Βάσεις Δεδομένων και Ευφυή Πληροφοριακά Συστήματα Επιχειρηματικότητας

3ο Μάθημα: Εισαγωγή στην SQL

Δρ. Κωνσταντίνος Χ. Γιωτόπουλος

SQL Background

- SQL
  - Structured Query Language
  - Standard query γλώσσα για εμπορικά relational DBMSs
  - Κάθε DBMS έχει και τα δικά του extensions
- Ιστορική εξέλιξη
  - SEQUEL at IBM (1974) on System R
  - IMB DB2 SQL (1983)
  - SQL (SQL1 or SQL-86): first standard version by ANSI and ISO
  - SQL2 (SQL-92): more DDL/DML features
  - SQL3 (SQL-99): object-oriented concepts

Γενικά

- Για να μπορέσουμε να δημιουργήσουμε και να διαχειριστούμε μια βάση δεδομένων, μπορούμε να χρησιμοποιήσουμε ειδικές γλώσσες προγραμματισμού, τις λεγόμενες *γλώσσες ερωταπαντήσεων (query languages)*.
- Είναι γλώσσες μη διαδικαστικές, τέταρτης γενιάς (4th generation languages).
- Εμείς απλά διατυπώνουμε με απλές και κατανοητές εντολές το τι πληροφορίες ζητάμε και το ΣΔΒΔ (Σύστημα Διαχείρισης Βάσεων Δεδομένων) αναλαμβάνει να μας απαντήσει.
- Η *SQL (Structured Query Language, δηλ. Δομημένη Γλώσσα Ερωταπαντήσεων)* είναι σήμερα η πιο δημοφιλής και πιο διαδεδομένη γλώσσα ανάπτυξης και διαχείρισης σχεσιακών βάσεων δεδομένων.

Γενικά (2)

- Η SQL αποτελείται από εντολές με τα ορίσματά τους, τις οποίες μπορούμε να χρησιμοποιήσουμε με συγκεκριμένους κανόνες σύνταξης για να πάρουμε τα αποτελέσματα που θέλουμε.
- Με την SQL μπορούμε να
  - δημιουργήσουμε μια βάση δεδομένων και τους πίνακές της με τα αντίστοιχα πεδία,
  - να καταχωρήσουμε δεδομένα στους πίνακες,
  - να τροποποιήσουμε και να διαγράψουμε τα δεδομένα αυτά,
  - να αλλάξουμε τη δομή των πινάκων με προσθήκη και διαγραφή πεδίων και
  - να εμφανίσουμε πληροφορίες (συνδυασμούς από δεδομένα).

Γενικά (3)

- Τη *Γλώσσα Ορισμού Δεδομένων (DDL, Data Definition Language)*, η οποία περιέχει τις απαραίτητες εντολές για τον ορισμό και την τροποποίηση του σχεσιακού σχήματος καθώς και για τη δημιουργία, την τροποποίηση και τη διαγραφή σχέσεων. Περιέχει ακόμη τις εντολές δημιουργίας και επεξεργασίας όψεων και ορισμού περιορισμών ακεραιότητας.
- Τη *Γλώσσα Χειρισμού Δεδομένων (DML, Data Manipulation Language)*, η οποία περιέχει τις απαραίτητες εντολές για την εμφάνιση (αναζήτηση) δεδομένων καθώς και για την καταχώρηση, τροποποίηση και διαγραφή των εγγραφών (πλειάδων) μιας σχέσης.
- Τέλος, περιέχει εντολές για τον ορισμό και την επεξεργασία *συναλλαγών (transactions)*.

Τύποι Δεδομένων

- *char(n)*, ένα αλφαριθμητικό (string) με n ακριβώς χαρακτήρες.
- *varchar(n)*, ένα αλφαριθμητικό (string) με μεταβλητό μήκος και με n το πολύ χαρακτήρες.
- *int*, ακέραιος αριθμός.
- *smallint*, ακέραιος αριθμός με μικρές τιμές.
- *Numeric(p, d)*, αριθμός με p ψηφία, από τα οποία τα d είναι δεκαδικά.
- *real*, αριθμός κινητής υποδιαστολής απλής ακρίβειας.
- *double precision*, αριθμός κινητής υποδιαστολής διπλής ακρίβειας.
- *float(n)*, αριθμός κινητής υποδιαστολής με ακρίβεια n ψηφίων.
- *date*, ημερομηνία (ημέρα, μήνας, έτος).
- *time*, ώρα (ώρα, λεπτά, δευτερόλεπτα).

