ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΑΤΡΑΣ

Τμήμα Διοίκησης Επιχειρήσεων

«Βάσεις Δεδομένων και Ευφυή Πληροφοριακά Συστήματα»

«Σημειώσεις για την SQL»

ΕΞΑΜΗΝΟ: ΣΤ

Δρ. Κωνσταντίνος Χ. Γιωτόπουλος

Πάτρα, Νοέμβριος 2010

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. Προεπιλεγμένη τιμή σημαίνει ότι σε κάθε νέα εγγραφή του πίνακα η τιμή στο πεδίο αυτό θα είναι συγκεκριμένη και φυσικά μπορούμε αν θέλουμε να την αλλάξουμε. Για την προσθήκη μιας προεπιλεγμένης τιμής σε ένα πεδίο πρέπει να προσθέσουμε την έκφραση default [τιμή] μετά τη δήλωση του τύπου δεδομένων του πεδίου. 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

Μπορούμε να εισάγουμε περιορισμούς όταν καθορίζουμε το τύπο δεδομένων ενός πεδίου. Αυτοί οι περιορισμοί εισάγονται είτε στην εντολή CREATE TABLE είτε χρησιμοποιώντας την εντολή ALTER TABLE αφού δημιουργήσουμε τον πίνακα.

Κοινοί τύποι περιορισμών είναι οι κάτωθι:

- **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.

Για παράδειγμα:

```
CREATE TABLE Customer
(SID integer NOT NULL,
Last_Name varchar (30) NOT NULL,
First_Name varchar(30));
```

Τα πεδία "SID" και "Last_Name" δεν μπορούν να είναι NULL, ενώ το "First_Name" μπορεί να μην πάρει τιμή σε μια εγγραφή.

UNIQUE

Ο περιορισμός UNIQUE διασφαλίζει ότι όλες οι τιμές σε ένα πεδίο είναι διακριτές δηλαδή μοναδικές.

Παράδειγμα:

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

Το πεδίο "SID" δε δέχεται διπλότυπες τιμές. Αυτό σημαίνει ότι κάθε εγγραφή του πίνακα Customer πρέπει να έχει διαφορετική τιμή στο πεδίο SID σε σχέση με όλες τις υπόλοιπες τιμές των άλλων εγγραφών στο συγκεκριμένο πεδίο. Το ίδιο χαρακτηριστικό ισχύει και στο πεδίο που χαρακτηρίζεται ως πρωτεύον κλειδί. Πρέπει όμως να τονισθεί ότι ένα πεδίο Unique μπορεί να είναι, μπορεί και να μην είναι όμως πεδίο κλειδί.

CHECK

Ο περιορισμός CHECK διασφαλίζει ότι όλες οι τιμές που θα εισαχθούν σε ένα πεδίο ικανοποιούν συγκεκριμένα κριτήρια.

Για παράδειγμα:

```
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. Μπορεί να είναι κάποιο υπάρχον χαρακτηριστικό πεδίο του πίνακα, ή μπορεί να είναι ένα τεχνητό χαρακτηριστικό πεδίο (που ουσιαστικά δεν έχει καμία σχέση με τον πίνακα) και απλά το δημιουργούμε εμείς για να καθορίσουμε κάθε εγγραφή με ένα και μόνο πεδίο. 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.

Παραδείγματα

MySQL:

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

Oracle:

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

SQL Server:

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

Με το ALTER TABLE

MySQL:

```
ALTER TABLE Customer ADD PRIMARY KEY (SID);
```

Oracle:

```
ALTER TABLE Customer ADD PRIMARY KEY (SID);
```

SQL Server:

```
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.

