),  
    CONSTRAINT itat FOREIGN KEY (it_AT) REFERENCES worker(wrk_AT)  
        ON UPDATE CASCADE ON DELETE CASCADE  
);
```

Συνδέουμε το it\_manager table με του worker ως δευτερεύον κλειδί προκειμένου να προσδιορίσουμε ότι ένας manager είναι και worker.

Για αυτό, τον αντίστοιχο manager τον αποθηκεύουμε και στο table worker.

Αφού φτιάξουμε το table του it\_manager για να επιτρέψουμε την δυνατότητα να υπάρχουν περισσότεροι από ένας υπεύθυνοι πληροφορικής την ίδια χρονική περίοδο, αφαιρούμε το not null constraint από το end\_date attribute.

Έτσι μπορούμε να έχουμε πολλούς it\_managers την ίδια χρονική περίοδο με την προϋπόθεση ότι str\_date < end\_date.

Παραθέτουμε την insert του manager στο table it\_manager , την select που απαιτείται για την εξαγωγή των στοιχείων του manager καθώς και το output σε screenshot. Προφανώς για την ορθή λειτουργία της insert χρειαζόμαστε την εισαγωγή στοιχείων και στους υπόλοιπους πίνακες (όπως προαναφέρθηκε).

```
INSERT INTO it_manager (it_AT, password, start_date, end_date)
VALUES ('AN453827', 'password123', '2022-06-01', '2022-06-30');
```

Select:

```
select *
from it_manager;
```

output:

	it_AT	password	start_date	end_date
▶	AN453827	password123	2022-06-01	2022-06-30

### 3.1.2.2

Για την δημιουργία του πίνακα log παραθέτουμε την λογική:

Χρησιμοποιούμε τα παρακάτω πιθανά κελιά για την κάθε ενέργεια του manager:

- Το id του κάθε μανατζερ(χρησιμοποιείται ως κλειδί για να προσδιορίσει με βάση τα υπόλοιπα στοιχεία για ποιόν μανατζερ μιλάμε).
- Μοναδικό ID το οποίο προσδιορίζει πόσα entry και ποια έκανε ο manager.
- Περιγραφή για την κάθε ενέργεια που πραγματοποιήθηκε.

```
create table if not exists log (
  login_AT char(10) not null,
  login_id int(11) not null auto_increment,
  login_activity varchar(500) not null,
  primary key (login_id),
  CONSTRAINT login_AT FOREIGN KEY (login_AT) REFERENCES it_manager(it_AT)
  ON UPDATE CASCADE ON DELETE CASCADE
);
```

## INSERT:

```
INSERT INTO log (login_AT, login_id, login_activity)
VALUES ('AN453827', '1', 'Created a new trip');
```

Παραθέτουμε την select μας η οποία συνδυάζεται με το table it\_manager για την απεικόνιση μιας δραστηριότητας ενός συγκεκριμένου μάνατζερ με βάση το κλειδί login\_AT :

	login_AT	login_id	login_activity	it_AT	password	start_date	end_date
▶	AN453827	1	Created a new trip	AN453827	password123	2022-06-01	2022-06-30

### 3.1.2.3

α) Με βάση τις μεταβλητές που ζητούνται (

- κωδικό προσφοράς ταξιδιού (μοναδικός)
- περίοδο που μπορεί να πραγματοποιηθεί ένα ταξίδι
- ημερομηνία έναρξης ,ημερομηνία λήξης
- κόστος ανά άτομο
- προορισμό (dst\_id) )

Φτιάχνουμε το παρακάτω table:

```
CREATE TABLE IF NOT EXISTS offers (
  promo_code char(10) NOT NULL,
  start_date date NOT NULL,
  expiry_date date NOT NULL,
  cost float(7,2) NOT NULL,
  dst_id int(11) NOT NULL,
  PRIMARY KEY (promo_code),
  CONSTRAINT dest FOREIGN KEY (dst_id) REFERENCES destination(dst_id)
  ON UPDATE CASCADE ON DELETE CASCADE
);
```

Επειδή ζητείται ο προορισμός να συνδέεται με το dst\_id , χρησιμοποιούμε την παραπάνω μεταβλητή ως δευτερεύον κλειδί στον πίνακα destination ώστε να έχουμε πρόσβαση στις πληροφορίες του πίνακα.

## INSERT:

```
INSERT INTO offers (promo_code, start_date, expiry_date, cost, dst_id)
VALUES
('LOCAL1', '2023-06-01', '2023-07-30', 260.01, 1),
('ABROAD4', '2023-08-01', '2023-09-30', 410.01, 4),
('ABROAD9', '2023-09-01', '2023-10-30', 660.01, 9);
```

## SELECT & OUTPUT:

```
select *
from offers;
```

	promo_code	start_date	expiry_date	cost	dst_id
▶	ABROAD4	2023-08-01	2023-09-30	410.01	4
	ABROAD9	2023-09-01	2023-10-30	660.01	9
	LOCAL1	2023-06-01	2023-07-30	260.01	1

Χρησιμοποιούμε τα παραπάνω ενδεικτικά κόστη, κωδικούς και ημερομηνίες οι οποίες αντιστοιχούν στους προορισμούς του destination 4,9,1.

b) Με βάση τις μεταβλητές που ζητούνται:

- ☐ κωδικό κράτησης προσφοράς ταξιδιού
- ☐ επώνυμο πελάτη
- ☐ όνομα πελάτη
- ☐ κωδικό προσφοράς ταξιδιού
- ☐ ποσό προκαταβολής

Φτιάχνουμε το παρακάτω table:

```
CREATE TABLE reservation_offers (
  booking_code INT(11) NOT NULL AUTO_INCREMENT,
  name CHAR(20) NOT NULL,
  last_name VARCHAR(20) NOT NULL,
  promo_code_res CHAR(10) NOT NULL,
  down_payment FLOAT(7,2) NOT NULL,
  PRIMARY KEY (booking_code),
  CONSTRAINT offers FOREIGN KEY (promo_code_res) REFERENCES
offers(promo_code)
  ON UPDATE CASCADE ON DELETE CASCADE
);
```



Τοποθετείται στον κώδικα κάτω από το table reservations (χρησιμοποιεί foreign key). Συνδέεται με το table offers από το οποίο παίρνει όλες τις απαραίτητες πληροφορίες.

Στο επόμενο ερώτημα ζητείται η εισαγωγή 60000 reservation\_offers επομένως την insert την παραθέτουμε εκεί , μαζί με την εξήγηση του κώδικα και το αντίστοιχο output.

c)Εισάγουμε αρχικά, 3 προσφορές ταξιδιών με κόστος μεγαλύτερο των 200 ευρώ:

INSERT:

```
INSERT INTO offers (promo_code, start_date, expiry_date, cost, dst_id)
VALUES
('LOCAL1', '2023-06-01', '2023-07-30', 260.01, 1),
('ABROAD4', '2023-08-01', '2023-09-30', 410.01, 4),
('ABROAD9', '2023-09-01', '2023-10-30', 660.01, 9);
```

Οι παραπάνω εγγραφές αναφέρονται με βάση το dst\_id στα ταξίδια 1,4 και 9 αντίστοιχα με κωδικούς LOCAL1, ABROAD4, ABROAD9.

SELECT & OUTPUT:

```
SELECT *
FROM OFFERS;
```

	promo_code	start_date	expiry_date	cost	dst_id
▶	ABROAD4	2023-08-01	2023-09-30	410.01	4
	ABROAD9	2023-09-01	2023-10-30	660.01	9
	LOCAL1	2023-06-01	2023-07-30	260.01	1

Στην συνέχεια, καταχωρούμε έναν αριθμό 60000 εγγραφών στο αρχείο μας. Για να το κάνουμε αυτό:

- Χρησιμοποιούμε τα 10000 ονόματα που έχουν δοθεί στο eclass
- Χρησιμοποιούμε το πρόγραμμα excel με τον εξής τρόπο:

Εισάγουμε τα ονόματα στο αρχείο και τα επικολλούμε 6 φορές ώστε να έχουμε τον επιθυμητό αριθμό εγγραφών. Έπειτα, εισάγουμε έναν κωδικό προσφοράς σε κάθε άτομο με βάση τον κωδικό του πίνακα offers. Τέλος εισάγουμε ένα ποσό προκαταβολής (στους πρώτους 20000 50 ευρώ, στους επόμενους 100 και στους τελευταίους 200).

Λόγω του τεράστιου όγκου εγγραφών, παραθέτουμε κάποια ενδεικτικά screenshots της insert ενώ στο αρχείο insert reservation\_offers.sql μπορείτε να δείτε ολόκληρο τον κώδικα:

```
INSERT INTO reservation_offers(booking_code, name, last_name, promo_code_res, down_payment  
VALUES(null, "Itzel", "Moyer", "LOCAL1", "50"),  
(null, "Thalia", "Hernandez", "LOCAL1", "50"),  
(null, "Myla", "Craig", "LOCAL1", "50"),  
(null, "Jazmine", "Alvarez", "LOCAL1", "50"),  
(null, "Tristin", "Banks", "LOCAL1", "50"),  
(null, "Aliyah", "Holmes", "LOCAL1", "50"),  
(null, "Joselyn", "Curtis", "LOCAL1", "50"),  
(null, "Harper", "Walters", "LOCAL1", "50"),  
(null, "Augustus", "Kent", "LOCAL1", "50"),  
(null, "Gaige", "Compton", "LOCAL1", "50"),  
(null, "Tommy", "Ewing", "LOCAL1", "50"),  
(null, "Denzel", "Mathis", "LOCAL1", "50"),  
(null, "Karli", "Vaughan", "LOCAL1", "50"),  
(null, "Milagros", "Tate", "LOCAL1", "50"),  
(null, "Cordell", "Richard", "LOCAL1", "50"),  
(null, "Adrien", "Miranda", "LOCAL1", "50"),  
(null, "Denise", "Wolf", "LOCAL1", "50"),  
(null, "Genesis", "Zamora", "LOCAL1", "50"),
```

33816 (null, "Briley", "Macias", "ABROAD4", "100"),  
 33817 (null, "Darnell", "Shannon", "ABROAD4", "100"),  
 33818 (null, "Marquise", "Stout", "ABROAD4", "100"),  
 33819 (null, "Wyatt", "Pineda", "ABROAD4", "100"),  
 33820 (null, "Rylan", "Curry", "ABROAD4", "100"),  
 33821 (null, "Autumn", "Harding", "ABROAD4", "100"),  
 33822 (null, "Violet", "Jacobs", "ABROAD4", "100"),  
 33823 (null, "Ryan", "Paul", "ABROAD4", "100"),  
 33824 (null, "Jazlynn", "Velez", "ABROAD4", "100"),  
 33825 (null, "Laylah", "Ewing", "ABROAD4", "100"),  
 33826 (null, "Jazmin", "Garrison", "ABROAD4", "100"),  
 33827 (null, "Hailey", "Noble", "ABROAD4", "100"),  
 33828 (null, "Mikayla", "Wolfe", "ABROAD4", "100"),  
 33829 (null, "Shiloh", "Wilkerson", "ABROAD4", "100"),  
 33830 (null, "Lilah", "Fleming", "ABROAD4", "100"),  
 33831 (null, "Alejandro", "Blackwell", "ABROAD4", "100"),  
 33832 (null, "Cason", "Beard", "ABROAD4", "100"),  
 33833 (null, "Maci", "Tucker", "ABROAD4", "100"),  
 33834 (null, "Zaria", "Maynard", "ABROAD4", "100"),  
 33835 (null, "Allison", "Payne", "ABROAD4", "100").

38297 (null, "Yair", "Kim", "ABROAD9", "200"),  
 38298 (null, "Arthur", "Williams", "ABROAD9", "200"),  
 38299 (null, "Rodney", "Potts", "ABROAD9", "200"),  
 38300 (null, "Eliezer", "Shields", "ABROAD9", "200"),  
 38301 (null, "Titus", "Armstrong", "ABROAD9", "200"),  
 38302 (null, "Nathaniel", "Saunders", "ABROAD9", "200"),  
 38303 (null, "Karsyn", "Frost", "ABROAD9", "200"),  
 38304 (null, "Sierra", "Montoya", "ABROAD9", "200"),  
 38305 (null, "Bailey", "Guerrero", "ABROAD9", "200"),  
 38306 (null, "Everett", "Cannon", "ABROAD9", "200"),  
 38307 (null, "Alissa", "Sanford", "ABROAD9", "200"),  
 38308 (null, "Nora", "Best", "ABROAD9", "200"),

### **3.1.3 Δημιουργία stored procedure:**

#### **3.1.3.1**

Σύμφωνα με την εκφώνηση, ζητείται η δημιουργία μιας procedure στην οποία δίνονται ως είσοδοι τα στοιχεία ενός νέου οδηγού. Η procedure θα εντοπίζει το υποκατάστημα με τους λιγότερους οδηγούς και θα εισάγει τον οδηγό ως υπάλληλο του συγκεκριμένου υποκαταστήματος στους πίνακες worker και driver.

Όταν δημιουργούμε έναν νέο οδηγό:

- Χρειαζόμαστε τα στοιχεία του πίνακα driver,
- τα στοιχεία του πίνακα worker (ο οδηγός είναι worker)
- πρέπει μέσω αυτού του πίνακα να συνδεθεί (Μέσω των κλειδιών) στον branch όπως φαίνεται στο σχεσιακό μοντέλο.

Αρχικά στα ορίσματα της procedure , η μεταβλητή wrk\_br\_code δεν εισάγεται γιατί το υποκατάστημα που εισάγεται ο οδηγός, είναι αυτό που αναζητείται στην procedure.

Εκτός από το παραπάνω στοιχείο εισάγουμε ως ορίσματα τα υπόλοιπα στοιχεία που θέλουμε να αποθηκευτούν στους πίνακες worker, driver

```
(IN driverID char(10), IN fname varchar(20), IN lname varchar(20), IN salary float, IN driverlicense enum('A','B','C','D'), IN routeType enum('LOCAL','ABROAD'), IN monthsExp tinyint).
```

**Πέρα από τα ορίσματα, χρησιμοποιούμε τον παρακάτω πίνακα για να επιλέξουμε τα στοιχεία br\_code και το ΠΛΗΘΟΣ (χρήση της count()) των οδηγών που δουλεύουν σε κάθε υποκατάστημα).**

**Αυτό για να το πετύχουμε χρειαζόμαστε στοιχεία από διαφορετικούς πίνακες. Επομένως μέσω των κατάλληλων κλειδιών ,join και της group by(καθορίζει κάθε φορά ότι ομαδοποιούνται με βάση το ίδιο κατάστημα μέσω του br\_code) ενώνουμε τους υπάρχοντες πίνακες και παίρνουμε τα στοιχεία που θέλουμε.**

Παρακάτω παραθέτουμε τον νέο πίνακα που χρησιμοποιούμε για την εξαγωγή των απαραίτητων στοιχείων:

```
SELECT br_code, COUNT(drv_AT) as numDrivers
FROM driver d
JOIN worker w ON w.wrk_AT=d.drv_AT
JOIN branch b ON b.br_code=wrk_br_code
GROUP BY br_code;
```

Πάνω στον παραπάνω πίνακα που εξηγήθηκε, επιλέγουμε μέσω της συνάρτησης MIN, το ελάχιστο πλήθος αριθμού οδηγών που υπάρχει σε ένα υποκατάστημα. Την πληροφορία αυτή, την αποθηκεύουμε στην μεταβλητή minDrivers.