SQL Create Table

Η CREATE TABLE χρησιμοποιείται για τη δημιουργία πινάκων στη βάση δεδομένων.
Η σύνταξη της **CREATE TABLE** είναι:

```
CREATE TABLE "table_name"
("column 1" "data_type_for_column_1",
"column 2" "data_type_for_column_2",
... )
```

Αν θέλουμε να δημιουργήσουμε έναν πίνακα θα έχουμε:

```
CREATE TABLE customer
(First_Name char(50),
Last_Name char(50),
Address char(50),
City char(50),
Country char(25),
Birth_Date date)
```

Μερικές φορές θέλουμε να βάλουμε μια προεπιλεγμένη τιμή σε ένα πεδίο. Παράδειγμα:

```
CREATE TABLE customer
(First_Name char(50),
Last_Name char(50),
Address char(50) default 'Unknown',
City char(50) default 'Mumbai',
Country char(25),
Birth_Date date)
```

Constraints – Περιορισμοί

- - **NOT NULL**
  - **UNIQUE**
  - **CHECK**
  - **Primary Key**
  - **Foreign Key**

- **NOT NULL**
- Εξ ορισμού ένα πεδίο μπορεί να μην έχει καμία τιμή σε μια εγγραφή του πίνακα. Αν θέλουμε να επιβάλουμε σε ένα πεδίο να μη δέχεται κενές τιμές (δηλαδή να μην δέχεται τη μη εισαγωγή δεδομένων, υποχρεωτικά να έχει κάποια τιμή) τότε καθορίζουμε το συγκεκριμένο πεδίο ως not null.
- **CREATE TABLE Customer**
  (SID integer NOT NULL,
  Last_Name varchar (30) NOT NULL,
  First_Name varchar(30));

- **UNIQUE**
- Ο περιορισμός UNIQUE διασφαλίζει ότι όλες οι τιμές σε ένα πεδίο είναι διακριτές δηλαδή μοναδικές.
- **CREATE TABLE Customer**
  (SID integer Unique,
  Last_Name varchar (30),
  First_Name varchar(30));

- **CHECK**
- Ο περιορισμός CHECK διασφαλίζει ότι όλες οι τιμές που θα εισαχθούν σε ένα πεδίο ικανοποιούν συγκεκριμένα κριτήρια.
- **CREATE TABLE Customer**
  (SID integer CHECK (SID > 0),
  Last_Name varchar (30),
  First_Name varchar(30));
- Το πεδίο SID πρέπει να πάρει ακέραιες τιμές και μάλιστα θετικές (>0)

Primary Key – Πρωτεύον Κλειδί

- Ένα πρωτεύον κλειδί χρησιμοποιείται για την ταυτοποίηση κάθε εγγραφής σε ένα πίνακα. Το πρωτεύον κλειδί μπορεί να είναι ένα ή περισσότερα πεδία σε ένα πίνακα. Όταν χρησιμοποιούνται πολλά τότε έχουμε σύνθετο κλειδί.
- Το πρωτεύον κλειδί καθορίζεται είτε στο CREATE TABLE ερώτημα ή χρησιμοποιώντας το ALTER TABLE μετά τη δημιουργία του πίνακα.

MySQL:

```
CREATE TABLE Customer
(SID integer,
Last_Name varchar(30),
First_Name varchar(30),
PRIMARY KEY (SID));
```

Primary Key – Πρωτεύον Κλειδί

- Ένα πρωτεύον κλειδί χρησιμοποιείται για την ταυτοποίηση κάθε εγγραφής σε ένα πίνακα. Όταν χρησιμοποιούνται πολλά πεδία τότε έχουμε σύνθετο κλειδί.
- Το πρωτεύον κλειδί καθορίζεται είτε στο CREATE TABLE ερώτημα ή χρησιμοποιώντας το ALTER TABLE μετά τη δημιουργία του πίνακα.

