ΠΕΡΙΦΕΡΕΙΑ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ 2014-2020

ΕΥΔ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Επενδυτικά Σχέδια Καινοτομίας

Κωδικός πράξης: ΚΜΡ6-0083129

Κωδικός MIS: 5136571

Δικαιούχος: ΑΔΕΛΦΟΙ ΓΕΩΡΓΙΟΥ ΚΟΥΡΤΙΔΗ, ΕΛΚΕ – ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΤΙΤΛΟΣ

«*Βελτιστοποίηση τοποθέτησης και καταμέτρησης εμπορευμάτων σε μεγάλους βιομηχανικούς χώρους με χρήση μη επανδρωμένων αεροσκαφών*»

Τίτλος Πράξης (Αγγλικά) Optimization of placement and counting products in large industrial areas using UAV

Παραδοτέο

Π2.3 Πρόγραμμα σε ελεύθερο λογισμικό ανάλυσης δεδομένων και κανόνων συσχέτισης

Αρ. Παραδοτέου	**Π2.3**
Ενότητα Εργασίας	**ΕΕ2. Ανάπτυξη μεθόδων και συστημάτων οργάνωσης δεδομένων**
Υπεύθυνος Φορέας ΕΕ2 / Π2	**Αδελφοί Γεωργίου Κουρτίδη - ΔΙ.ΠΑ.Ε.**
Είδος Παραδοτέου	**Τεχνική Αναφορά**
Παράδοση	**Σεπτέμβριος 2023**
Σύντομη Περιγραφή Ενότητας	**Βιομηχανική έρευνα στην ανάπτυξη μεθόδων για την οργάνωση δεδομένων, συλλογή δεδομένων από ποικίλες πηγές, δημιουργία πολυδιάστατης αποθήκης δεδομένων, ανάπτυξη συστήματος καταγραφής και συστήματος λήψης αποφάσεων, ανάπτυξη προγραμμάτων σε ελεύθερο λογισμικό για την εφαρμογή των μεθόδων και συστημάτων.**
Έναρξη Προγράμματος, Λήξη	**Οκτώβριος 2021, Δεκέμβριος 2024**
Φορείς	**ΑΔΕΛΦΟΙ ΓΕΩΡΓΙΟΥ ΚΟΥΡΤΙΔΗ,** **ΕΛΚΕ – ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ**

Λίστα συγγραφέων:

#	***Ονοματεπώνυμο***	***Φορέας***	***email***
1	Γεώργιος Κουρτίδης	ΝΕΚΤΑΡ	info@nektar.gr
2	Γεώργιος Κουρτίδης	ΝΕΚΤΑΡ	info@nektar.gr
3	Ειρήνη Κουρτίδου	ΝΕΚΤΑΡ	info@nektar.gr
4	Αναστάσιος Βαζίκογλου	ΝΕΚΤΑΡ	info@nektar.gr
5	Δημήτριος Βαρσάμης	ΔΙ.ΠΑ.Ε.	dvarsam@ihu.gr
6	Αλκιβιάδης Τσιμπίρης	ΔΙ.ΠΑ.Ε.	atsimpiris@ihu.gr
7	Κωνσταντίνος Χειλάς	ΔΙ.ΠΑ.Ε.	chilas@ihu.gr
8	Πάρις Μαστοροκώστας	ΔΙ.ΠΑ.Ε.	mast@uniwa.gr
9	Δημήτριος Μάνος	ΔΙ.ΠΑ.Ε.	dmanos@ihu.gr
10	Ιορδάνης Ζιώγας	ΔΙ.ΠΑ.Ε.	ziogasi@ihu.gr
11	Ηλίας Πανταζής	ΔΙ.ΠΑ.Ε.	hpant@ihu.gr

Επιμέλεια εγγράφου:

#	***Ονοματεπώνυμο***	***Φορέας***	***email***
	Γεώργιος Κουρτίδης	ΝΕΚΤΑΡ	info@nektar.gr
	Δημήτριος Βαρσάμης	ΔΙ.ΠΑ.Ε.	dvarsam@ihu.gr

Ιστορικό εγγράφου:

Εκδοση	**Ημ/νια**	**Είδος εγγράφου**	**Φορέας**
0.1		1st draft	ΔΙ.ΠΑ.Ε.

