

## Σημειώσεις στο μάθημα Βάσεις Δεδομένων

### **Hands on 05 (final)**

## Πίνακας περιεχομένων

1	<b>Γενικά</b> .....	6
1.1	<b>Εισαγωγή</b> .....	6
1.2	SQL .....	8
1.3	<b>Το</b> Human Resources (HR) Schema .....	9
2	<b>Εντολές SQL</b> .....	10
2.1	SELECT .....	10
2.1.1	Ανάκτηση δεδομένων με την χρήση του <i>SELECT</i> .....	10
2.1.2	Χρήση αριθμητικών τελεστών στο <i>SELECT</i> .....	11
2.1.3	Χρήση του <i>NULL</i> στο <i>SELECT</i> .....	12
2.1.4	Ψευδώνυμα στηλών στο <i>SELECT</i> .....	12
2.1.5	Ο χαρακτήρας συνένωσης, και ο τελεστής <i>DISTINCT</i> .....	13
2.1.6	Ασκήσεις στο <i>SELECT</i> .....	14
2.2	<b>Περιορισμός (WHERE) και ταξινόμηση εγγραφών (ORDER BY)</b> 15	
2.2.1	Σκοπός .....	15
2.2.2	Περιορισμός των εγγραφών που επιλέγονται .....	15
2.2.3	Χρήση Συμβολοσειρών και Ημερομηνιών .....	16
2.2.4	Τελεστές σύγκρισης .....	16
2.2.5	Χρήση Τελεστών Σύγκρισης .....	17
2.2.6	Χρήση του <i>BETWEEN</i> για αναζητήσεις σε εύρος τιμών .....	17
2.2.7	Χρήση του <i>IN</i> για αναζητήσεις σε σύνολο τιμών .....	18
2.2.8	Χρήση του <i>LIKE</i> για σύγκριση μοτίβων ( <i>pattern matching</i> ) .....	18
2.2.9	Έλεγχος <i>NULL</i> .....	19
2.2.10	Σύνθεση συνθηκών με την χρήση λογικών τελεστών .....	19

2.2.11	<i>Ο τελεστής AND</i> .....	20
2.2.12	<i>Ο τελεστής OR</i> .....	20
2.2.13	<i>Ο τελεστής NOT</i> .....	21
2.2.14	<i>Σειρά εκτέλεσης πράξεων τελεστών</i> .....	21
2.2.15	<i>Χρήση του ORDER BY για ταξινόμηση εγγραφών</i> .....	22
2.2.16	<i>Ασκήσεις</i> .....	23
2.3	<b>Συναθροιστικές Συναρτήσεις και Ομαδοποίηση εγγραφών</b> .....	25
2.3.1	<i>Γενικά</i> .....	25
2.3.2	<i>Σύνταξη Συναθροιστικών Συναρτήσεων</i> .....	25
2.3.3	<i>Χρήση των AVG και SUM συναρτήσεων</i> .....	26
2.3.4	<i>Χρήση των MIN και MAX συναρτήσεων</i> .....	26
2.3.5	<i>Χρήση της COUNT</i> .....	26
2.3.6	<i>Συναθροιστικές συναρτήσεις και NULL τιμές</i> .....	27
2.3.7	<i>Ομαδοποίηση δεδομένων</i> .....	28
2.3.8	<i>Ομαδοποίηση δεδομένων με χρήση της έκφρασης GROUP BY</i> .....	28
2.3.9	<i>Χρήση της έκφρασης GROUP BY με περισσότερα του ενός πεδία</i> .....	30
2.3.10	<i>Σφάλματα και περιορισμοί στην εφαρμογή ομαδοποιήσεων</i> .....	31
2.3.11	<i>Περιορισμός αποτελεσμάτων ομαδοποιήσεων με το HAVING</i> .....	32
2.3.12	<i>Παραδείγματα χρήσης του HAVING</i> .....	33
2.3.13	<i>Φωλευμένες συναθροιστικές συναρτήσεις</i> .....	33
2.3.14	<i>Ασκήσεις</i> .....	33
2.4	<b>Σύνδεση πινάκων</b> .....	35
2.4.1	<i>Γενικά</i> .....	35
2.4.2	<i>Σύνταξη των ερωτημάτων σύνδεσης πινάκων (SQL: 1999 πρότυπο)</i> .....	36
2.4.3	<i>Μετονομασία κατά την σύνδεση πινάκων</i> .....	36

2.4.4	Φυσική σύνδεση πινάκων.....	37
2.4.5	Σύνδεση πινάκων με την χρήση του <i>USING</i> .....	38
2.4.6	Σύνδεση πινάκων με την χρήση του <i>ON</i> .....	38
2.4.7	Σύνδεση πίνακα με τον εαυτό του με την χρήση του <i>ON</i> .....	40
2.4.8	Συνδέσεις μεταξύ πινάκων με την χρήση και άλλων τελεστών .....	40
2.4.9	Συνδέσεις μεταξύ πινάκων με εξωτερικές συνδέσεις .....	44
2.4.10	Αριστερή εξωτερική σύνδεση.....	45
2.4.11	Δεξιά εξωτερική σύνδεση .....	46
2.4.12	Πλήρης εξωτερική σύνδεση .....	47
2.4.13	Καρτεσιανό γινόμενο.....	47
2.4.14	Ασκήσεις .....	48
2.5	<b>Χρήση υποερωτημάτων</b> .....	50
2.5.1	Γενικά .....	50
2.5.2	Σύνταξη των υποερωτημάτων .....	51
2.5.3	Υποερωτήματα μονής εγγραφής.....	52
2.5.4	Υποερωτήματα πολλαπλών εγγραφών .....	53
2.5.5	Ο τελεστής <i>ALL</i> .....	54
2.5.6	Ο τελεστής <i>ANY</i> .....	54
2.5.7	Ασκήσεις .....	55
2.6	<b>Τελεστές Συνόλων</b> .....	57
2.6.1	Γενικά .....	57
2.6.2	Ο τελεστής <i>UNION/UNION ALL</i> .....	57
2.6.3	Ο τελεστής <i>INTERSECT</i> .....	58
2.6.4	Ο τελεστής <i>MINUS</i> .....	59
2.6.5	Ασκήσεις .....	59