MySQL:

```
CREATE TABLE Customer
(SID integer,
Last_Name varchar(30),
First_Name varchar(30),
PRIMARY KEY (SID));
```

Με το ALTER TABLE

MySQL:

ALTER TABLE Customer ADD PRIMARY KEY (SID);

Foreign Key – Ξένο Κλειδί

- Ένα ξένο κλειδί είναι ένα πεδίο (πεδία) τα οποία σχετίζονται με το πρωτεύον κλειδί ενός άλλου πίνακα. Ο σκοπός του ξένου κλειδιού είναι να διασφαλίσει την σχεσιακή ακεραιότητα των δεδομένων.
- Σε αυτό το παράδειγμα το πεδίο Customer_SID στον πίνακα ORDERS είναι ένα ξένο κλειδί που δείχνει στο πρωτεύον κλειδί του πίνακα CUSTOMERS που είναι το πεδίο SID.
- **MySQL**:
CREATE TABLE ORDERS
(Order_ID integer,
Order_Date date,
Customer_SID integer,
Amount double,
Primary Key (Order_ID),
Foreign Key (Customer_SID)
references CUSTOMER(SID));

Table CUSTOMER

Column name	Characteristic
SID	Primary Key
Last_Name	
First_Name	

Table ORDERS

Column name	Characteristic
Order_ID	Primary Key
Order_Date	
Customer_SID	Foreign Key
Amount	

ALTER TABLE ORDERS
ADD FOREIGN KEY (customer_sid)
REFERENCES CUSTOMER(SID);

Alter Table

Μετά τη δημιουργία ενός πίνακα είναι δυνατή η αλλαγή της δομής του πίνακα χρησιμοποιώντας την εντολή ALTER TABLE. Κλασικές τέτοιες περιπτώσεις είναι οι κάτωθι:

- Add a column
- Drop a column
- Change a column name
- Change the data type for a column

Οι προαναφερθείσες περιπτώσεις δεν είναι οι μοναδικές. Άλλες τέτοιες περιπτώσεις είναι η αλλαγή του πρωτεύοντος κλειδιού, η προσθήκη περιορισμών κλπ.

Η SQL σύνταξη για την **ALTER TABLE** είναι

ALTER TABLE "table_name"
[alter specification]

Add a column: ADD "column 1" "data type for column 1"

Drop a column: DROP "column 1"

Change a column name: CHANGE "old column name" "new column name" "data type for new column name"

Change the data type for a column: MODIFY "column 1" "new data type"

Drop - Truncate

Drop Table

Το ερώτημα DROP TABLE διαγράφει εντελώς από τη βάση δεδομένων τον πίνακα που θέλουμε.

DROP TABLE "table_name"

Αν ήθελα να διαγράψω τον πίνακα customer:

DROP TABLE customer.

Truncate Table

Στην περίπτωση που θέλω να διαγράψω μόνο τα περιεχόμενα του πίνακα και να κρατήσω τη δομή του πίνακα χρησιμοποιώ την εντολή TRUNCATE TABLE.

TRUNCATE TABLE "table_name"

Αν ήθελα να διαγράψω τα περιεχόμενα του πίνακα Customer:

TRUNCATE TABLE customer

SQL SELECT

Η εντολή **SQL SELECT** χρησιμοποιείται για την επιλογή δεδομένων από έναν πίνακα σε μια βάση δεδομένων.

Μια γενική μορφή σύνταξης για το ερώτημα **SQL SELECT** είναι η ακόλουθη:

```
SELECT πεδίο 1, πεδίο 2, πεδίο 3, …
FROM Table1
```

Η λίστα με τα ονόματα των πεδίων μετά την εντολή **SQL SELECT** καθορίζει ποια πεδία θα επιστραφούν προς εμφάνιση ως αποτέλεσμα της εκτέλεσης του ερωτήματος.. Εάν θέλουμε να επιλέξουμε όλα τα πεδία του πίνακα χρησιμοποιούμε το * (αστεράκι):

```
SELECT *
FROM Table1
```

Το όνομα του πίνακα καθορίζεται μετά τη λέξη FROM (στο παράδειγμα Table1) καθορίζει τον πίνακα προέλευσης των δεδομένων.

SQL SELECT INTO

SQL SELECT INTO

Η εντολή **SQL SELECT INTO** χρησιμοποιείται για την επιλογή δεδομένων από έναν πίνακα και εισαγωγή των δεδομένων σε έναν διαφορετικό πίνακα της βάσης.

Μια γενική μορφή σύνταξης είναι η ακόλουθη:

```
SELECT πεδίο 1, πεδίο 2, πεδίο 3,
INTO Table2
FROM Table1
```

Η λίστα με τα πεδία μετά το SELECT καθορίζει ποια πεδία θα αντιγραφούν και το όνομα του πίνακα μετά το INTO καθορίζει τον πίνακα που θα αντιγραφούν τα δεδομένα..