```
SELECT MIN(numDrivers) INTO minDrivers
FROM (SELECT br_code, COUNT(drv_AT) as numDrivers
FROM driver d
JOIN worker w ON w.wrk_AT=d.drv_AT
JOIN branch b ON b.br_code=wrk_br_code
GROUP BY br_code) as temp;
```

Επειδή πολλά καταστήματα είναι δυνατόν να έχουν το ίδιο ελάχιστο πλήθος αριθμού οδηγών, επιλέγεται το πρώτο υποκατάστημα που πληρεί την παραπάνω προϋπόθεση (WHERE numDrivers = minDrivers). Η πληροφορία αυτή αποθηκεύεται στην μεταβλητή minBranchID:

```
SELECT br_code INTO minBranchID
FROM (SELECT br_code, COUNT(drv_AT) as numDrivers
FROM driver d
JOIN worker w ON w.wrk_AT=d.drv_AT
JOIN branch b ON b.br_code=wrk_br_code
GROUP BY br_code) as temp
WHERE numDrivers = minDrivers
LIMIT 1 ;
```

Τέλος αφού αντλήσουμε τις παραπάνω απαραίτητες πληροφορίες με βάση τις δικές μας προϋποθέσεις, εισάγουμε τους οδηγούς στους πίνακες driver και worker:

```
INSERT INTO worker (wrk_AT, wrk_name, wrk_lname, wrk_salary, wrk_br_code)
VALUES (driverID, fname, lname, salary, minBranchID);
```

```
INSERT INTO driver (drv_AT, drv_license, drv_route, drv_experience)
VALUES (driverID, driverlicense, routeType, monthsExp);
```

Εδώ δείχνουμε ενδεικτικά ένα call και ένα output:

```
CALL addDriver('AM111112', 'Neos', 'Odhgos', 1000.00, 'B', 'ABROAD', 6);
```

SELECT & OUTPUT:

```
SELECT br_code, COUNT(drv_AT) as numDrivers ,wrk_name, wrk_lname
FROM driver d join worker w
ON w.wrk_AT=d.drv_AT
join branch b
ON b.br_code=wrk_br_code
GROUP BY br_code
Order by br_code;
```

	br_code	numDrivers	wrk_name	wrk_lname
▶	1	2	Neos	Odhgos
	2	1	Alekos	Karpas
	3	1	Panagioths	Patsidis
	4	1	Alexandros	Karatzas
	5	1	Anastasis	Koutsougeorgou
	6	1	Mpampis	Fesatoglou

### 3.1.3.2

Σύμφωνα με την εκφώνηση, ζητείται η δημιουργία μιας procedure στην οποία θα δίνονται ως όρισμα ο κωδικός ενός υποκαταστήματος και δύο ημερομηνίες και θα επιστρέφονται τα εξής:

Κόστος ταξιδιού (tr\_cost ), μέγιστες θέσεις maxseat, σύνολο κρατήσεων (reservations), κενές θέσεις (maxseat – σύνολο κρατήσεων), επώνυμο και όνομα οδηγού και ξεναγού, ημερομηνία αναχώρησης και επιστροφής.

Αρχικά, για την παραπάνω procedure δημιουργούμε μια βοηθητική procedure η οποία έχει στόχο να υπολογίζει τις **κενές θέσεις** (maxseat – σύνολο κρατήσεων).

Ονομάζεται seats\_available και παίρνει ως ορίσματα τα ίδια με την αρχική μας procedure.

Για την procedure αυτή, χρειαζόμαστε στοιχεία από τους πίνακες:

trip,reservatios,branch. Για αυτό μέσω των κατάλληλων κλειδιών βάση του σχεσιακού μοντέλου, ενώνουμε τα tables και :

**Εξάγουμε στην select τις μέγιστες θέσεις - το ΠΛΗΘΟΣ των reservations.**



Παράθεση ολόκληρου του κώδικα της βοηθητικής procedure:

```
drop procedure if exists seats_available;

DELIMITER $$
CREATE PROCEDURE seats_available(IN br_code INT, IN start_date DATE, IN
end_date DATE,OUT available_seats int(4))
BEGIN
Select tr_maxseats - count(r.res_tr_id) into available_seats
FROM trip t
JOIN branch b ON b.br_code = t.tr_br_code
JOIN reservation r ON t.tr_id = r.res_tr_id
WHERE (b.br_code = br_code AND t.tr_departure = start_date AND
t.tr_return=end_date)
GROUP BY t.tr_id;
END $$
DELIMITER ;
```

Στη συνέχεια δημιουργούμε την procedure που ζητείται με όνομα trip\_info. Παίρνει ως ορίσματα τις παραπάνω προϋποθέσεις και :

-Καλεί την βοηθητική συνάρτηση παίρνοντας τα ίδια ορίσματα με την trip\_info  
-Δημιουργεί με declare τα στοιχεία που θέλουμε να εξαχθούν **με σκοπό να μπορέσουμε να τα αποθηκεύσουμε κάπου, ώστε να τα χρησιμοποιήσουμε στην τελική μας select που εξάγει τα επιθυμητά αποτελέσματα.**  
Για τις παραπάνω πληροφορίες(κόστη ταξιδιού, όνομα οδηγού κλπ.) , χρειαζόμαστε στοιχεία από τους πίνακες trip, reservatios, branch worker, driver, guide και trip.

Επειδή τα στοιχεία του driver, guide είναι αποθηκευμένα στον πίνακα worker, για να μπορέσουμε να τα ξεχωρίσουμε χρησιμοποιούμε 2 διαφορετικά κομμάτια select. Μέσω του tr\_gui\_AT και του tr\_drv\_at στον πίνακα trip ενώνουμε τα απαραίτητα στοιχεία για τους οδηγούς, ξεναγούς, θέσεις κλπ.

Παράθεση του κώδικα:

```
drop procedure if exists trip_info;

DELIMITER $$
CREATE PROCEDURE trip_info(IN br_code INT, IN start_date DATE, IN end_date
DATE)
BEGIN
declare tr_seats INT(4);
declare cost float(7,2);
declare maxseats tinyint(4);
```

```

declare reservations int(200);
declare departure date;
declare return_day date;
declare driver_name varchar(20);
declare driver_lname varchar(20);
declare guide_name varchar(20);
declare guide_lname varchar(20);
CALL seats_available(br_code, start_date, end_date, tr_seats);
    SELECT w.wrk_name, w.wrk_lname into driver_name,driver_lname
    FROM trip t
    JOIN branch b ON b.br_code=t.tr_br_code
    JOIN driver d ON t.tr_drv_AT = d.drv_AT
    JOIN worker w ON w.wrk_AT = d.drv_AT
    WHERE b.br_code = br_code AND t.tr_departure=start_date AND
t.tr_return=end_date
    GROUP BY t.tr_id;

    SELECT w.wrk_name,w.wrk_lname into guide_name,guide_lname
    FROM trip t
    JOIN branch b ON b.br_code=t.tr_br_code
    JOIN guide g ON t.tr_gui_AT = g.gui_AT
    JOIN worker w ON w.wrk_AT = g.gui_AT
    WHERE b.br_code = br_code AND t.tr_departure=start_date AND
t.tr_return=end_date
    GROUP BY t.tr_id;

    SELECT t.tr_cost,tr_maxseats,COUNT(r.res_tr_id),tr_departure
,t.tr_return into cost,maxseats,reservations,departure,return_day
    from trip t
    JOIN branch b ON b.br_code=t.tr_br_code
    JOIN reservation r ON r.res_tr_id = t.tr_id
    WHERE b.br_code = br_code AND t.tr_departure=start_date AND
t.tr_return=end_date
    GROUP BY t.tr_id;

    select cost as Trip_Cost,maxseats as Max_Seats,reservations as
Reservations, tr_seats as Available_Seats,driver_name as
Driver_Name,driver_lname as Driver_Lname,
    guide_name as Guide_Name,guide_lname as guide_lname,departure as
Departure_Date , return_day as Return_Day;

END $$
DELIMITER ;

```

Call & output:

```
call trip_info(2, '2024-01-04', '2024-01-08');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: 1A

	Trip_Cost	Max_Seats	Reservations	Available_Seats	Driver_Name	Driver_Lname	Guide_Name	guide_lname	Departure_Date	Return_Day
	650.00	80	1	79	Alekos	Karpas	Mike	Mprakopoulos	2024-01-04	2024-01-08

### 3.1.3.3

Σύμφωνα με την εκφώνηση, ζητείται η δημιουργία μιας procedure στην οποία ως όρισμα το όνομα και το επώνυμο ενός υπαλλήλου. Αν είναι διοικητικός τον διαγράφει. Αν ο υπάλληλος είναι διευθυντής υποκαταστήματος τότε εμφανίζεται μήνυμα ότι είναι διευθυντής του υποκαταστήματος και δεν επιτρέπει τη διαγραφή.

Αρχικά, χρησιμοποιούμε το όνομα και το επώνυμο που δίνονται ως ορίσματα ως προϋπόθεση (WHERE) μέσω της οποίας εντοπίζουμε τον αντίστοιχο admin. Επίσης, χρησιμοποιούμε το at του admin/worker ως κλειδιά για να συνδέσουμε τους δυο πίνακες. Τέλος δημιουργούμε με την declare μια μεταβλητή η οποία αποθηκεύει τον τύπο του admin ώστε να τον έχουμε στην κατοχή μας μέσα στην procedure.

```
DECLARE worker_type enum('LOGISTICS', 'ADMINISTRATIVE', 'ACCOUNTING');  
SELECT adm_type INTO worker_type  
FROM worker w join admin a  
on w.wrk_AT=a.adm_AT  
WHERE name = wrk_name AND wrk_lname = lname;
```