<b>2.7</b>	<b>Χειρισμός Δεδομένων</b>	60
2.7.1	Γενικά	60
2.7.2	Εισαγωγή Δεδομένων - Η εντολή <i>INSERT</i>	61
2.7.3	Ενημέρωση Δεδομένων - Η εντολή <i>UPDATE</i>	62
2.7.4	Διαγραφή Δεδομένων - Η εντολή <i>DELETE</i>	64
2.7.5	<i>DEFAULT</i> τιμές	65
2.7.6	Συναλλαγές με την <i>ΒΔ</i>	66
2.7.7	Ασκήσεις	67
<b>2.8</b>	<b>Δημιουργία πινάκων</b>	68
2.8.1	Γενικά	68
2.8.2	Η εντολή <i>CREATE TABLE</i>	68
2.8.3	Περιορισμοί ακεραιότητας ( <i>integrity constraints</i> )	69
2.8.4	Περιορισμός <i>NOT NULL</i>	70
2.8.5	Περιορισμός <i>UNIQUE</i>	71
2.8.6	Περιορισμός <i>PRIMARY KEY</i> (πρωτεύοντος κλειδιού)	72
2.8.7	Περιορισμός <i>FOREIGN KEY</i> (ξένου κλειδιού)	72
2.8.8	Ασκήσεις	73

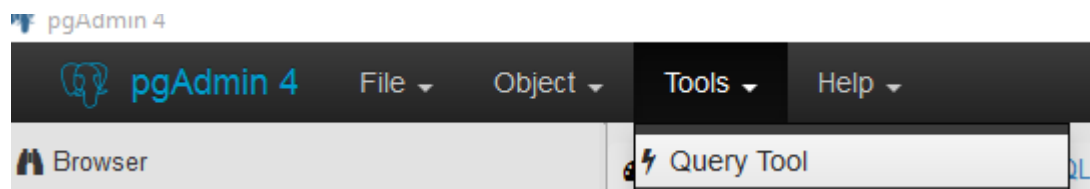
## 1 Γενικά

### 1.1 Εισαγωγή

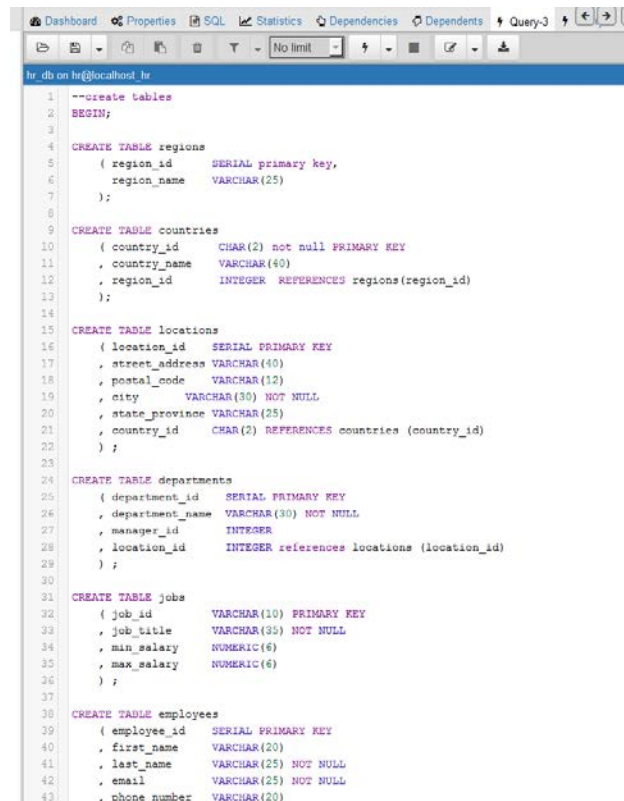
Σε αυτή την πρακτική εκπαίδευση θα μελετήσουμε πρακτικά βασικά θέματα των σχεσιακών βάσεων δεδομένων αλλά και θα δούμε διάφορες εντολές της SQL γλώσσας. Σκοπός είναι μετά την ολοκλήρωση της πρακτικής να μπορείτε να γράφετε ερωτήματα σε έναν ή περισσότερους πίνακες, να μπορείτε να τροποποιείτε δεδομένα και να δημιουργείτε αντικείμενα.

Στην περίπτωση

**α.** που έχετε επιλέξει το περιβάλλον του pgAdmin και αφού έχετε ολοκληρώσει τα προηγούμενα βήματα (postgresql\_installation.01.oct2016, pgAdmin.01.oct2016) ,συνδέεστε στον server της χρήσης **hr** επιλέγετε την **HRProdDB** και επιλέγετε *Tools>QueryTool* οπότε ανοίγει ένα μια καρτέλα για την εκτέλεση εντολών σε SQL.




Στη συνέχεια ανοίγετε το αρχείο HR\_pgsql.sql (με notepad) από το οποίο επιλέγετε ΟΛΟ το κείμενο και κάνετε αντιγραφή και επικόλληση μέσα στην καρτέλα SQL του pgAdmin που ανοίξατε προηγουμένως.