Αν θέλουμε να φτιάξουμε ένα ακριβές αντίγραφο των δεδομένων του πίνακα Customers, χρειαζόμαστε το κάτωθι ερώτημα **SQL SELECT INTO**:

```
SELECT *
INTO Customers_copy
FROM Customers
```

SQL DISTINCT

Η εντολή SQL DISTINCT χρησιμοποιείται μαζί την εντολή SELECT για να επιστρέψει ένα σύνολο δεδομένων με μοναδικές εγγραφές ενός πεδίου του πίνακα.

Θα χρησιμοποιήσουμε τον κάτωθι πίνακα Customers.

FirstName	LastName	Email	DOB	Phone
John	Smith	John.Smith@yahoo.com	2/4/1968	626 222-2222
Steven	Goldfish	goldfish@fishhere.net	4/4/1974	323 455-4545
Paula	Brown	pb@herowndomain.org	5/24/1978	416 323-3232
James	Smith	jim@supergig.co.uk	20/10/1980	416 323-8888

Παράδειγμα: θέλω να εμφανίσω τα μοναδικά επίθετα από το πεδίο surnames του πίνακα Customers, θα εφαρμόσω την ακόλουθη εντολή SQL DISTINCT:

SELECT DISTINCT LastName FROM Customers ;

LastName
Smith
Goldfish
Brown

SQL WHERE

Η εντολή **SQL WHERE** χρησιμοποιείται για την εφαρμογή κριτηρίων σε ένα ερώτημα SQL SELECT. Θα χρησιμοποιήσουμε τον πίνακα **Customers** για ένα παράδειγμα

FirstName	LastName	Email	DOB	Phone
John	Smith	John.Smith@yahoo.com	2/4/1968	626 222-2222
Steven	Goldfish	goldfish@fishhere.net	4/4/1974	323 455-4545
Paula	Brown	pb@herowndomain.org	5/24/1978	416 323-3232
James	Smith	jim@supergig.co.uk	20/10/1980	416 323-8888

Θέλουμε να επιλέξουμε να εμφανισθούν όλοι οι πελάτες μας από τον πίνακα δεδομένων που το επίθετό τους είναι 'Smith'. Το ερώτημα που θα δημιουργήσουμε είναι το κάτωθι:

```
SELECT *
FROM Customers
WHERE LastName = 'Smith'
```

Σε αυτό το απλό ερώτημα χρησιμοποιήσαμε το "=" (Ίσον) στα κριτήρια του WHERE: LastName = 'Smith'

SQL WHERE

Μπορούμε όμως να χρησιμοποιήσουμε και τους υπόλοιπους τελεστές σύγκρισης σε ένα ερώτημα που έχει **SQL WHERE**:

<>(Διάφορο)
SELECT * FROM Customers WHERE LastName <> 'Smith‘

> (Μεγαλύτερο από)
SELECT *FROM Customers WHERE DOB > '1/1/1970‘

>= (Μεγαλύτερο ή ίσον)
SELECT * FROM Customers WHERE DOB >= '1/1/1970‘

< (Μικρότερο από)
SELECT *FROM Customers WHERE DOB < '1/1/1970‘

<= (Μικρότερο ή ίσον)
SELECT * FROM Customers WHERE DOB =< '1/1/1970‘

LIKE (μοιάζει με)
SELECT * FROM Customers WHERE Phone LIKE '626%‘
Σημείωση: Η σύνταξη του LIKE διαφοροποιείται ανάλογα με το Σύστημα Διαχείρισης Βάσης Δεδομένων που χρησιμοποιούμε.

Between (Καθορίζει ένα εύρος περιοχής)
SELECT * FROM Customers WHERE DOB BETWEEN '1/1/1970' AND '1/1/1975'

SQL LIKE

Κριτήριο αναζήτησης σε ένα SQL WHERE, βασιζόμενοι σε **ένα μέρος του περιεχομένου του πεδίου**. Θέλουμε να αναζητήσουμε όλους τους πελάτες που το FirstName ξεκινάει από 'J'.

```
SELECT *
FROM Customers
WHERE FirstName LIKE 'J%‘
```

Αν θέλουμε να εμφανίσουμε όλους τους πελάτες που το τηλέφωνό τους ξεκινάει από '416' θα δημιουργήσουμε το ακόλουθο ερώτημα:

```
SELECT *
FROM Customers
WHERE Phone LIKE '416%'
```

Το '%' είναι χαρακτήρας που αντιπροσωπεύει οποιοδήποτε αλφαριθμητικό.