Περιεχόμενα

1.Περίληψη έργου

Ο σκοπός του παρόντος έργου είναι η ανάπτυξη ενός ολοκληρωμένου πληροφοριακού συστήματος με στόχο την βέλτιστη τοποθέτηση εμπορευματοκιβωτίων σε μεγάλους βιομηχανικούς χώρους. Για την επίτευξη του αρχικά, θα δημιουργηθεί μία κεντρική σχεσιακή βάση δεδομένων η οποία θα ενοποιεί δεδομένα από διαφορετικές πηγές και χρονικές στιγμές, της επιχείρησης. Έπειτα, θα γίνει καθορισμός των απαιτήσεων και μεθόδων για την ανάλυση των δεδομένων της βάσης μέσω της σκιαγράφησης ενός γενικότερου προτύπου εκτέλεσης όλων των διαδικασιών παραγωγής. Η αποθήκη δεδομένων (warehouse), θα λειτουργήσει ως βάση για την ανάπτυξη και εφαρμογή αλγορίθμων που έχουν ως στόχο την εξαγωγή και παρουσίαση Business Analytics υπό μορφή γραφημάτων, πινάκων και δεικτών.

Ακολούθως, θα πραγματοποιηθούν δοκιμαστικές πτήσεις των ειδικών για το έργο ΣμηΕΑ, πρωτίστως για την διαμόρφωσή τους και εν συνεχεία για την καταγραφή της θέσης και της διαθέσιμης ποσότητας του κάθε προϊόντος. Με βάση τα δεδομένα αυτά, θα αναπτυχθούν υπολογιστικές μέθοδοι οι οποίες θα είναι σε θέση να εντοπίζουν το ζητούμενο προϊόν και να υπολογίζουν την διαθεσιμότητά του. Σύμφωνα με τα δεδομένα αυτά, αλλά και με χρήση business analytics, προκύπτουν συμπεράσματα σχετικά με την ζήτηση και την αναγκαία επάρκεια των προϊόντων.

Στη συνέχεια θα αναπτυχθεί μία ειδική μέθοδος βελτιστοποίησης η οποία με δεδομένα όλα τα παραπάνω στοχεύει στην βέλτιστη τοποθέτηση των παραγόμενων προϊόντων στην αποθήκη με βάση συγκεκριμένες προδιαγραφές.

Τέλος, θα αναπτυχθεί ολοκληρωμένο πληροφοριακό σύστημα το οποίο θα εκτελεί αυτόματη καταμέτρηση της αποθήκης και θα προτείνει την βέλτιστη τοποθέτηση προϊόντων βάση προδιαγραφών που θα δίνονται από τον χρήστη. Το σύστημα αυτό, γίνεται πράξη μέσω της ανάπτυξης ολοκληρωμένης πλατφόρμας αλλά και εφαρμογής για έξυπνες κινητές συσκευές, με τη βοήθεια της οποίας οι εργαζόμενοι της επιχείρησης θα μπορούν να έχουν εικόνα της αποθήκης προϊόντων κάθε στιγμή καθώς και την δυνατότητα μετάδοσης της εικόνας αυτής μεταξύ συνεργατών μέσω ηλεκτρονικού ταχυδρομείου ή κινητής τηλεφωνίας για την επίτευξη της καλύτερης δυνατής επικοινωνίας και συνεργασίας αυτών.

Το έργο λοιπόν, υποβοηθώντας στη διαδικασία λήψης αποφάσεων, θα συνδράμει σημαντικά στη μείωση των εξόδων της εταιρίας, θα μειώσει τον φόρτο εργασίας των εργαζομένων απαλλάσσοντάς τους από τετριμμένες διαδικασίες και θα καταστήσει την επιχείρηση ιδιαιτέρως ανταγωνιστική στον χώρο της παραγωγής. Επιπρόσθετα, η αξία ενός τέτοιου προϊόντος δεν περιορίζεται μόνο στο λειτουργικό του κομμάτι αλλά εστιάζεται και στο εμπορικό, καθιστώντας την επιχείρηση που το διαθέτει άκρως ανταγωνιστική στην αγορά εργασίας, καθώς μία τέτοια εφαρμογή προσδίδει υπεραξία στην επιχείρηση αν επιλεγεί η εμπορική της εκμετάλλευσή της.