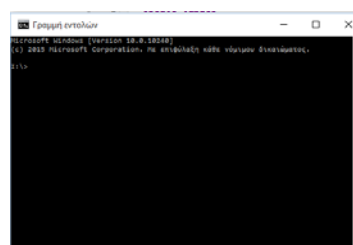
```

hr_db on hr@localhost:hr
1  --create tables
2  BEGIN;
3
4  CREATE TABLE regions
5  (
6    region_id    SERIAL primary key,
7    region_name  VARCHAR(25)
8  );
9
10 CREATE TABLE countries
11 (
12   country_id    CHAR(2) not null PRIMARY KEY
13   , country_name VARCHAR(40)
14   , region_id    INTEGER REFERENCES regions(region_id)
15 );
16
17 CREATE TABLE locations
18 (
19   location_id    SERIAL PRIMARY KEY
20   , street_address VARCHAR(40)
21   , postal_code   VARCHAR(12)
22   , city          VARCHAR(30) NOT NULL
23   , state_province VARCHAR(25)
24   , country_id    CHAR(2) REFERENCES countries (country_id)
25 );
26
27 CREATE TABLE departments
28 (
29   department_id    SERIAL PRIMARY KEY
30   , department_name VARCHAR(30) NOT NULL
31   , manager_id      INTEGER
32   , location_id      INTEGER references locations (location_id)
33 );
34
35 CREATE TABLE jobs
36 (
37   job_id          VARCHAR(10) PRIMARY KEY
38   , job_title      VARCHAR(35) NOT NULL
39   , min_salary     NUMERIC(6)
40   , max_salary     NUMERIC(6)
41 );
42
43 CREATE TABLE employees
44 (
45   employee_id    SERIAL PRIMARY KEY
46   , first_name    VARCHAR(20)
47   , last_name     VARCHAR(25) NOT NULL
48   , email         VARCHAR(25) NOT NULL
49   , phone_number  VARCHAR(20)

```

Πατάτε το εικονίδιο  και περιμένετε για να ολοκληρωθεί η εκτέλεση των εντολών. Μετά από αυτό είστε έτοιμοι να ξεκινήσετε.

**β.** που έχετε επιλέξει το περιβάλλον του SQL Developer και αφού έχετε ολοκληρώσει τα προηγούμενα βήματα (oracleexpress\_installation.01.oct2016, sqldeveloper.01.oct2016) , ανοίγετε μια γραμμή εντολών των windows στην οποία δίνετε:



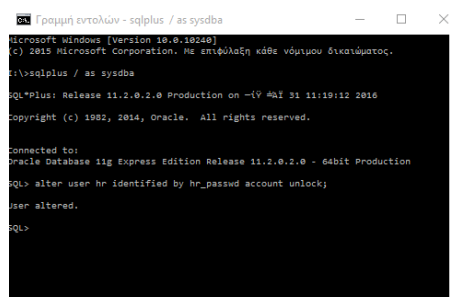
```

C:\>

```

*sqlplus / as sysdba*

*sql>alter user hr identified by hr\_passwd account unlock;*



```

C:\>sqlplus / as sysdba

SQL*Plus: Release 11.2.0.2.0 Production on -[V 01 OCT 2016
Copyright (c) 1982, 2014, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
SQL> alter user hr identified by hr_passwd account unlock;
User altered.
SQL>

```

Μετά από αυτό είστε έτοιμοι να ξεκινήσετε αφού δημιουργήσετε μια νέα σύνδεση αυτή τη φορά για τον χρήστη *hr* (ΟΧΙ για τον hr2).

## 1.2 SQL

Όπως έχουμε ήδη αναφέρει με την χρήση της sql (structure query language) σε μια σχεσιακή βάση δεδομένων, αποκρύπτονται η διαδικαστικές λεπτομέρειες που απαιτούνται για την διαχείριση των δεδομένων. Η sql αποτελεί την επιλεγμένη από την ANSI (American National Standard Institute) και το ISO (International Standards Organization) γλώσσα για σχεσιακές βάσεις δεδομένων και προσφέρει την δυνατότητα για:

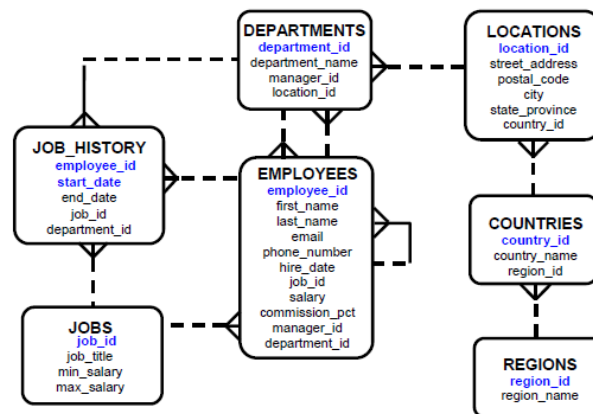
- α. εκτέλεση ερωτημάτων
- β. εισαγωγή, ενημέρωση και διαγραφή γραμμών από πίνακα
- γ. δημιουργία, αντικατάσταση, μεταβολή και διαγραφή αντικειμένων
- δ. έλεγχο πρόσβασης στην βάση δεδομένων και σε αντικείμενα
- ε. διασφάλιση της ακεραιότητας και των περιορισμών

Μπορούμε να κατηγοριοποιήσουμε τις εντολές τις SQL ανάλογα ως εξής:

Εντολή	Περιγραφή
<i>SELECT</i> <i>INSERT</i> <i>UPDATE</i> <i>DELETE</i> <i>MERGE</i>	Ανακτά, εισάγει, τροποποιεί και διαγράφει δεδομένα. Αναφέρεται ως <i>data manipulation language</i> (DML)
<i>CREATE</i> <i>ALTER</i> <i>DROP</i> <i>RENAME</i> <i>TRUNCATE</i> <i>COMMENT</i>	Ορίζει, μεταβάλλει και διαγράφει δομές δεδομένων. Αναφέρεται ως <i>data definition language</i> (DDL)
<i>GRANT</i> <i>REVOKE</i>	Παραχωρεί ή αφαιρεί δικαιώματα πρόσβασης στην βάση δεδομένων και σε δομές της. Αναφέρεται ως <i>data control language</i> (DCL)
<i>COMMIT</i> <i>ROLLBACK</i> <i>SAVEPOINT</i>	Δίνει την δυνατότητα ελέγχου των αλλαγών που γίνονται από τις DML εντολές. Οι αλλαγές μπορούν να ομαδοποιηθούν σε λογικές συναλλαγές (logical transactions). Αναφέρεται ως <i>transaction control language</i> (TCL)



### 1.3 **To** Human Resources (HR) Schema

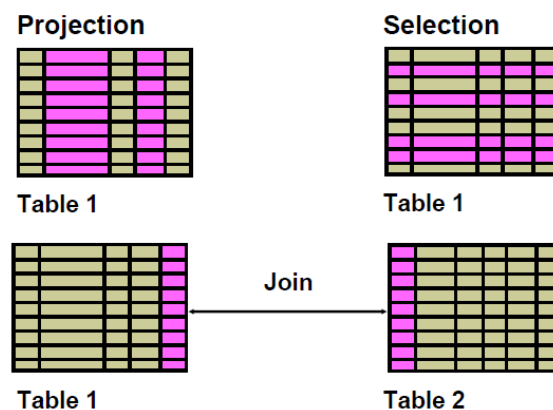


## 2 Εντολές SQL

### 2.1 SELECT

#### 2.1.1 Ανάκτηση δεδομένων με την χρήση του *SELECT*

Για την ανάκτηση δεδομένων από την βάση δεδομένων χρησιμοποιείτε την εντολή *SELECT* της sql η οποία σας δίνει την δυνατότητα να περιορίσετε και τα πεδία τα οποία εμφανίζονται. Πιο συγκεκριμένα οι δυνατότητες που σας δίνονται με την χρήση του *SELECT* είναι



**Προβολή (projection):** με την οποία επιλέγετε να προβάλλονται μια ή περισσότερες στήλες (πεδία) της επιλογής σας.

**Επιλογή (selection):** με την οποία επιλέγετε τις γραμμές του πίνακα που επιστρέφονται με το ερώτημα σας. Εδώ μπορούν να εφαρμοστούν διάφορα κριτήρια για τον περιορισμό των γραμμών.

**Σύνδεση (joining):** με την οποία ανακτούνται δεδομένα που βρίσκονται σε διαφορετικούς συσχετισμένους πίνακες. Μια βασική σύνταξη του *SELECT* δείχνεται στην παρακάτω εικόνα.

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM    table;
```

Μπορούμε να επιλέξουμε **όλες τις στήλες** από έναν πίνακα με την χρήση του συμβόλου *\**. Ένας άλλος τρόπος είναι να γράψουμε τα ονόματα όλων των στηλών του πίνακα αμέσως μετά το *SELECT*.

*SELECT department\_id, department\_name, manager\_id, location\_id FROM departments;*

```
SELECT *  
FROM departments;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Shipping	124	1500
4	40	IT	103	1400
5	50	Sales	149	2500
6	60	Executive	100	1700
7	70	Accounting	205	1700
8	80	Contracting	(null)	1700

Μπορούμε να επιλέξουμε **ορισμένες στήλες** από έναν πίνακα με την χρήση του συμβόλου γράφοντας τα ονόματα όλων των στηλών χωρισμένα με κόμμα του πίνακα που ενδιαφερόμαστε αμέσως μετά το *SELECT*.

*SELECT location\_id, department\_id FROM departments;*

Στη σύνταξη εντολών SQL, όπως είναι το *SELECT*, ισχύει ότι οι εντολές δεν είναι case sensitive, ότι μπορούμε να τις γράψουμε σε μια ή περισσότερες γραμμές, ότι οι λέξεις κλειδιά δεν μπορούν να διαχωριστούν μεταξύ γραμμών και ότι τερματίζονται με την χρήση του αγγλικού ερωτηματικού (;).

### 2.1.2 Χρήση αριθμητικών τελεστών στο *SELECT*

Κατά την σύνταξη του *SELECT* μπορούμε να πραγματοποιήσουμε και αριθμητικές πράξεις σε αριθμητικά δεδομένα ή σε ημερομηνίες με την χρήση τελεστών όπως + (πρόσθεση), - (αφαίρεση), \* (πολλαπλασιασμός), / (διαίρεση). Η χρήση των τελεστών μπορεί να γίνει σε οποιοδήποτε σημείο εκτός από το *FROM*.

Παράδειγμα χρήσης αριθμητικών τελεστών είναι το παρακάτω όπου στις τιμές του πεδίου *salary* προσθέτουμε 300 (έστω ότι θέλουμε να δώσουμε αύξηση 300 ευρώ σε ΟΛΟΥΣ!!!).

```
SELECT last_name, salary, salary + 300  
FROM employees;
```

	LAST_NAME	SALARY	SALARY+300
1	King	24000	24300
2	Kochhar	17000	17300
3	De Haan	17000	17300
4	Hunold	9000	9300
5	Ernst	6000	6300
6	Lorentz	4200	4500
7	Mourgos	5800	6100
8	Rajs	3500	3800
9	Davies	3100	3400
10	Matos	2600	2900

...