Παράδειγμα: Ας υποθέσουμε ότι έχουμε δύο πίνακες τον Customers και τον Orders. Ο περιορισμός που υπάρχει είναι ότι όλες οι παραγγελίες, δηλαδή όλες οι εγγραφές του πίνακα orders να είναι συσχετισμένες με κάποιο πελάτη, δηλαδή κάποια εγγραφή του πίνακα customers. Έτσι πρέπει να εισαχθεί ένα ξένο κλειδί στον πίνακα ORDERS και να συσχετισθεί με το πρωτεύον κλειδί του πίνακα CUSTOMERS. Έτσι, θα διασφαλίσουμε ότι όλες οι παραγγελίες του πίνακα orders θα είναι συσχετισμένες με κάποιο υπαρκτό πελάτη. Έτσι δε θα μπορούμε να εισάγουμε παραγγελία που ο πελάτης μας δεν υπάρχει στον αντίστοιχο πίνακα.

Έτσι έχουμε τα κάτωθι:

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	

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.

ΠΑΡΑΔΕΙΓΜΑΤΑ:

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));
```

Oracle:

```
CREATE TABLE ORDERS
(Order_ID integer primary key,
Order_Date date,
Customer_SID integer references CUSTOMER(SID),
Amount double);
```

SQL Server:

```
CREATE TABLE ORDERS
(Order_ID integer primary key,
Order_Date datetime,
Customer_SID integer references CUSTOMER(SID),
Amount double);
```

Χρήση του ερωτήματος ALTER TABLE

MySQL:

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

Oracle:

```
ALTER TABLE ORDERS
ADD (CONSTRAINT fk_orders1) FOREIGN KEY (customer_sid)
REFERENCES CUSTOMER(SID);
```

SQL Server:

```
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]

Το [alter specification] εξαρτάται από την τροποποίηση που θέλουμε να κάνουμε. Για τα προαναφερθέντα έχουμε:

- 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"

Παραδείγματα:

Table ***customer***

Column Name	Data Type
First_Name	char(50)
Last_Name	char(50)
Address	char(50)
City	char(50)
Country	char(25)
Birth_Date	date

Καταρχήν θέλουμε να προσθέσουμε ένα πεδίο με όνομα "Gender" στον πίνακα:

ALTER table customer add Gender char(1)

Έτσι έχουμε την κάτωθι δομή πίνακα:

Table ***customer***

Column Name	Data Type
First_Name	char(50)
Last_Name	char(50)
Address	char(50)
City	char(50)
Country	char(25)
Birth_Date	date
Gender	char(1)

Έπειτα θέλουμε να μετονομάσουμε το πεδίο "Address" σε "Addr":

ALTER table customer change Address Addr char(50)

Έτσι έχουμε την κάτωθι δομή πίνακα:

Table ***customer***

Column Name	Data Type
First_Name	char(50)
Last_Name	char(50)
Addr	char(50)
City	char(50)
Country	char(25)
Birth_Date	date
Gender	char(1)

Έπειτα, θέλουμε να τροποποιήσουμε το τύπο δεδομένων του "Addr" σε char(30):

ALTER table customer modify Addr char(30)

Έτσι έχουμε την κάτωθι δομή πίνακα:

Table ***customer***

Column Name	Data Type
First_Name	char(50)
Last_Name	char(50)
Addr	char(30)
City	char(50)
Country	char(25)
Birth_Date	date
Gender	char(1)

Τέλος, θέλουμε να διαγράψουμε το πεδίο "Gender":

ALTER table customer drop Gender

Έτσι έχουμε την κάτωθι δομή πίνακα:

Table ***customer***

Column Name	Data Type
First_Name	char(50)
Last_Name	char(50)
Addr	char(30)
City	char(50)
Country	char(25)
Birth_Date	date

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

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
```

Το αποτέλεσμα της εκτέλεσης του ερωτήματος **SQL DISTINCT** θα είναι το ακόλουθο:

LastName
Smith
Goldfish
Brown

Παρατηρούμε ότι το επίθετο Smith εμφανίζεται μόνο μια φορά.

SQL WHERE

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

Πίνακας: **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

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'
```

Το αποτέλεσμα του ερωτήματος είναι το εξής:

FirstName	LastName	Email	DOB	Phone
John	Smith	John.Smith@yahoo.com	2/4/1968	626 222-2222
James	Smith	jim@supergig.co.uk	20/10/1980	416 323-8888

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

LastName = 'Smith'