Το έργο αποτελείται από έξι επιμέρους ενότητες εργασίας, η ολοκλήρωση των οποίων θα οδηγήσει στην επίτευξη του έργου. Οι ενότητες αυτές είναι:

- ΕΕ1 Σχεδίαση υλοποίησης έργου
- ΕΕ2 Ανάπτυξη μεθόδων και συστημάτων οργάνωσης δεδομένων

- ΕΕ3 Ανάπτυξη μεθόδων αυτόματης καταμέτρησης και βέλτιστης τοποθέτησης βιομηχανικών προϊόντων
- ΕΕ4 Ανάπτυξη διαδικτυακής πλατφόρμας και εφαρμογής σε έξυπνες κινητές συσκευές
- ΕΕ5 Πιλοτική λειτουργία και αξιολόγηση ολοκληρωμένου συστήματος
- ΕΕ6 Υποστήριξη έργου

2. Περίληψη Ενότητας Εργασίας 2 (ΕΕ2)

Βιομηχανική έρευνα στην ανάπτυξη μεθόδων για την οργάνωση δεδομένων (ΔΙ.ΠΑ.Ε.)

Κατά την διάρκεια της Ενότητας Εργασίας 2, πρόκειται να συλλεχθούν δεδομένα από διάφορες πηγές της επιχείρησης και αφού οργανωθούν θα ενταχθούν σε πίνακες δημιουργώντας ένα είδος σχήματος χιονονιφάδας. Αυτό το πολυδιάστατο μοντέλο δεδομένων θα απεικονίζει τα δεδομένα ως κύβους και θα αναπτυχθεί σε δύο βασικές κατηγορίες πινάκων: πίνακες διαστάσεων και πίνακες γεγονότων. Οι πίνακες διαστάσεων θα περιέχουν πληροφορίες για τις διαστάσεις του κύβου, ενώ οι πίνακες γεγονότων θα περιλαμβάνουν τα πραγματικά δεδομένα με ξένα κλειδιά προς τους σχετιζόμενους πίνακες διαστάσεων.

Παράλληλα, θα αναπτυχθεί ένα σύστημα καταγραφής της πρώτης ύλης κατά τη διαδικασία παραγωγής προϊόντων. Αυτό το σύστημα θα δώσει τη δυνατότητα να παρακολουθείται η ροή των πρώτων υλών και να εφαρμόζονται αναλύσεις για τη βελτίωση της παραγωγικής διαδικασίας.

Τέλος, θα αναπτυχθούν ειδικά προγράμματα σε ελεύθερο λογισμικό, όπως τη γλώσσα προγραμματισμού Python. Αυτά τα προγράμματα θα εφαρμόζουν τις μεθόδους ανάλυσης δεδομένων και κανόνων συσχέτισης και θα ενσωματώνονται στις τεχνολογίες που θα χρησιμοποιηθούν για την υλοποίηση της πλατφόρμας.

Με αυτόν τον τρόπο, θα δημιουργηθεί ένα ολοκληρωμένο σύστημα που θα βοηθήσει στην ανάλυση και τη λήψη αποφάσεων βασιζόμενες σε δεδομένα, ενισχύοντας την αποτελεσματικότητα της επιχείρησής.

Παραδοτέα:

- Π2.1 Τεχνική αναφορά μοντέλου αποθήκης δεδομένων
- Π2.2 Μέθοδος ανάλυσης δεδομένων και κανόνων συσχέτισης
- Π2.3 Πρόγραμμα σε ελεύθερο λογισμικό (π.χ. Python) ανάλυσης δεδομένων και κανόνων συσχέτισης
- Π2.4 Πρόγραμμα σε ελεύθερο λογισμικό (π.χ. Python) λήψης αποφάσεων μέσα από αναλύσεις και κανόνες συσχέτισης.