Στην συνέχεια αφού αποθηκεύσουμε με βάση την παραπάνω προϋπόθεση και τους πίνακες, τον τύπο του admin παίρνουμε το admin\_type και το τοποθετούμε σε μια if. Αν ο admin είναι logistics ή accounting τότε διαγράφεται και εκτυπώνεται μήνυμα ότι συνέβη διαγραφή ενώ αν είναι administrative εκτυπώνεται μήνυμα ότι απορρίπτεται η διαγραφή του.

```
IF worker_type IN ('LOGISTICS', 'ACCOUNTING') THEN  
DELETE w, a  
FROM worker w  
JOIN admin a ON w.wrk_AT = a.adm_AT  
WHERE name=wrk_name AND lname=wrk_lname ;
```

```

SELECT CONCAT(name, ' ', lname, ' Admin deleted deleted.') AS message
;
ELSEIF worker_type = 'ADMINISTRATIVE' THEN
    SELECT CONCAT(name, ' ', lname, ' is Administrative of a branch
and cannot be deleted.') AS message ;
ELSE
    SELECT CONCAT(name, ' ', lname, ' is not an admin.') AS message ;
END IF;

```

Παραθέτουμε ένα call με έναν admin ο οποίος είναι στον τομέα administrative και έναν που είναι στον τομέα logistics. Επίσης, παραθέτουμε την περίπτωση όπου ο worker που προσπαθούμε να διαγράψουμε δεν είναι admin καμίας κατηγορίας. Για καθέναν από αυτούς έχουμε βάλει να εκτυπώνεται κατάλληλο μήνυμα:

Output 1:

```
call delete_admin("Petros", "Petrou");
```

	message
▶	Petros Petrou is Administrative of a branch and cannot be deleted.

Output 2:

```
call delete_admin("Mark", "Zukenberg");
```

	message
▶	Mark Zukenberg is not an admin.

### **3.1.3.4**

Πριν την επεξήγηση των procedure, επεκτείνουμε την database μας ώστε πλέον στο table reservation offers να έχουμε τα κατάλληλα ευρετήρια για τις κατάλληλες τιμές:

```

create index res_name_idx
on reservation_offers(name);
create index res_lname_idx
on reservation_offers(last_name);
create index res_payment_idx
on reservation_offers(down_payment);

```

Δημιουργούμε τα παραπάνω index για τις τιμές που αναφέρονται.

Επίσης παραθέτουμε τις διαφορές χρόνου για κάθε περίπτωση που τρέχουμε τις 60000 εγγραφές:

1<sup>η</sup> περίπτωση:

59999 row(s) affected Records: 59999 Duplicates: 0 Warnings: 0	2.453 sec
--	-----------

2<sup>η</sup> περίπτωση:

Message	Duration / Fetch
59999 row(s) affected Records: 59999 Duplicates: 0 Warnings: 0	1.484 sec

**A)** Σύμφωνα με την εκφώνηση, ζητείται η δημιουργία μιας procedure στην οποία δίνονται ως παράμετροι εισόδου δύο τιμές και επιστρέφει τους πελάτες (επώνυμο, όνομα) που έκαναν κρατήσεις σε προσφορές ταξιδιών και πλήρωσαν για προκαταβολή, ποσό ανάμεσα στις δύο τιμές.

```
INSERT INTO reservation_offers(booking_code,name,last_name,promo_code_res,down_payment)
VALUES(null, "Itzel", "Moyer", "LOCAL1", "51"),
(null, "Thalia", "Hernandez", "LOCAL1", "52"),
(null, "Myla", "Craig", "LOCAL1", "53"),
(null, "Jazmine", "Alvarez", "LOCAL1", "54"),
(null, "Tristin", "Banks", "LOCAL1", "55"),
(null, "Aliyah", "Holmes", "LOCAL1", "56"),
(null, "Joselyn", "Curtis", "LOCAL1", "57"),
(null, "Harper", "Walters", "LOCAL1", "58"),
(null, "Augustus", "Kent", "LOCAL1", "59"),
(null, "Gaige", "Compton", "LOCAL1", "50"),
(null, "Tommy", "Ewing", "LOCAL1", "50"),
(null, "Denzel", "Mathis", "LOCAL1", "50"),
(null, "Karli", "Vaughan", "LOCAL1", "50"),
(null, "Milagros", "Tate", "LOCAL1", "50"),
(null, "Cordell", "Richard", "LOCAL1", "50"),
(null, "Adrien", "Miranda", "LOCAL1", "50"),
(null, "Denise", "Wolf", "LOCAL1", "50"),
(null, "Genesis", "Zamora", "LOCAL1", "50"),
(null, "Manuel", "Livingston", "LOCAL1", "50"),
(null, "Layla", "Buckley", "LOCAL1", "50"),
(null, "Ishaan", "Moss", "LOCAL1", "50"),
-- 33 More rows --
```

Όπως παραθέσαμε παραπάνω στο αρχείο με τις 60000 κρατήσεις, έχουμε βάλει ως ποσά προκαταβολής 50, 100 και 200 ευρώ.