Στην περίπτωση όπου μια αριθμητική παράσταση περιέχει περισσότερους του ενός τελεστές τότε ο πολλαπλασιασμός και η διαίρεση εκτελούνται πρώτοι. Εάν οι τελεστές έχουν την ίδια προτεραιότητα τότε η εκτέλεση γίνεται από αριστερά προς τα δεξιά. Τέλος με την χρήση παρενθέσεων μπορούμε να καθορίσουμε ποιες αριθμητικές παραστάσεις θα εκτελεστούν πρώτες. Για παράδειγμα το αποτέλεσμα των δύο παρακάτω εντολών sql διαφέρει (γιατί;).

```
SELECT last_name, salary, 12*(salary+100) FROM employees;
```

```
SELECT last_name, salary, 12*salary+100 FROM employees;
```

### 2.1.3 Χρήση του NULL στο SELECT

Το *NULL* δηλώνει ότι τιμή μιας στήλης σε μια γραμμή δεν μας είναι διαθέσιμη ή δεν την γνωρίζουμε ή δεν μπορεί να εφαρμοστεί. Διαφέρει από το τιμή μηδέν ή το κενό διάστημα.

```
SELECT last_name, job_id, salary, commission_pct
FROM employees;
```

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)
...				
12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2
...				
19	Higgins	AC_MGR	12000	(null)
20	Gietz	AC_ACCOUNT	8300	(null)

Η χρήση του null σε μια αριθμητική έκφραση μας επιστρέφει πάλι null π.χ.

```
SELECT last_name, 12*salary*commission_pct
FROM employees;
```

	LAST_NAME	12*SALARY*COMMISSION_PCT
1	King	(null)
2	Kochhar	(null)
...		
12	Zlotkey	25200
13	Abel	39600
14	Taylor	20640
...		
19	Higgins	(null)
20	Gietz	(null)

### 2.1.4 Ψευδώνυμα στηλών στο SELECT

Με τη χρήση των ψευδώνυμων σε στήλες μπορούμε να μετονομάσουμε μια στήλη (κατά την εκτέλεση του *SELECT*). Η λειτουργία αυτή είναι χρήσιμη κατά την εκτέλεση υπολογισμών. Πραγματοποιείτε είτε γράφοντας το ψευδώνυμο αμέσως μετά το όνομα της στήλης που θέλουμε να μετονομάσουμε ή γράφοντας την δεσμευμένη λέξη *AS* και μετά το ψευδώνυμο. Στην περίπτωση που αυτό περιέχει κενά ή ειδικούς χαρακτήρες είναι χρήση διπλών εισαγωγικών (π.χ. "SALARY INCR") οπότε και είναι case sensitive.

```
SELECT last_name AS name, commission_pct comm
FROM employees;
```

	NAME	COMM
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)

...

```
SELECT last_name "Name", salary*12 "Annual Salary"
FROM employees;
```

	Name	Annual Salary
1	King	288000
2	Kochhar	204000
3	De Haan	204000

...

### 2.1.5 Ο χαρακτήρας συνένωσης, και ο τελεστής *DISTINCT*

Ο χαρακτήρας συνένωσης δηλώνετε με το σύμβολο // ( δύο κάθετες μπάρες) και χρησιμοποιείται για να συνενώσει τιμές στηλών ή χαρακτήρες σε άλλες στήλες (κατά την εκτέλεση του *SELECT*). Στην περίπτωση που γίνει συνένωση στήλης με *NULL* τότε το αποτέλεσμα είναι η τιμή της στήλης.

```
SELECT last_name || ' is a ' || job_id
AS "Employee Details"
FROM employees;
```

	Employee Details
1	Abel is a SA_REP
2	Davies is a ST_CLERK
3	De Haan is a AD_VP
4	Ernst is a IT_PROG
5	Fay is a MK_REP

...

18	Vargas is a ST_CLERK
19	Whalen is a AD_ASST
20	Zlotkey is a SA_MAN

Κατά την εκτέλεση του ερωτήματος η προκαθορισμένη λειτουργία είναι η εμφάνιση όλων των εγγραφών συμπεριλαμβανομένων και των διπλότυπων. Η χρήση του τελεστή *DISTINCT* μας δίνει την δυνατότητα να απαλείψουμε τις διπλότυπες εγγραφές. Τοποθετούμε τον τελεστή αυτόν αμέσως μετά την δεσμευμένη λέξη *SELECT*, πχ. *SELECT DISTINCT ...*

```
SELECT department_id  
FROM employees;
```

1

	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60

...

```
SELECT DISTINCT department_id  
FROM employees;
```

2

	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110

Στην περίπτωση που μετά το *DISTINCT* επιλέξουμε περισσότερες από μια στήλες τότε ο τελεστής λειτουργεί στο συνδυασμό των τιμών αυτών των στηλών.

```
SELECT DISTINCT department_id, job_id
FROM employees;
```

	DEPARTMENT_ID	JOB_ID
1	110	AC_ACCOUNT
2	90	AD_VP
3	50	ST_CLERK
4	80	SA_REP
5	50	ST_MAN

### 2.1.6 Ασκήσεις στο *SELECT*

1. Τα ακόλουθα *SELECT* εκτελούνται χωρίς σφάλματα;

```
SELECT last_name, job_id, salary AS Sal FROM employees; (ΣΩΣΤΟ/ΛΑΘΟΣ)
```

```
SELECT * FROM job_grades; (ΣΩΣΤΟ/ΛΑΘΟΣ)
```

2. Η παρακάτω εντολή περιέχει τέσσερα σφάλματα μπορείτε να τα εντοπίσετε;

```
SELECT employee_id, last_name sal x 12 ANNUAL SALARY FROM employees;
```

3. Μέσα από το pgAdmin ή τον SQL Developer δείτε την δομή και τα δεδομένα των πινάκων *DEPARTMENTS*, *EMPLOYEES*. Το τμήμα HR θέλει τα επώνυμα, τον κωδικό θέσης, την ημερομηνία πρόσληψης και τον αριθμό του κάθε εργαζομένου με το πεδίο (στήλη *HIRE\_DATE* να εμφανίζεται με το ψευδώνυμο *STARTDATE*). Σώστε το ερώτημα αυτό με το όνομα lab\_01\_05.sql
4. Το τμήμα HR θέλει μοναδικά τους κωδικούς θέσεων από τον πίνακα *EMPLOYEES*.