Ένας άλλος χαρακτήρας που αντιπροσωπεύει έναν οποιοδήποτε χαρακτήρα είναι το '_'. Η χρήση του σημαίνει ένα οποιοδήποτε αλφαριθμητικό που αποτελείται από έναν μόνο χαρακτήρα.

Το '[]' καθορίζει ένα σύνολο από χαρακτήρες.

```
SELECT *
FROM Customers
WHERE Phone LIKE '[4-6]_6%‘
```

Το ερώτημα θα επιστρέψει τους πελάτες που ικανοποιούν τα κάτωθι κριτήρια:

- Το πεδίο Phone ξεκινά από ένα ψηφίο μεταξύ 4 και 6 ([4-6])
- Ο δεύτερος χαρακτήρας στο πεδίο Phone είναι οτιδήποτε (_)
- Ο τρίτος χαρακτήρας στο πεδίο Phone είναι το 6 (6)
- Το υπόλοιπο μέρος του πεδίου Phone column μπορεί να είναι οποιοδήποτε αλφαριθμητικό (%)

SQL INSERT INTO

Ο πρώτος τρόπος σύνταξης του INSERT INTO SQL ερωτήματος δεν καθορίζει τα ονόματα των πεδίων αλλά μόνο τις τιμές τους:

```
INSERT INTO Table1
VALUES (value1, value2, value3…);
```

Ο δεύτερος τρόπος σύνταξης του **SQL INSERT INTO** ερωτήματος, τόσο τα ονόματα των πεδίων αλλά και τις αντίστοιχες τιμές των πεδίων:

```
INSERT INTO Table1 (Column1, Column2, Column3…)
VALUES (Value1, Value2, Value3…);
```

Θα πρέπει ο αριθμός των πεδίων που εισάγετε στην παρένθεση να είναι ακριβώς ο ίδιος με των αριθμό των τιμών που θα εισαχθούν στα αντίστοιχα πεδία αλλιώς θα προκληθεί σφάλμα στην εκτέλεση του ερωτήματος..

Αν θέλουμε να εισάγουμε μια νέα εγγραφή στον πίνακα Customers, θα χρησιμοποιήσουμε μια από τις ακόλουθες εντολές:

```
INSERT INTO Customers
VALUES ('Peter', 'Hunt', 'peter.hunt@tgmail.net', '1/1/1974', '626 888-8888')
```

```
INSERT INTO Customers (FirstName, LastName, Email, DOB, Phone)
VALUES ('Peter', 'Hunt', 'peter.hunt@tgmail.net', '1/1/1974', '626 888-8888')
```

SQL UPDATE

```
UPDATE Table1
SET Column1 = Value1, Column2 = Value2
WHERE Some_Column = Some_Value
```

Η **SQL UPDATE** αλλάζει τα δεδομένα σε μια υπάρχουσα εγγραφή (ή πολλές εγγραφές) σε μια βάση δεδομένων και συνήθως τη χρησιμοποιούμε μαζί με κάποια κριτήρια (προσθέτοντας και το αντίστοιχο WHERE).

Η **SQL UPDATE** αλλάζει τα δεδομένα σε μια υπάρχουσα εγγραφή (ή πολλές εγγραφές) σε μια βάση δεδομένων και συνήθως τη χρησιμοποιούμε μαζί με κάποια κριτήρια (προσθέτοντας και το αντίστοιχο WHERE). Στο WHERE καθορίζουμε ποιες από τις εγγραφές του πίνακα πρέπει να αλλάξουν τιμή (αυτές που ικανοποιούν τα κριτήρια του WHERE). Στο WHERE καθορίζουμε ποιες από τις εγγραφές του πίνακα πρέπει να αλλάξουν τιμή (αυτές που ικανοποιούν τα κριτήρια του WHERE).

SQL DELETE

```
DELETE FROM Table1
WHERE Some_Column = Some_Value
```

Αν δεν εισάγουμε το WHERE και τα κριτήρια , τότε θα διαγραφούν όλες οι εγγραφές από τον πίνακα.

Εφόσον καθορίσουμε συγκεκριμένα κριτήρια στο WHERE τότε θα διαγραφούν από τον πίνακα οι εγγραφές εκείνες που ικανοποιούν τα κριτήρια αυτά.