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

Θα χρησιμοποιήσουμε τον πίνακα Customers για να δούμε την εφαρμογή του **SQL LIKE**:

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

Η εντολή **SQL LIKE** είναι πολύ χρήσιμη όταν θέλουμε να καθορίσουμε κριτήριο αναζήτησης σε ένα SQL WHERE, βασιζόμενοι σε ένα μέρος του περιεχομένου του πεδίου. Για παράδειγμα ας υποθέσουμε ότι θέλουμε να αναζητήσουμε όλους τους πελάτες που το FirstName ξεκινάει από 'J'. Θα πρέπει να δημιουργήσουμε το ακόλουθο ερώτημα SQL:

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

Το αποτέλεσμα:

FirstName	LastName	Email	DOB	Phone
John	Smith	John.Smith@yahoo.com	2/4/1968	626 222-2222
James	Smith	jim@supergig.co.uk	20/10/1980	416 323-8888

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. Τοποθετείται σε οποιοδήποτε μέρος του αλφαριθμητικού που εισάγουμε στο **SQL LIKE** και μάλιστα όσες φορές θέλουμε.

Σημείωση: Ανάλογα με το ΣΔΒΔ που χρησιμοποιούμε ο χαρακτήρας ‘%’ μπορεί να είναι διαφορετικός για παράδειγμα '%' είναι στον MS SQL Server, and '*' είναι ο αντίστοιχος χαρακτήρας που χρησιμοποιείται στην MS Access.

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. Παράδειγμα:

```
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 (%)

Το αποτέλεσμα της εκτέλεσης του SQL ερωτήματος:

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

SQL INSERT INTO

Η σύνταξη του **SQL INSERT INTO** έχει δύο φόρμες και το αποτέλεσμα και των δύο είναι η εισαγωγή μιας νέας εγγραφής στον πίνακα δεδομένων.

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')
```

Το αποτέλεσμα της εκτέλεσης του ερωτήματος και στις δύο περιπτώσεις είναι η εισαγωγή νέας εγγραφής στον 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
Peter	Hunt	peter.hunt@tgmail.net	1/1/1974	626 888-8888

Αν θέλετε να εισάγετε μερικά δεδομένα σε αντίστοιχα πεδία και όχι σε όλα, πρέπει να χρησιμοποιηθεί ο δεύτερος τρόπος σύνταξης αλλιώς θα προκληθεί λάθος.

Ας υποθέσουμε ότι θέλω να εισάγω καινούρια εγγραφή στον πίνακα Customers και να εισάγω μόνο τιμές στα πεδία FirstName και LastName, πρέπει να εκτελεσθεί η ακόλουθη εντολή:

```
INSERT INTO Customers (FirstName, LastName)
VALUES ('Peter', 'Hunt')
```

SQL UPDATE

Η γενική σύνταξη ενός ερωτήματος **SQL UPDATE** είναι η ακόλουθη:

```
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). Στο WHERE καθορίζουμε ποιες από τις εγγραφές του πίνακα πρέπει να αλλάξουν τιμή (αυτές που ικανοποιούν τα κριτήρια του WHERE).

Αν θέλουμε να αλλάξουμε την ημερομηνία γέννησης του Steven Goldfish's σε '5/10/1974' στο 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

Πρέπει να συντάξουμε και να εκτελέσουμε το κάτωθι **SQL UPDATE** ερώτημα:

```
UPDATE Customers
SET DOB = '5/10/1974'
WHERE LastName = 'Goldfish' AND FirstName = 'Steven'
```

Αν δεν καθορίσουμε κάποιο κριτήριο στο WHERE τότε η αλλαγή που επιθυμούμε θα εφαρμοσθεί σε όλες τις εγγραφές του πίνακα. Γίνεται κατανοητό, ότι πρέπει να είμαστε πολύ προσεκτικοί στη χρήση του ερωτήματος..

Πρέπει να τονίσουμε ότι η μπορούμε να αλλάξουμε τις τιμές πολλών πεδίων πολλών εγγραφών χρησιμοποιώντας κατάλληλα τα κριτήρια στο WHERE σε μια εντολή UPDATE. Για παράδειγμα ας υποθέσουμε ότι θέλουμε να αλλάξουμε το τηλέφωνο σε όλους τους πελάτες μας στον πίνακα Customers που το επίθετό τους είναι Smith (έχουμε δύο στην δική μας περίπτωση). Πρέπει να εκτελέσουμε το κάτωθι ερώτημα:

```
UPDATE Customers
SET Phone = '626 555-5555'
WHERE LastName = 'Smith'
```

Μετά την εκτέλεση, ο πίνακας Customer θα είναι ο εξής:

FirstName	LastName	Email	DOB	Phone
John	Smith	John.Smith@yahoo.com	2/4/1968	626 555-5555
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	626 555-5555

SQL DELETE

Μέχρι τώρα είδαμε πως να επιλέγουμε δεδομένα προς εμφάνιση από έναν πίνακα και πως να εισάγουμε και να τροποποιούμε δεδομένα στον πίνακα. Πρέπει να αναφερθούμε και στη διαγραφή εγγραφών από έναν πίνακα μιας βάσης δεδομένων. Έτσι, έχουμε την εντολή SQL DELETE

Η εντολή **SQL DELETE** ακολουθεί την εξής σύνταξη:

```
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.. Το ακόλουθο ερώτημα θα διαγράψει όλα τα δεδομένα από τον πίνακα Table 1 και θα καταλήξουμε να έχουμε έναν άδειο πίνακα.

```
DELETE FROM Table1
```

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

```
DELETE FROM Customers
WHERE LastName = 'Smith'
```

Το προηγούμενο ερώτημα θα διαγράψει από τον πίνακα Customers όλες τις εγγραφές εκείνες που στο πεδίο LastName έχουν τιμή ‘SMITH’ και ο πίνακας Customers θα είναι πλέον ο κάτωθι:

FirstName	LastName	Email	DOB	Phone
Steven	Goldfish	goldfish@fishhere.net	4/4/1974	323 455-4545
Paula	Brown	pb@herowndomain.org	5/24/1978	416 323-3232

SQL ORDER BY

Η έκφραση ORDER BY χρησιμοποιείται όταν θέλουμε να ταξινομήσουμε τα αποτελέσματα βάση κάποιου πεδίου (ή κάποιων πεδίων). Ας υποθέσουμε ότι θέλουμε να επιλέξουμε όλες τις εγγραφές από τον πίνακα Customers και να τους ταξινομήσουμε βάση την ημερομηνία γέννησής τους. Τότε, θα πρέπει ναεκτελέσουμε το κάτωθι ερώτημα:

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

Το αποτέλεσμα της εκτέλεσης του ερωτήματος θα είναι το ακόλουθο:

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

Όπως βλέπουμε η ταξινόμηση γίνεται βάση του πεδίο DOB και μάλιστα η ταξινόμηση είναι αύξουσα (δηλαδή από την μικρότερη ημερομηνία γέννησης προς την μεγαλύτερη). Στη περίπτωση που θέλουμε φθίνουσα ταξινόμηση πρέπει να το δηλώσουμε προσθέτοντας στο τέλος του τμήματος του ORDER BY τη λέξη DESC. Παράδειγμα:

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

Το αποτέλεσμα θα είναι το κάτωθι:

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

Επισημαίνεται ότι αν δεν καθορίσουμε τρόπο ταξινόμησης, τότε από το σύστημα επιλέγεται η αύξουσα ταξινόμηση. Έτσι, τα κάτωθι ερωτήματα παράγουν τα ίδια ακριβώς αποτελέσματα:

```
SELECT * FROM Customers
ORDER BY DOB

SELECT * FROM Customers
ORDER BY DOB ASC
```

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.