5. Το τμήμα HR θέλει καλύτερες περιγραφές στο ερώτημα lab\_01\_05.sql για αυτό το λόγο δώστε κατάλληλα ονόματα στις στήλες *Emp#*, *Employee*, *Job* και *Hire Date* αντίστοιχα.
6. Το τμήμα HR ζητά επίσης μια αναφορά όπου θα φαίνονται όλοι οι υπάλληλοι και οι κωδικοί θέσεων. Εμφανίστε την αναφορά αυτή χρησιμοποιώντας το επίθετο σε συνένωση με τον κωδικό θέσης και ονομάστε τη στήλη *Employee and Title*
7. Εμφανίστε όλα τα δεδομένα του πίνακα *EMPLOYEES* χωρίζοντας τις τιμές όλων των πεδίων με κόμμα και ονομάστε την στήλη αυτή *THE\_OUTPUT*.

## 2.2 Περιορισμός (WHERE) και ταξινόμηση εγγραφών (ORDER BY)

### 2.2.1 Σκοπός

Σκοπός της ενότητας αυτής είναι να μάθουμε πως μπορούμε να περιορίσουμε και να ταξινομήσουμε τις εγγραφές ενός ερωτήματος. Πιο συγκεκριμένα για τον περιορισμό των αποτελεσμάτων θα δούμε την χρήση του *WHERE* σε συνδυασμό με διάφορους τελεστές σύγκρισης όπως είναι τα *=*, *<=*, *BETWEEN*, *IN*, *LIKE*, αλλά και τελεστών για *NULL* και για λογικούς ελέγχους με *AND*, *OR* και *NOT*.

### 2.2.2 Περιορισμός των εγγραφών που επιλέγονται

Για να περιορίσουμε τις εγγραφές που επιλέγονται χρησιμοποιούμε το *WHERE* με το οποίο ορίζεται η συνθήκη που πρέπει να επαληθεύουν οι εγγραφές για να εμφανίζονται στο αποτέλεσμα. Ακολουθεί πάντα μετά από το *FROM* στην σύνταξη του ερωτήματος.

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM table
[WHERE condition(s)];
```

Το *WHERE* μπορούμε να το χρησιμοποιήσουμε για να συγκρίνουμε τιμές στηλών (για τα ονόματα των οποίων ΔΕΝ μπορούμε να χρησιμοποιήσουμε ψευδώνυμα), αριθμητικές εκφράσεις ή συναρτήσεις και αποτελείται από τρία μέρη: το όνομα της στήλης που θέλουμε να συγκρίνουμε, τον τελεστή σύγκρισης και το όνομα της στήλης, σταθερά ή σύνολο τιμών με τον οποίο γίνεται η σύγκριση.

Για παράδειγμα παρακάτω επιλέγουμε το *employee ID*, *last name*, *job ID* και το *department number* όλων των υπαλλήλων που ανήκουν στο department 90.

```
SELECT employee_id, last_name, job_id, department_id
FROM employees
WHERE department_id = 90;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

### 2.2.3 Χρήση Συμβολοσειρών και Ημερομηνιών

Οι συμβολοσειρές και οι ημερομηνίες στο *WHERE* πρέπει να εμπεριέχονται σε μονά εισαγωγικά (' '). Η αναζήτηση στις συμβολοσειρές είναι *case sensitive* πράγμα που σημαίνει ότι παίζει ρόλο η χρήση πεζών ή κεφαλαίων. Το παρακάτω ερώτημα δεν επιστρέφει αποτελέσματα καθώς οι τιμές του *last\_name* συντάσσονται με συνδυασμό κεφαλαίων και μικρών π.χ. 'Whalen'

```
SELECT last_name, department_id FROM employees WHERE last_name = 'WHALEN';
```

Στις ημερομηνίες οι αναζητήσεις πρέπει επίσης να γίνονται με την χρήση μονών εισαγωγικών π.χ.

```
SELECT last_name FROM employees WHERE hire_date = '17-FEB-96';
```

### 2.2.4 Τελεστές σύγκρισης

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN ...AND...	Between two values (inclusive)
IN (set)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

Οι τελεστές σύγκρισης χρησιμοποιούνται στις συνθήκες για την σύγκριση μια έκφρασης με μια τιμή ή μια άλλη έκφραση. Η χρήση τους γίνεται στο *WHERE*

```
... WHERE expr operator value
```

π.χ.

```
... WHERE hire_date = '17-FEB-96';
```

```
... WHERE salary > 6000;
```

```
... WHERE hire_date < '17-FEB-96';
```



### 2.2.5 Χρήση Τελεστών Σύγκρισης

Στο παρακάτω παράδειγμα επιλέγουμε το *last\_name* και το *salary* από τον πίνακα *EMPLOYEES* για εκείνους του υπαλλήλους των οποίων ο μισθός είναι μικρότερος ή ίσος με 3000. Παρατηρήστε ότι ορίζεται μια τιμή στο τμήμα του *WHERE* η οποία συγκρίνεται με τις τιμές που περιέχει η στήλη *SALARY* του πίνακα.

```
SELECT last_name, salary
FROM employees
WHERE salary <= 3000 ;
```

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

### 2.2.6 Χρήση του BETWEEN για αναζητήσεις σε εύρος τιμών

Με την χρήση του *BETWEEN* μπορεί να γίνει αναζήτηση σε εύρος τιμών για το οποίο καθορίζουμε πρώτο το κάτω και δεύτερο το πάνω όριο. Μπορεί να γίνει ορισμός του εύρους σε αριθμητικές τιμές, σε κείμενο και σε ημερομηνίες π.χ.