SQL ORDER BY

```
SELECT * FROM Customers
ORDER BY DOB;
```

Στη περίπτωση που θέλουμε φθίνουσα ταξινόμηση πρέπει να το δηλώσουμε προσθέτοντας στο τέλος του τμήματος του ORDER BY τη λέξη DESC

Μπορούμε να ταξινομήσουμε τα αποτελέσματα ενός ερωτήματος χρησιμοποιώντας περισσότερα του ενός πεδία:

```
SELECT * FROM Customers
ORDER BY DOB, LastName
```

Στην περίπτωση αυτή η ταξινόμηση γίνεται με βάση το πρώτο κατά σειρά που βάλαμε στο ORDER BY πεδίο και εφόσον υπάρξει κάποιο πρόβλημα (π. χ. Οι τιμές των πεδίων είναι ίδιες) τότε η ταξινόμηση μόνο για αυτά που έχουν το πρόβλημα συνεχίζει στο δεύτερο πεδίο κ.ο.κ.

SQL AND & OR

Το **SQL AND** χρησιμοποιείται όταν θέλουμε να καθορίσουμε περισσότερα του ενός κριτήρια και θέλουμε ταυτόχρονα να είναι τα κριτήρια αληθή.

```
SELECT * FROM Customers
WHERE FirstName = 'John' AND LastName = 'Smith'
```

Το **SQL OR** χρησιμοποιείται με παρόμοιο τρόπο και η διαφορά του με το AND είναι ότι το OR επιστρέφει τις εγγραφές που ικανοποιούν τουλάχιστον ένα από τα κριτήρια που εισαχθήκανε στο WHERE.

```
SELECT * FROM Customers
WHERE FirstName = 'James' OR FirstName = 'Paula'
```

Μπορούμε φυσικά να συνδυάσουμε το AND και το OR με οποιοδήποτε τρόπο θέλουμε και να χρησιμοποιήσουμε και παρενθέσεις και να δημιουργηθούν με το τρόπο αυτό πολύπλοκα κριτήρια στο WHERE

```
SELECT * FROM Customers
WHERE (FirstName = 'James' OR FirstName = 'Paula') AND LastName = 'Brown'
```

SQL IN

Το **SQL IN** μας επιτρέπει να καθορίσουμε διακριτές τιμές στα κριτήρια αναζήτησης σε ένα SQL WHERE

```
SELECT Column1, Column2, Column3, …
FROM Table1
WHERE Column1 IN (Value1, Value2, …)
```

```
SELECT *
FROM EmployeeHours
WHERE Date IN ('5/6/2004', '5/7/2004')
```

SQL BETWEEN

Το **SQL BETWEEN & AND** καθορίζουν ένα εύρος τιμών μεταξύ μιας αρχικής και μιας τελικής τιμής.

```
SELECT Column1, Column2, Column3, …
FROM Table1
WHERE Column1 BETWEEN Value1 AND Value2
```

Οι δύο τιμές καθορίζουν το εύρος τιμών και μπορεί να είναι ημερομηνίες, αριθμοί ή απλώς κείμενο.

Σε αντίθεση με το SQL IN keyword, το οποίο επιτρέπει τον καθορισμό μόνο διακριτών τιμών στα κριτήρια, το **SQL BETWEEN** δίνει τη δυνατότητα να καθορίσετε ένα εύρος τιμών στα κριτήρια αναζήτησης.

SQL ALIASES

Τα **SQL aliases** (ψευδώνυμα) χρησιμοποιούνται για πίνακες και για πεδία πινάκων.

Τα ψευδώνυμα των πεδίων χρησιμοποιούνται για να γίνονται τα αποτελέσματα των ερωτημάτων πιο κατανοητά και ευανάγνωστα

```
SELECT Employee, SUM(Hours) As SumHoursPerEmployee
FROM EmployeeHours
GROUP BY Employee
```

SQL COUNT

Η εντολή **SQL COUNT** είναι μια αθροιστική συνάρτηση και χρησιμοποιείτε για την καταμέτρηση του αριθμού των εγγραφών που επιστρέφονται από ένα ερώτημα επιλογής.

```
SELECT COUNT(Column1)
FROM Table1
```

Αν θέλουμε να μετρήσουμε τον αριθμό των πελατών στον πίνακα Customers θα χρησιμοποιήσαμε το κάτωθι ερώτημα

```
SELECT COUNT(LastName) AS NumberOfCustomers
FROM Customers
```