**Ωστόσο** για την αποδοτικότερη απεικόνιση του ερωτήματος αλλάζουμε τις 10 πρώτες προκαταβολές σε τιμές μεταξύ του 50 και του 100.

```
create procedure payment_insert(var1 float(7.2), var2 float (7.2))
Begin
declare bname char(20);
declare blname char(20);
DECLARE not_found INT default 0;
DECLARE bcursor CURSOR FOR
    SELECT name, last_name
    FROM reservation_offers
    WHERE down_payment between var1 AND var2;
```

Για την δημιουργία της procedure βάζουμε ως παράμετρούς εισόδου 2 μεταβλητές var1 var2 τύπου float οι οποίες προσδιορίζουν το εύρος τιμών που θέλουμε να βρούμε. Παράλληλα χρησιμοποιούμε ένα cursor για να σαρώσουμε τον πίνακα reservation\_offers ώστε να εντοπίσουμε τα ονοματεπώνυμα που έχουν το παραπάνω εύρος τιμών (προσδιορίζεται με μια where) :

```
DECLARE bcursor CURSOR FOR
    SELECT name, last_name
    FROM reservation_offers
    WHERE down_payment between var1 AND var2;
```

Επιπλέον μέσω της μεταβλητής not\_found προσδιορίζουμε πότε τελειώνει η παρακάτω repeat until που χρειάζεται ώστε να εντοπίσουμε όλες τις φορές που υπάρχει το παραπάνω εύρος τιμών.

Παρουσιάζουμε τα αποτελέσματα για την εξαγωγή των ατόμων που έκλεισαν προσφορά **μεταξύ των τιμών 51 και 70**.

Call & output:

```
Call payment_insert("51", "70");
```



	name	last_name
►	Itzel	Moyer
	Thalia	Hernandez
	Myla	Craig
	Jazmine	Alvarez
	Tristin	Banks
	Aliyah	Holmes
	Joselyn	Curtis
	Harper	Walters
	Augustus	Kent

Όπως δείχνουμε παραπάνω στο screenshot του νέου insert, αυτά είναι τα ονόματα των ατόμων που έκλεισαν προσφορά με προκαταβολή μεταξύ 51 και 70 ευρώ.

**B)** Σύμφωνα με το 2<sup>ο</sup> ερώτημα , ζητείται να δίνεται ως παράμετρος εισόδου το επώνυμο ενός πελάτη και επιστρέφει τα ονόματα και επώνυμα των πελατών με το επώνυμο αυτό και η προσφορά ταξιδιού στην οποία έχει γίνει εγγραφή.

Η διαδικασία δηλώνει πολλές μεταβλητές, συμπεριλαμβανομένου ενός "bcursor" που επιλέγει τις στήλες "name", "last\_name" και "promo\_code\_res" από τον πίνακα "reservation\_offers" όπου το επίθετο ταιριάζει με την τιμή εισόδου.

Ο cursor ανοίγει , και τα δεδομένα λαμβάνονται σε βρόχο χρησιμοποιώντας την πρόταση FETCH μέχρι να μην βρεθούν άλλα δεδομένα. Στη συνέχεια, ο κέρσορας κλείνει και η procedure τελειώνει:

```

declare blname char(20);
declare bname char(20);
declare bpromo_code char(10);
DECLARE not_found INT default 0;
DECLARE bcursor CURSOR FOR
    SELECT name,last_name,promo_code_res
FROM reservation_offers
    WHERE lname=last_name
group by bpromo_code;

    DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET not_found=1;

    OPEN bcursor;

    REPEAT

```

```

    FETCH bcursor INTO bname,blname,bpromo_code;
    SELECT name,last_name,promo_code_res
    FROM reservation_offers
    WHERE lname=last_name;

```

```

UNTIL(not_found=1)
END REPEAT;

```

```

CLOSE bcursor;

```

Παρακάτω δείχνουμε ένα απόσπασμα από τα output για όταν καλούμε την lname\_insert με όρισμα το επώνυμο Tate:

	name	last_name	promo_code_res
►	Milagros	Tate	LOCAL1
	Nola	Tate	LOCAL1
	Karlee	Tate	LOCAL1
	Romeo	Tate	LOCAL1
	Eddie	Tate	LOCAL1
	Myah	Tate	LOCAL1
	Marques	Tate	LOCAL1
	Trinity	Tate	LOCAL1
	Cannon	Tate	LOCAL1
	Jorge	Tate	LOCAL1
	Beatrice	Tate	LOCAL1

### **3.1.4 Δημιουργία triggers:**

**B)** Ο παραπάνω κώδικας δημιουργεί το trigger με όνομα "apotropi\_allagis\_hmerominias". Το trigger εκτελείται πριν γίνει μια ενημέρωση στον πίνακα "event". Το trigger έχει ρυθμιστεί να εκτελείται για κάθε σειρά που ενημερώνεται στον πίνακα "event".

Δηλώνει πρώτα μια μεταβλητή που ονομάζεται "reservation\_count" και ορίζει την προεπιλεγμένη τιμή σε 0. Στη συνέχεια εκτελεί μια δήλωση SELECT για να μετρήσει τον αριθμό των σειρών στον πίνακα "reservation" που έχουν το ίδιο "event\_id" με το "event\_id" του γραμμή που ενημερώνεται στον πίνακα "event". Το αποτέλεσμα της καταμέτρησης αποθηκεύεται στη μεταβλητή "reservation\_count".

Εάν η τιμή του "reservation\_count" είναι μεγαλύτερη από 0, σημαίνει ότι υπάρχει ήδη μια κράτηση για αυτό το συμβάν, επομένως ο κανόνας trigger δημιουργεί ένα σφάλμα χρησιμοποιώντας την πρόταση SIGNAL με τιμή SQLSTATE 45000 και ένα προσαρμοσμένο μήνυμα σφάλματος 'Dysthws den epitrepetai h allagh hmeromhnias.'