3. Πρόγραμμα σε ελεύθερο λογισμικό ανάλυσης δεδομένων και κανόνων συσχέτισης

Παρακάτω παρουσιάζεται αναλυτικά ο κώδικας που χρησιμοποιήθηκε σε γλώσσα Python για την ανάλυση δεδομένων και κανόνων συσχέτισης:

```
# ------------------------------------------------------------------------------
# Check for leap year
# ------------------------------------------------------------------------------
def is_leap_year(yr):
  return (yr % 4 == 0 and yr % 100 != 0) or yr % 400 == 0

# ------------------------------------------------------------------------------
# Get month days for specific year
# Return dictionary with month->days key-value pairs
# ------------------------------------------------------------------------------
def month_days(year):
  if is_leap_year(year):
    feb = 29
  else:
    feb = 28
  return {
    1: 31,
    2: feb,
    3: 31,
    4: 30,
    5: 31,
    6: 30,
    7: 31,
    8: 31,
    9: 30,
    10: 31,
    11: 30,
    12: 31
  }

# ------------------------------------------------------------------------------
# Create the warehouse table
# If already exists delete it first
# ------------------------------------------------------------------------------
def create_warehouse_table(mydb):
  cursor = mydb.cursor(dictionary=False)
  create_sql = '''
    CREATE TABLE warehouse (
      id int(11) unsigned NOT NULL AUTO_INCREMENT,
      product_id int(11) NOT NULL,
      product_name varchar(255) NOT NULL,
      group_id int(11) DEFAULT 0,
      group_name varchar(255) DEFAULT NULL,
      category_id int(11) DEFAULT 0,
      category_name varchar(255) DEFAULT NULL,
      year int(11) DEFAULT NULL,
      month int(11) DEFAULT NULL,
```

```
      week int(11) DEFAULT NULL,
      day int(11) DEFAULT NULL,
      sales_per_day DOUBLE DEFAULT 0,
      sales_per_week DOUBLE DEFAULT 0,
      sales_per_month DOUBLE DEFAULT 0,
      sales_per_year DOUBLE DEFAULT 0,
      PRIMARY KEY (id)
    )
  '''
  print('Creating warehouse table...')
  cursor.execute('DROP TABLE IF EXISTS warehouse')
  cursor.execute(create_sql)
  cursor.close()
  print('warehouse table created!')

# ------------------------------------------------------------------------------
# Create the sales_per_year_view
# ------------------------------------------------------------------------------
def create_sales_per_year_view(mydb):
  print('Creating sales per year view...')
  cursor = mydb.cursor(dictionary=False)
  sql = '''
    CREATE VIEW sales_per_year_view AS
    (
      SELECT
        `year`,
        `productId` product_id,
        sum(amount) amount
      FROM
        sale
      GROUP BY
        `year`,
        `productId`
      ORDER BY
        `year`
    )
  '''
  cursor.execute('DROP VIEW IF EXISTS sales_per_year_view')
  cursor.execute(sql)
  cursor.close()
  print('sales per year view created!')


# ------------------------------------------------------------------------------
# Create the sales_per_month_view
# ------------------------------------------------------------------------------
def create_sales_per_month_view(mydb):
  print('Creating sales per month view...')
  cursor = mydb.cursor(dictionary=False)
  sql = '''
    CREATE VIEW sales_per_month_view AS
    (
      SELECT
        `year`,
        `month`,
        `productId` product_id,
```

```
        sum(amount) amount
      FROM
        sale
      GROUP BY
        `year`,
        `month`,
        `productId`
      ORDER BY
        `year`,
        `month`
    )
  '''
  cursor.execute('DROP VIEW IF EXISTS sales_per_month_view')
  cursor.execute(sql)
  cursor.close()
  print('sales per month view created!')


# ------------------------------------------------------------------------------
# Create the sales_per_week_view
# ------------------------------------------------------------------------------
def create_sales_per_week_view(mydb):
  print('Creating sales per week view...')
  cursor = mydb.cursor(dictionary=False)
  sql = '''
    CREATE VIEW sales_per_week_view AS
    (
      SELECT
        `year`,
        WEEK(DATE(concat_ws("-", year, month, day)), 3) `week`,
        `productId` product_id,
        sum(amount) amount
      FROM
        sale
      GROUP BY
        `year`,
        `week`,
        `productId`
      ORDER BY
        `year`,
        `week`
    )
  '''
  cursor.execute('DROP VIEW IF EXISTS sales_per_week_view')
  cursor.execute(sql)
  cursor.close()
  print('sales per week view created!')


# ------------------------------------------------------------------------------
# Create the sales_per_day_view
# ------------------------------------------------------------------------------
def create_sales_per_day_view(mydb):
  print('Creating sales per day view...')
  cursor = mydb.cursor(dictionary=False)
  sql = '''
```

```
    CREATE VIEW sales_per_day_view AS
    (
      SELECT
        `year`,
        `month`,
        `day`,
        `productId` product_id,
        sum(amount) amount
      FROM
        sale
      GROUP BY
        `year`,
        `month`,
        `day`,
        `productId`
      ORDER BY
        `year`,
        `month`,
        `day`
    )
  '''
  cursor.execute('DROP VIEW IF EXISTS sales_per_day_view')
  cursor.execute(sql)
  cursor.close()
  print('sales per day view created!')


# ------------------------------------------------------------------------------
# Retrieve a list of distinct years from sale table
# ------------------------------------------------------------------------------
def get_years(mydb):
  print('Retrieving available years from sale table...')
  years = []
  cursor = mydb.cursor(dictionary=False)
  cursor.execute('SELECT DISTINCT year FROM sale ORDER BY year')
  result = cursor.fetchall()
  for row in result:
    years.append(row[0])

  # print years
  print("Available years:", end=' ')
  for year in years:
    print(year, end=' ')
  print()

  cursor.close()
  return years

# ------------------------------------------------------------------------------
# Initialize warehouse with products for all calendar years and days
# Retrieve only the products for given groups (list of ids)
# ------------------------------------------------------------------------------
def initialize_warehouse(mydb, groups):
  print(f"Initializing warehouse with products for group ids {groups}...")

  # get availabe years
```

```
years = get_years(mydb)

# Retrieve products for groups
cursor = mydb.cursor(dictionary=True)
group_ids = ','.join(str(e) for e in groups)
sql = f'''