SQL MAX - MIN

Η συνάρτηση **SQL MAX - MIN** μας επιτρέπει την επιλογή της μέγιστης / ελάχιστης τιμής από ένα πεδίο.

```
SELECT MAX(Column1)
FROM Table1
```

```
SELECT MIN(Column1)
FROM Table1
```

```
SELECT MAX(DOB) AS MaxDOB
FROM Customers
```

```
SELECT MIN(DOB) AS MinDOB
FROM Customers
```

SQL AVG

Η **SQL AVG** συνάρτηση μας επιτρέπει την εξαγωγή του μέσου όρου των τιμών ενός πεδίου.

```
SELECT AVG(Column1)
FROM Table1
```

Sales:

CustomerID	Date	SaleAmount
2	5/6/2004	$100.22
1	5/7/2004	$99.95
3	5/7/2004	$122.95
3	5/13/2004	$100.00
4	5/22/2004	$555.55

```
SELECT AVG(SaleAmount) AS AvgSaleAmount FROM Sales;
```

AvgSaleAmount
$195.73

SQL SUM

Η **SQL SUM** συνάρτηση μας επιτρέπει να υπολογίζουμε το άθροισμα των τιμών ενός αριθμητικού πεδίου

```
SELECT SUM(Column1)
FROM Table1
```

Sales:

CustomerID	Date	SaleAmount
2	5/6/2004	$100.22
1	5/7/2004	$99.95
3	5/7/2004	$122.95
3	5/13/2004	$100.00
4	5/22/2004	$555.55

```
SELECT SUM(SaleAmount)
FROM Sales
WHERE CustomerID = 3
```

SaleAmount
$222.95

```
SELECT SUM(SaleAmount)
FROM Sales;
```

SaleAmount
$195.73

SQL GROUP BY

Το **SQL GROUP BY** χρησιμοποιείται μαζί με SQL αθροιστικές ή άλλες συναρτήσεις (τις είδαμε προηγουμένως) όπως η SUM για να καθορίσει τρόπους ομαδοποίησης των αποτελεσμάτων βάση κάποιου πεδίου (ή κάποιων πεδίων).

Αν ο διευθυντής της επιχείρησης θέλει το άθροισμα των ωρών εργασίας των εργαζομένων πρέπει να εκτελέσει το κάτωθι ερώτημα

SELECT SUM (Hours)
FROM EmployeeHours;

Το άθροισμα των ωρών εργασίας ανά εργαζόμενο ξεχωριστά; Τότε πρέπει να αλλάξουμε το ερώτημα και να χρησιμοποιήσουμε το SQL GROUP BY:

SELECT Employee, SUM (Hours)
FROM EmployeeHours
GROUP BY Employee**;**

πίνακα EmployeeHours

Employee	Date	Hours
John Smith	5/6/2004	8
Allan Babel	5/6/2004	8
Tina Crown	5/6/2004	8
John Smith	5/7/2004	9
Allan Babel	5/7/2004	8
Tina Crown	5/7/2004	10
John Smith	5/8/2004	8
Allan Babel	5/8/2004	8
Tina Crown	5/8/2004	9

Employee	Hours
John Smith	25
Allan Babel	24
Tina Crown	27

SQL HAVING

Το **SQL HAVING** χρησιμοποιείται για να περιορίσει βάση συνθηκών την έξοδο ενός ερωτήματος θέτοντας μια συνάρτηση σε ένα πεδίο του ερωτήματος.

```
SELECT Employee, SUM (Hours)
FROM EmployeeHours
WHERE SUM (Hours) > 24
GROUP BY Employee
```

SQL JOIN

- Το **SQL JOIN** χρησιμοποιείται όταν θέλουμε να επιλέξουμε δεδομένα που προέρχονται από δύο ή περισσότερους πίνακες.
- Θα πρέπει οι πίνακες αυτοί να σχετίζονται μεταξύ τους μέσω κάποιας συσχέτισης πεδίων τους.