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500 ;
```

Lower limit      Upper limit

	LAST_NAME	SALARY
1	Rajs	3500
2	Davies	3100
3	Matos	2600
4	Vargas	2500

```
SELECT last_name
FROM employees
WHERE last_name BETWEEN 'King' AND 'Smith';
```

	LAST_NAME
1	King
2	Kochhar
3	Lorentz
4	Matos
5	Mourgos
6	Rajs

### 2.2.7 Χρήση του *IN* για αναζητήσεις σε σύνολο τιμών

```
SELECT employee_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IN (100, 101, 201) ;
```

	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	101	Kochhar	17000	100
2	102	De Haan	17000	100
3	124	Mourgos	5800	100
4	149	Zlotkey	10500	100
5	201	Hartstein	13000	100
6	200	Whalen	4400	101
7	205	Higgins	12000	101
8	202	Fay	6000	201

Με την χρήση του *IN* μπορεί να γίνει έλεγχος ύπαρξης μιας τιμής σε σύνολο τιμών. Μπορεί να χρησιμοποιηθεί για οποιοδήποτε τύπο δεδομένων και στην περίπτωση των συμβολοσειρών και των ημερομηνιών πρέπει αυτά να περικλείονται σε μονά εισαγωγικά π.χ.

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE last_name IN ('Hartstein', 'Vargas');
```

### 2.2.8 Χρήση του *LIKE* για σύγκριση μοτίβων (pattern matching)

```
SELECT first_name
FROM employees
WHERE first_name LIKE 'S%';
```

Με την χρήση του *LIKE* μπορεί να γίνει σύγκριση με χαρακτήρες μπαλαντέρ (wildcard) σε τιμές οι οποίες μπορεί να είναι κείμενο/αριθμοί ή να συνδυασμός τους με την χρήση του

% το οποίο δηλώνει ύπαρξη μηδέν ή περισσότερων χαρακτήρων

\_ το οποίο δηλώνει ενός χαρακτήρα (ακριβώς)

Χρήση του *LIKE* γίνεται σε περιπτώσεις που δεν είναι γνωστή επακριβώς η τιμή την οποία ψάχνουμε, για παράδειγμα στην παραπάνω εικόνα γίνεται αναζήτηση για εκείνους του υπαλλήλους που το όνομα τους ξεκινά με το γράμμα S και μπορεί να περιέχει, στην συνέχεια, οποιοσδήποτε χαρακτήρες και οποιοδήποτε αριθμό χαρακτήρων (χρήση του %).

Υπάρχει η δυνατότητα να γίνει χρήση συνδυασμού των δύο χαρακτήρων μπαλαντέρ (% \_ ) κατά την αναζήτηση π.χ.

```
SELECT last_name
FROM employees
WHERE last_name LIKE '_o%' ;
```

LAST_NAME
1 Kochhar
2 Lorentz
3 Mourgos

### 2.2.9 Έλεγχος NULL

```
SELECT last_name, manager_id
FROM employees
WHERE manager_id IS NULL ;
```

LAST_NAME	MANAGER_ID
1 King	(null)

Επειδή η τιμή *null* σημαίνει την απουσία τιμής δεν μπορεί να γίνει η χρήση του τελεστή της ισότητας = και για αυτό τον λόγο χρειαζόμαστε ειδικό τελεστή. Έτσι για να ελέγξουμε την (μη)ύπαρξη του *null* χρησιμοποιούμε τους τελεστές *IS NOT NULL* και *IS NULL*.

Στο πιο πάνω παράδειγμα αναζητούνται οι υπάλληλοι οι οποίοι δεν έχουν μάνατζερ.

### 2.2.10 Σύνθεση συνθηκών με την χρήση λογικών τελεστών

Operator	Meaning
AND	Returns TRUE if <i>both</i> component conditions are true
OR	Returns TRUE if <i>either</i> component condition is true
NOT	Returns TRUE if the condition is false

Με την χρήση των λογικών τελεστών μπορούμε να συνδυάσουμε επιμέρους συνθήκες ώστε το τελικό αποτέλεσμα να προκύπτει από την επαλήθευση τους ή την επαλήθευση της αντίστροφής τους. Σε κάθε περίπτωση μια εγγραφή εμφανίζεται στο αποτέλεσμα μόνο εάν η συνολική συνθήκη είναι αληθής, επιστρέφει TRUE. Ο λογικοί τελεστές οι οποίοι είναι διαθέσιμοι στην SQL είναι οι: *AND*, *OR* και *NOT* και στον παραπάνω πίνακα δείχνεται πότε επιστρέφεται αποτέλεσμα ανά τελεστή.

### 2.2.11 Ο τελεστής AND

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >= 10000
AND job_id LIKE '%MAN%';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	149	Zlotkey	SA_MAN	10500
2	201	Hartstein	MK_MAN	13000

Ο τελεστής *AND* απαιτεί όλες οι συνθήκες που συμμετέχουν να είναι αληθείς, δηλαδή να επιστρέφουν TRUE. Για να γίνει κατανοητό αυτό στο παράδειγμα που δείχνεται επιστρέφονται μόνο εκείνοι οι υπάλληλοι που έχουν μισθό μεγαλύτερο από 10.000 και των οποίων ο τίτλος εργασίας περιέχει την συμβολοσειρά 'MAN'. Στον παρακάτω πίνακα φαίνονται τα αποτελέσματα του συνδυασμού δύο συνθηκών με *AND*.