  SELECT
    product.id product_id,
    product.name product_name,
    product.groupId group_id,
    product.categoryId category_id,
    productgroup.name group_name,
    productcategory.name category_name
  FROM
    product,
    productgroup,
    productcategory
  WHERE
    product.id in (SELECT DISTINCT productId FROM sale) AND
    product.groupId = productgroup.id AND
    product.categoryId = productcategory.id AND
    product.groupId in ({group_ids})
'''
cursor.execute(sql)
products = cursor.fetchall()
cursor.close()

# fill warehouse table with products
cursor = mydb.cursor(dictionary=True)
total_products = len(products)
counter = 0

sql = '''
    INSERT INTO warehouse (
      product_id,
      product_name,
      category_id,
      category_name,
      group_id,
      group_name,
      year,
      month,
      day
    )
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
  '''

for product in products:
  counter += 1
  print(f"Product {counter} of {total_products}")
  for year in years:
    mdays = month_days(year)
    for month in range(1, 13):
      for day in range(1, mdays[month]+1):
        cursor.execute(sql, (
          product['product_id'],
```

```
            product['product_name'],
            product['category_id'],
            product['category_name'],
            product['group_id'],
            product['group_name'],
            str(year),
            str(month),
            str(day)
          ))

  # Update week number
  print('Updating week number...')
  cursor.execute('UPDATE warehouse set week = WEEK(DATE(concat_ws("-", year,
month, day)), 3)')
  cursor.close()
  print('Warehouse initialization completed!')


# ------------------------------------------------------------------------------
# update sales_per_year into warehouse
# ------------------------------------------------------------------------------
def update_sales_per_year(mydb):
  print('Updating sales per year...')
  sql = '''
    UPDATE
      warehouse,
      sales_per_year_view
    SET
      warehouse.sales_per_year = sales_per_year_view.amount
    WHERE
      warehouse.product_id = sales_per_year_view.product_id AND
      warehouse.year = sales_per_year_view.year
  '''
  pcursor = mydb.cursor()
  pcursor.execute(sql)
  pcursor.close()
  print('Update sales_per_year completed!')


# ------------------------------------------------------------------------------
# update sales_per_month into warehouse
# ------------------------------------------------------------------------------
def update_sales_per_month(mydb):
  print('Updating sales per month...')
  sql = '''
    UPDATE
      warehouse,
      sales_per_month_view
    SET
      warehouse.sales_per_month = sales_per_month_view.amount
    WHERE
      warehouse.product_id = sales_per_month_view.product_id AND
      warehouse.year = sales_per_month_view.year AND
      warehouse.month = sales_per_month_view.month
  '''
  pcursor = mydb.cursor()
```

```
  pcursor.execute(sql)
  pcursor.close()
  print('Update sales_per_month completed!')

# ------------------------------------------------------------------------------
# update sales_per_day into warehouse
# ------------------------------------------------------------------------------
def update_sales_per_day(mydb):
  print('Updating sales per day...')
  sql = '''
    UPDATE
      warehouse,
      sales_per_day_view
    SET
      warehouse.sales_per_day = sales_per_day_view.amount
    WHERE
      warehouse.product_id = sales_per_day_view.product_id AND
      warehouse.year = sales_per_day_view.year AND
      warehouse.month = sales_per_day_view.month AND
      warehouse.day = sales_per_day_view.day
  '''
  pcursor = mydb.cursor()
  pcursor.execute(sql)
  pcursor.close()
  print('Update sales_per_day completed!')


# ------------------------------------------------------------------------------
# update sales_per_week into warehouse
# ------------------------------------------------------------------------------
def update_sales_per_week(mydb):
  print('Updating sales per week...')
  sql = '''
    UPDATE
      warehouse,
      sales_per_week_view
    SET
      warehouse.sales_per_week = sales_per_week_view.amount
    WHERE
      warehouse.product_id = sales_per_week_view.product_id AND
      warehouse.year = sales_per_week_view.year AND
      warehouse.week = sales_per_week_view.week
  '''
  pcursor = mydb.cursor()
  pcursor.execute(sql)
  pcursor.close()
  print('Update sales_per_week completed!')


# ------------------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------------------

import  mysql.connector
from functions import (
  create_warehouse_table,
```

```
  create_sales_per_year_view,
  create_sales_per_month_view,
  create_sales_per_day_view,
  create_sales_per_week_view,
  initialize_warehouse,
  update_sales_per_year,
  update_sales_per_month,
  update_sales_per_day,
  update_sales_per_week,
)

# ------------------------------------------------------------------------
# Connect to mysql
# ------------------------------------------------------------------------
mydb = mysql.connector.connect(
  host='localhost',
  user='root',
  password='t41206',
  database='nektar'
)

# ------------------------------------------------------------------------
# Create warehouse table
# ------------------------------------------------------------------------
create_warehouse_table(mydb)

# ------------------------------------------------------------------------
# Create sale views
# ------------------------------------------------------------------------
create_sales_per_year_view(mydb)
create_sales_per_month_view(mydb)
create_sales_per_day_view(mydb)
create_sales_per_week_view(mydb)

# ------------------------------------------------------------------------
# Initialize warehouse with products for group ids 1,2, and 8
# ------------------------------------------------------------------------
initialize_warehouse(mydb, [1,2,8])

# ------------------------------------------------------------------------
# update sale amounts into warehouse
# ------------------------------------------------------------------------
update_sales_per_year(mydb)
update_sales_per_month(mydb)
update_sales_per_day(mydb)
update_sales_per_week(mydb)

# ------------------------------------------------------------------------
# Commit transaction and close the database
# ------------------------------------------------------------------------
mydb.commit()
mydb.close()
```