Για παράδειγμα θέλουμε να επιλέξουμε όλους τους πελάτες μας που το FirstName είναι "John" και το LastName είναι "Smith", τότε θα εκτελέσουμε το κάτωθι ερώτημα:

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

Το αποτέλεσμα είναι το κάτωθι:

FirstName	LastName	Email	DOB	Phone
John	Smith	John.Smith@yahoo.com	2/4/1968	626 222-2222

Επιστρέφεται η εγγραφή από τον πίνακα Customers που ικανοποιεί και τα δύο κριτήρια που εισαχθήκανε στο WHERE τμήμα του SELECT ερωτήματος.

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.

Ας υποθέσουμε ότι θέλουμε να επιλέξουμε τους πελάτες εκείνους που έχουν FirstName 'James' ή FirstName 'Paula', πρέπει να εκτελέσουμε το ακόλουθο SQL ερώτημα:

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

Το αποτέλεσμα θα είναι το εξής:

FirstName	LastName	Email	DOB	Phone
Paula	Brown	pb@herowndomain.org	5/24/1978	416 323-3232
James	Smith	jim@supergig.co.uk	20/10/1980	416 323-8888

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.

Παράδειγμα είναι το εξής ερώτημα, να επιλέξετε τους πελάτες εκείνους που έχουν LastName 'Brown' και FirstName είτε 'James' είτε 'Paula':

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

Το αποτέλεσμα:

FirstName	LastName	Email	DOB	Phone
Paula	Brown	pb@herowndomain.org	5/24/1978	416 323-3232

SQL IN

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

Η γενική σύνταξη **SQL IN** είναι:

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

Ας χρησιμοποιήσουμε τον 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

Ας υποθέσουμε ότι έχουμε το κάτωθι SQL ερώτημα με το **SQL IN** τμήμα της:

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

Η εκτέλεση αυτού του ερωτήματος θα επιστρέψει τις εγγραφές εκείνες που η τιμή του πεδίου Date έχει τιμή είτε '5/6/2004' είτε '5/7/2004', το αποτέλεσμα είναι:

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

Μπορούμε να χρησιμοποιήσουμε το **SQL IN** και σε άλλο πεδίο στον πίνακα EmployeeHours:

```
SELECT *
FROM EmployeeHours
WHERE Hours IN (9, 10)
```

Το αποτέλεσμα της εκτέλεσης του ερωτήματος είναι:

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

SQL BETWEEN

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

Το **SQL BETWEEN** συντάσσεται ως εξής:

```
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.

Θα χρησιμοποιήσουμε τον γνωστό πίνακα Customers για να δείξουμε πως λειτουργεί το **SQL BETWEEN**:

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

Έχουμε το κάτωθι ερώτημα:

```
SELECT *
FROM Customers
WHERE DOB BETWEEN '1/1/1975' AND '1/1/2004'
```

Η εκτέλεση του ερωτήματος θα επιστρέψει τις εγγραφές εκείνες που οι τιμές των εγγραφών στο πεδίο DOB βρίσκονται μεταξύ των ημερομηνιών '1/1/1975' και '1/1/2004'. Το αποτέλεσμα είναι το κάτωθι:

FirstName	LastName	Email	DOB	Phone
Paula	Brown	pb@herowndomain.org	5/24/1978	416 323-3232
James	Smith	jim@supergig.co.uk	20/10/1980	416 323-8888

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
```

Στο προηγούμενο ερώτημα δημιουργήσαμε ένα ψευδώνυμο SQL alias το SumHoursPerEmployee και το αποτέλεσμα της εκτέλεσης του ερωτήματος θα είναι το εξής:

Employee	SumHoursPerEmployee
John Smith	25
Allan Babel	24
Tina Crown	27

Στο επόμενο παράδειγμα βλέπουμε πως χρησιμοποιείτε το ψευδώνυμο σε πίνακες:

```
SELECT Emp.Employee
FROM EmployeeHours AS Emp
```

Το αποτέλεσμα είναι το κάτωθι:

Employee
John Smith
Allan Babel
Tina Crown

Τα ψευδώνυμα πινάκων είναι πολύ χρήσιμα όταν έχουμε να επιλέξουμε δεδομένα από πολλούς πίνακες.

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.