Customers

CID	Fname	Lname	Email	DOB	Phone
1	John	Smith	John.Smith@yahoo.com	2/4/1968	626 222-2222
2	Steven	Goldfish	goldfish@fishhere.net	4/4/1974	323 455-4545
3	Paula	Brown	pb@herowndomain.org	5/24/1978	416 323-3232
4	James	Smith	jim@supergig.co.uk	20/10/1980	416 323-8888

Sales

CID	Date	SaleAmount
2	5/6/2004	$100.22
1	5/7/2004	$99.95
3	5/7/2004	$122.95
3	5/13/2004	$100.00
4	5/22/2004	$555.55

```
SELECT Customers.FirstName, Customers.LastName, SUM(Sales.SaleAmount) AS
SalesPerCustomer
FROM Customers, Sales
WHERE Customers.CID = Sales.CID
GROUP BY Customers.FirstName, Customers.LastName
```

SQL JOIN

Το ερώτημα αυτό επιλέγει όλους τους πελάτες (όνομα και επίθετο) και το συνολικό ποσό που έχουν καταναλώσει σε αγορές.

Το JOIN καθορίζεται μετά το WHERE και μας εκφράζει το γεγονός ότι οι δύο πίνακες έχουν ταιριασθεί βασιζόμενοι στο κοινό τους πεδίο CustomerID.

FirstName	**LastName**	**SalesPerCustomers**
John	Smith	$99.95
Steven	Goldfish	$100.22
Paula	Brown	$222.95
James	Smith	$555.55

SQL JOIN

```
SELECT Customers.FirstName, Customers.LastName,
SUM(Sales.SaleAmount) AS SalesPerCustomer
FROM Customers JOIN Sales
ON Customers.CustomerID = Sales.CustomerID
GROUP BY Customers.FirstName, Customers.LastName;
```

Υπάρχουν δύο τύποι των SQL JOINS, το INNER JOIN και το OUTER JOIN. Σε περίπτωση που δεν διευκρινίζεται τότε επιλέγεται το INNER JOIN ("INNER JOIN" = "JOIN")

Το "INNER JOIN" θα επιλέξει όλες τις εγγραφές εκείνες από τους δύο πίνακες εφόσον υπάρχει κοινό σημείο μεταξύ των εγγραφών των δύο πινάκων.

Δηλαδή θα επιστρέψει πληροφορίες για τους πελάτες που έχουν ήδη παραγγείλει κάτι.

Στην περίπτωση που έχουμε πελάτη στον πίνακα Customers και δεν έχει παραγγείλει κάτι ακόμη (δεν υπάρχει δηλαδή εγγραφή στον πίνακα Sales με αυτό το CustomerID) τότε δεν θα εμφανισθεί στα αποτελέσματα του ερωτήματος.

SQL JOIN

- Αν θέλουμε να εμφανίσουμε όλους τους πελάτες μας και τις αγορές τους, ανεξαρτήτως του γεγονότος αν έχουν αγοράσει κάτι ή όχι, τότε χρησιμοποιούμε το **OUTER JOIN**.
- Ο δεύτερος τύπος του JOIN καλείται **OUTER JOIN** και έχει δύο υποκατηγορίες που ονομάζονται **LEFT OUTER JOIN** και **RIGHT OUTER JOIN**.
- Το **LEFT OUTER JOIN** ή απλά **LEFT JOIN** επιλέγει όλες τις εγγραφές από το πρώτο πίνακα που εμφανίζεται στο FROM δίχως να υπολογίσει αν έχει κάποιο ταίρι στο δεύτερο πίνακα.

SQL JOIN

Παράδειγμα LEFT JOIN

- SELECT Customers.FirstName, Customers.LastName,
SUM(Sales.SaleAmount) AS SalesPerCustomer
FROM Customers LEFT JOIN Sales
ON Customers.CustomerID = Sales.CustomerID
GROUP BY Customers.FirstName, Customers.LastName;

- Το **RIGHT OUTER JOIN** ή απλά **RIGHT JOIN** συμπεριφέρεται ακριβώς όπως το **LEFT JOIN**, το **RIGHT JOIN** επιστρέφει όλες τις εγγραφές από τον δεύτερο πίνακα (το δεξιό πίνακα στο JOIN τμήμα του ερωτήματος)
- SELECT Customers.FirstName, Customers.LastName,
SUM(Sales.SaleAmount) AS SalesPerCustomer
FROM Customers RIGHT JOIN Sales
ON Customers.CustomerID = Sales.CustomerID
GROUP BY Customers.FirstName, Customers.LastName;

ΤΕΛΟΣ ΓΙΑ ΣΗΜΕΡΑ

ΚΟΥΡΑΣΤΗΚΑΜΕ????