4. Ανάλυση Κώδικα Python

Έλεγχος Δίσεκτου Έτους

Η συνάρτηση `is_leap_year(yr)` ελέγχει αν το έτος είναι δίσεκτο. Αυτό επιτυγχάνεται μέσω των ακόλουθων κανόνων:

Ένα έτος είναι δίσεκτο αν διαιρείται ακριβώς με το 4 και δεν διαιρείται ακριβώς με το 100.

Εξαίρεση: Αν το έτος διαιρείται ακριβώς με το 400, τότε είναι δίσεκτο.

Ημέρες ανά Μήνα για Συγκεκριμένο Έτος

Η συνάρτηση `month_days(year)` επιστρέφει ένα λεξικό με τους μήνες και τις αντίστοιχες ημέρες για το δοσμένο έτος. Αν το έτος είναι δίσεκτο, ο Φεβρουάριος έχει 29 ημέρες, διαφορετικά έχει 28.

Δημιουργία Πίνακα *warehouse*

Η συνάρτηση `create_warehouse_table(mydb)` δημιουργεί τον πίνακα *warehouse*. Αν υπάρχει ήδη, τον διαγράφει πριν τον ξαναδημιουργήσει. Ο πίνακας περιλαμβάνει πεδία για τα προϊόντα, τις πωλήσεις και χρονικές μεταβλητές όπως έτος, μήνας, εβδομάδα και ημέρα.