Η σύνταξη της SQL COUNT:

```
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
```

Το αποτέλεσμα της εκτέλεσης του ερωτήματος είναι το εξής:

NumberOfCustomers
4

SQL MAX

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

Η σύνταξή της είναι πολύ απλή και είναι η εξής:

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

Αν χρησιμοποιήσουμε τον γνωστό πίνακα Customers και προσπαθήσουμε να βρούμε την μεγαλύτερη ημερομηνία γέννησης τότε θα πρέπει να εκτελέσουμε το κάτωθι ερώτημα:

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

SQL MIN

Η **SQL MIN** συνάρτηση μας επιτρέπει την επιλογή της ελάχιστης τιμής ενός πεδίου.

Η σύνταξη της είναι απλή:

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

Αν χρησιμοποιήσουμε τον γνωστό πίνακα Customers και προσπαθήσουμε να βρούμε την μικρότερη ημερομηνία γέννησης τότε θα πρέπει να εκτελέσουμε το κάτωθι ερώτημα:

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

SQL AVG

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

Η σύνταξή της είναι:

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

Αν υποθέσουμε ότι θέλουμε να βρούμε το μέσο όρο από το πεδίο SaleAmount ενός πίνακα Sales

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

θα χρησιμοποιήσουμε το κάτωθι ερώτημα:

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

Θα επιστρέψει το εξής:

AvgSaleAmount
$195.73

SQL SUM

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

Η σύνταξή της είναι:

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

Χρησιμοποιούμε τον κάτωθι πίνακα:

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

Εκτελούμε το ακόλουθο ερώτημα:

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

Το αποτέλεσμα είναι:

SaleAmount
$978.67

Φυσικά μπορούμε να εισάγουμε συγκεκριμένα κριτήρια στο WHERE τμήμα του SELECT και να μην επιλέξουμε όλες τις εγγραφές του πίνακα. Παράδειγμα CustomerID = 3, τότε έχουμε το εξής:

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

Το αποτέλεσμα θα είναι:

SaleAmount
$222.95

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).

Ο καλύτερος τρόπος για να κατανοήσουμε τη χρήση του GROUP BY είναι με κάποιο παράδειγμα.

Ας υποθέσουμε ότι έχουμε τον κάτωθι πίνακα EmployeeHours στον οποίο αποθηκεύουμε τις ώρες εργασίας των εργαζομένων ανά ημέρα σε μια επιχείρηση:

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

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

```
SELECT SUM (Hours)
FROM EmployeeHours
```

Αλλά αν θέλουμε να δούμε 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
```

Το αποτέλεσμα της εκτέλεσης του ερωτήματος θα είναι το εξής:

Employee	Hours
John Smith	25
Allan Babel	24
Tina Crown	27

Όπως μπορούμε να δούμε έχουμε μια γραμμή για κάθε εργαζόμενο, που μας δείχνει το συνολικό αριθμό ωρών εργασίας του.

Το **SQL GROUP BY** χρησιμοποιείται και με άλλες συναρτήσεις όπως η SQL AVG:

```
SELECT Employee, AVG(Hours)
FROM EmployeeHours
GROUP BY Employee
```

Το αποτέλεσμα της εκτέλεσης του ερωτήματος θα είναι το εξής:

Employee	Hours
John Smith	8.33
Allan Babel	8
Tina Crown	9

Στον συγκεκριμένο πίνακα έχει νόημα να εφαρμόσουμε το GROUP BY και στο πεδίο date για να δούμε το συνολικό αριθμό ωρών εργασίας ανά ημέρα.