Έπειτα τερματίζεται με μια δήλωση ΤΕΛΟΣ. Στη συνέχεια, οριοθετείτε με ένα σύμβολο διπλού δολαρίου.

Η τελευταία δήλωση στον κώδικα είναι μια δήλωση UPDATE που επιχειρεί να ενημερώσει την ημερομηνία έναρξης και λήξης του συμβάντος με την τιμή "ev\_tr\_id" 1. Εάν γίνει κράτηση για αυτό το συμβάν, θα ενεργοποιηθεί η ενεργοποίηση, η δήλωση UPDATE θα αποτύχει και εμφανίζεται το προσαρμοσμένο μήνυμα σφάλματος "Dysthws den epitrepetai h allagh hmeromhnias." θα εμφανιστεί.

### **Παράθεση του κώδικα:**

```
drop trigger if exists apotropi_allagis_hmerominias;

DELIMITER $$
CREATE TRIGGER apotropi_allagis_hmerominias
BEFORE UPDATE ON event
FOR EACH ROW
BEGIN
DECLARE reservation_count INT DEFAULT 0;
SELECT COUNT(*) INTO reservation_count
FROM reservation
INNER JOIN event
ON reservation.res_tr_id = event.ev_tr_id
WHERE event.ev_tr_id = OLD.ev_tr_id;

IF reservation_count > 0 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'Dysthws den epitrepetai h allagh hmeromhnias. ';
END IF;
END$$
DELIMITER ;

UPDATE event SET ev_start = '2024-06-01', ev_end = '2024-06-05' WHERE
ev_tr_id = 1;
```

### **OUTPUT CODE META THN EKTEΛΕΣΗ TOY UPDATE:**

72 21:46:58 UPDATE event SET ev_start = '2024-06-01', ev_end = '2024-06-05' WHERE ev_tr_id = 1	Error Code: 1644. Dysthws den epitrepetai h allagh hmeromhnias.
--	---

**Q** Στο τελευταίο trigger δεν θα επιτρέπεται η μείωση του μισθού ενός υπαλλήλου. Το trigger έχει οριστεί να εκτελείται πριν από μια ενημέρωση στον πίνακα "εργαζόμενος" και να εκτελείται για κάθε σειρά που ενημερώνεται. Ορίζει μια μεταβλητή με το όνομα "updatedsalary" ίση με τη νέα τιμή μισθού στη δήλωση ενημέρωσης.

Στη συνέχεια, ελέγχει εάν η νέα αξία μισθού είναι μικρότερη από την παλιά αξία μισθού. Εάν συμβαίνει αυτό, η νέα αξία μισθού ορίζεται στην παλιά αξία μισθού για να αποφευχθεί η μείωση του μισθού. Έτσι, διασφαλίζεται ότι ο μισθός ενός εργαζομένου μπορεί μόνο να αυξηθεί και όχι να μειωθεί στον πίνακα «εργαζομένων».

Αφού δημιουργηθεί το trigger, ο κωδικός ενημερώνει τον μισθό εργαζομένου με τον κωδικό AT «AN453827» σε 1500,00. Ο προηγούμενος μισθός του συγκεκριμένου AT ήταν **μικρότερος** και με το παρακάτω screenshot εμφανίζουμε μέσω ένας update του worker, τον ενημερωμένο μισθό και άλλα στοιχεία του εργαζομένου.

### **OUTPUT CODE ΠΕΡΙΠΤΩΣΗ 1 :**

```
UPDATE worker
SET
    wrk_salary = '1500.00'
WHERE
    wrk_AT = 'AN453827';

SELECT wrk_AT, wrk_salary, wrk_br_code
from worker
WHERE
    wrk_AT = 'AN453827';
```

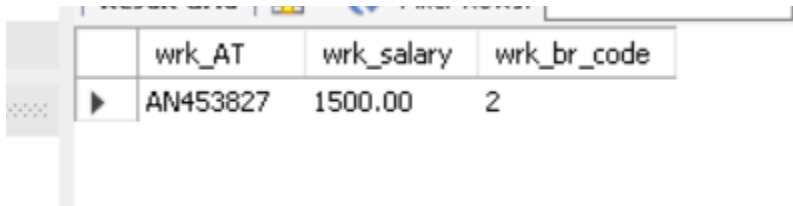
	wrk_AT	wrk_salary	wrk_br_code
▶	AN453827	1500.00	2

## **OUTPUT CODE ΠΕΡΙΠΤΩΣΗ 2 :**

(ΘΕΤΟΥΜΕ ΜΙΣΘΟ <1500 ΚΑΙ ΒΛΕΠΟΥΜΕ ΟΤΙ Ο ΜΙΣΘΟΣ ΔΕΝ ΑΛΛΑΖΕΙ)

```
UPDATE worker
SET
  wrk_salary = '1200.00'
WHERE
  wrk_AT = 'AN453827';

SELECT wrk_AT, wrk_salary, wrk_br_code
  from worker
 WHERE
  wrk_AT = 'AN453827';
```



	wrk_AT	wrk_salary	wrk_br_code
▶	AN453827	1500.00	2

# ΤΕΛΟΣ ΑΝΑΦΟΡΑΣ