Δημιουργία *views* για τις Πωλήσεις

Οι ακόλουθες συναρτήσεις δημιουργούν προβολές (*views*) που συγκεντρώνουν τα δεδομένα πωλήσεων:

- `create_sales_per_year_view(mydb)` : Δημιουργεί προβολή συνολικών πωλήσεων ανά έτος.
- `create_sales_per_month_view(mydb)` : Δημιουργεί προβολή συνολικών πωλήσεων ανά μήνα.
- `create_sales_per_week_view(mydb)` : Δημιουργεί προβολή συνολικών πωλήσεων ανά εβδομάδα.
- `create_sales_per_day_view(mydb)` : Δημιουργεί προβολή συνολικών πωλήσεων ανά ημέρα.

Ανάκτηση Διαθέσιμων Ετών από τον Πίνακα *sale*

Η συνάρτηση `get_years(mydb)` ανακτά τα διαθέσιμα έτη από τον πίνακα *sale* και τα επιστρέφει ως λίστα.

Αρχικοποίηση Αποθήκης με Προϊόντα και Ημερομηνίες

Η συνάρτηση `initialize_warehouse(mydb, groups)`:

- Ανακτά τα διαθέσιμα έτη από τον πίνακα *sale*.
- Ανακτά τα προϊόντα που ανήκουν στις καθορισμένες ομάδες (*groups*).
- Δημιουργεί εγγραφές στον πίνακα *warehouse* για κάθε συνδυασμό προϊόντος-έτους-μήνα-ημέρας.
- Υπολογίζει τον αριθμό εβδομάδας για κάθε εγγραφή.

Ενημέρωση Πωλήσεων στον Πίνακα warehouse

Οι συναρτήσεις ενημερώνουν τα δεδομένα πωλήσεων στον πίνακα *warehouse* από τις αντίστοιχες *views*:

- `update_sales_per_year(mydb)` : Ενημερώνει τις ετήσιες πωλήσεις.
- `update_sales_per_month(mydb)` : Ενημερώνει τις μηνιαίες πωλήσεις.
- `update_sales_per_day(mydb)` : Ενημερώνει τις ημερήσιες πωλήσεις.
- `update_sales_per_week(mydb)` : Ενημερώνει τις εβδομαδιαίες πωλήσεις.

Κύριο Πρόγραμμα

Το κύριο πρόγραμμα εκτελεί τα παρακάτω βήματα:

- Συνδέεται με τη βάση δεδομένων MySQL.
- Δημιουργεί τον πίνακα *warehouse*.
- Δημιουργεί τις *views* για τις πωλήσεις.
- Αρχικοποιεί την αποθήκη με προϊόντα για συγκεκριμένες ομάδες (1,2,8).
- Ενημερώνει τις πωλήσεις στον πίνακα *warehouse*.
- Κάνει *commit* και κλείνει τη σύνδεση με τη βάση δεδομένων.

Σύνοψη

Ο κώδικας δημιουργεί και διαχειρίζεται μια αποθήκη δεδομένων που καταγράφει και ενημερώνει τις πωλήσεις προϊόντων με βάση χρονικές παραμέτρους (έτος, μήνας, εβδομάδα, ημέρα). Η ανάλυση δεδομένων μπορεί να βοηθήσει στη βελτιστοποίηση της αποθήκευσης και στη λήψη επιχειρηματικών αποφάσεων.

5. Κατάλογος όρων και συντομογραφιών

ΣμηΕΑ	Συστήματα μη Επανδρωμένων Αεροσκαφών
ΔΙ.ΠΑ.Ε.	Διεθνές Πανεπιστήμιο της Ελλάδος
ΠΑ.Δ.Α.	Πανεπιστήμιο Δυτικής Αττικής
Δ.Ε.Π.	Διδακτικό Ερευνητικό Προσωπικό
Ε.ΔΙ.Π.	Εργαστηριακό Διδακτικό Προσωπικό
Α.Ε.Ι.	Ανώτατο Εκπαιδευτικό Ίδρυμα
Δ.Ο.Α.Τ.Α.Π.	Διεπιστημονικός Οργανισμός Αναγνώρισης Τίτλων Ακαδημαϊκών και πληροφόρησης