```
SELECT Date, SUM(Hours)
FROM EmployeeHours
GROUP BY Date
```

Το αποτέλεσμα:

Date	Hours
5/6/2004	24
5/7/2004	27
5/8/2004	25

SQL HAVING

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

Δεν μπορούμε να εισάγουμε κριτήρια σε ένα WHERE πάνω σε ένα πεδίο που βρίσκεται στο SELECT και στο οποίο εφαρμόζεται πάνω του μια συνάρτηση. Για παράδειγμα η εκτέλεση του κάτωθι ερωτήματος θα προκαλέσει σφάλμα εκτέλεσης:

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

Σε αυτή την περίπτωση χρησιμοποιούμε το **SQL HAVING** με σκοπό ακριβώς το να εισάγουμε κριτήριο ή κριτήρια στο αποτέλεσμα μιας συνάρτησης που εφαρμόζεται σε ένα πεδίο.

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

Το προηγούμενο ερώτημα θα επιλέξει και εμφανίσει όλους τους υπαλλήλους και το άθροισμα των ωρών εργασίας τους εφόσον το άθροισμα αυτό είναι μεγαλύτερο από 24.

Employee	Hours
John Smith	25
Tina Crown	27

SQL JOIN

Το **SQL JOIN** χρησιμοποιείται όταν θέλουμε να επιλέξουμε δεδομένα που προέρχονται από δύο ή περισσότερους πίνακες.

Για να μπορέσουμε να χρησιμοποιήσουμε ένα JOIN για επιλογή δεδομένων από δύο (ή περισσότερους πίνακες) θα πρέπει οι πίνακες αυτές να σχετίζονται μεταξύ τους μέσω κάποιας συσχέτισης πεδίων τους.

Για παράδειγμα έχουμε τους κάτωθι πίνακες:

Customers:

CustomerID	FirstName	LastName	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:

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

Μπορούμε εύκολα να δούμε ότι οι 2 πίνακες συσχετίζονται μέσο ενός κοινού πεδίου που ονομάζεται CustomerID και χάρη σε αυτό μπορούμε να επιλέξουμε δεδομένα που προέρχονται και από τους δύο πίνακες, ταιριάζοντας το κοινό αυτό πεδίο.

Θεωρούμε το κάτωθι ερώτημα:

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

```
GROUP BY Customers.FirstName, Customers.LastName
```

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.

Το αποτέλεσμα του ερωτήματος είναι:

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

Το προαναφερθέν ερώτημα μπορεί να γραφθεί και χρησιμοποιώντας τη λέξη JOIN ως εξής:

```
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.

Αν ο πίνακας Sales έχει τις ακόλουθες εγγραφές:

CustomerID	Date	SaleAmount
2	5/6/2004	$100.22
1	5/6/2004	$99.95

Αν χρησιμοποιήσουμε το προηγούμενο **SQL JOIN**, θα έχουμε:

```
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
```

Το αποτέλεσμα:

FirstName	LastName	SalesPerCustomers
John	Smith	$99.95
Steven	Goldfish	$100.22

Παρότι η Paula και ο James υπάρχουν ως πελάτες, δεν εμφανίζονται στα αποτελέσματα διότι δεν έχον καμία αγορά μέχρι τώρα..

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.

Ελαφρώς αλλάζουμε το ερώτημά μας:

```
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
```

Και ας υποθέσουμε ότι ο πίνακας Sales έχει τις εξής εγγραφές:

CustomerID	Date	SaleAmount
2	5/6/2004	$100.22
1	5/6/2004	$99.95

Το αποτέλεσμα θα είναι το εξής:

FirstName	LastName	SalesPerCustomers

John	Smith	$99.95
Steven	Goldfish	$100.22
Paula	Brown	NULL
James	Smith	NULL

Όπως βλέπουμε έχει επιλέξει όλες τις εγγραφές από τον Customers (πρώτο πίνακα). Για κάθε εγγραφή του πίνακα Customers που δεν έχει κάποιο ταίρι (ίδιο κωδικό) στον SALES (δεύτερο πίνακα) το SalerPerCustomers είναι NULL (NULL σημαίνει ότι η τιμή του πεδίου είναι κενή).

Το **RIGHT OUTER JOIN** ή απλά **RIGHT JOIN** συμπεριφέρεται ακριβώς όπως το **LEFT JOIN**, η διαφορά τους είναι ότι το **RIGHT JOIN** επιστρέφει όλες τις εγγραφές από τον δεύτερο πίνακα (το δεξιό πίνακα στο JOIN τμήμα του ερωτήματος).