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

### 2.2.12 Ο τελεστής OR

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >= 10000
OR job_id LIKE '%MAN%';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	100	King	AD_PRES	24000
2	101	Kochhar	AD_VP	17000
3	102	De Haan	AD_VP	17000
4	124	Mourgos	ST_MAN	5800
5	149	Zlotkey	SA_MAN	10500
6	174	Abel	SA_REP	11000
7	201	Hartstein	MK_MAN	13000
8	205	Higgins	AC_MGR	12000

Ο τελεστής *OR* απαιτεί κάποια από τις συνθήκες που συμμετέχουν να είναι αληθείς, δηλαδή να επιστρέφει TRUE. Για να γίνει κατανοητό αυτό στο παράδειγμα που δείχνεται επιστρέφονται εκείνοι οι υπάλληλοι που έχουν μισθό μεγαλύτερο από 10.000 ή των οποίων ο τίτλος εργασίας (*job\_id*) περιέχει την συμβολοσειρά 'MAN'. Στον παρακάτω πίνακα φαίνονται τα αποτελέσματα του συνδυασμού δύο συνθηκών με *OR*.

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

### 2.2.13 Ο τελεστής NOT

```
SELECT last_name, job_id
FROM employees
WHERE job_id
      NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP') ;
```

	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

Στο πιο πάνω ερώτημα επιστρέφονται εκείνοι οι υπάλληλοι των οποίων ο τίτλος εργασίας (*job\_id*) δεν περιέχει την συμβολοσειρά 'IT\_PROG', 'ST\_CLERK' και 'SA\_REP'. Στον παρακάτω πίνακα φαίνονται τα αποτελέσματα όταν εφαρμόζεται το *NOT*.

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Το *NOT* μπορεί επίσης να χρησιμοποιηθεί με το *BETWEEN*, *LIKE* και *NULL*.

### 2.2.14 Σειρά εκτέλεσης πράξεων τελεστών

Operator	Meaning
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical condition
8	AND logical condition
9	OR logical condition

Στον πίνακα φαίνεται η σειρά εκτέλεσης των πράξεων των τελεστών στην περίπτωση που εμφανίζονται στην ίδια έκφραση. Η σειρά αυτή μπορεί να ελεγχθεί με την χρήση παρενθέσεων. Για παράδειγμα το ερώτημα 1 μεταφράζεται ως εξής "Επέλεξε μια εγγραφή στην περίπτωση που ο υπάλληλος έχει τίτλο εργασίας 'AD\_PRES' και έχει μισθό μεγαλύτερο από 15.000, ή έχει τίτλο εργασίας 'SA\_REP' ". Το ερώτημα 2 μεταφράζεται "Επέλεξε μια

εγγραφή στην περίπτωση που ένας υπάλληλος έχει τίτλο εργασίας 'AD\_PRES' ή 'SA\_REP', και έχει μισθό μεγαλύτερο από 15.000'.

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = 'SA_REP'
OR job_id = 'AD_PRES'
AND salary > 15000;
```

1

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000
2	Abel	SA_REP	11000
3	Taylor	SA_REP	8600
4	Grant	SA_REP	7000

```
SELECT last_name, job_id, salary
FROM employees
WHERE (job_id = 'SA_REP'
OR job_id = 'AD_PRES')
AND salary > 15000;
```

2

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000

### 2.2.15 Χρήση του ORDER BY για ταξινόμηση εγγραφών

#### Syntax

```
SELECT          expr
FROM            table
[WHERE          condition(s)]
[ORDER BY {column, expr, numeric_position} [ASC|DESC]];
```

#### In the syntax:

ORDER BY	specifies the order in which the retrieved rows are displayed
ASC	orders the rows in ascending order (this is the default order)
DESC	orders the rows in descending order

Με την χρήση του *ORDER BY* μπορούν να ταξινομηθούν οι εγγραφές που επιστρέφονται σε ένα ερώτημα είτε με αύξουσα σειρά (*ASC*), που είναι το προκαθορισμένο, είτε με φθίνουσα (*DESC*). Το *ORDER BY* τοποθετείται πάντα τελευταίο στην σύνταξη του *SELECT* και εκτός από το όνομα της στήλης μπορεί να γίνει χρήση έκφρασης, ψευδώνυμου ή αριθμητική θέση της στήλης στο *SELECT* για να οριστεί η σειρά ταξινόμησης. Τέλος με την χρήση των *NULLS FIRST* και *NULLS LAST* μπορούμε να ορίσουμε αν στην περίπτωση που υπάρχουν *NULL* αυτά θα εμφανίζονται πρώτα ή τελευταία.

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date DESC;
```

1

```
SELECT employee_id, last_name, salary*12 annsal
FROM employees
ORDER BY annsal;
```

2

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY 3;
```

3

```
SELECT last_name, department_id, salary  
FROM employees  
ORDER BY department_id, salary DESC;
```

4

### 2.2.16 Ασκήσεις

1. Λόγω περικοπών, το τμήμα HR θέλει το επώνυμο και τον μισθό των εργαζομένων που κερδίζουν περισσότερα από 12.000. Σώστε το ερώτημα αυτό με το όνομα lab\_02\_01.sql.

	AZ	LAST_NAME	AZ	SALARY
1		King		24000
2		Kochhar		17000
3		De Haan		17000
4		Hartstein		13000

2. Σε ένα ερώτημα εμφανίστε το επώνυμο και τον αριθμό τμήματος του υπαλλήλου με αριθμό 176.

	AZ	LAST_NAME	AZ	DEPARTMENT_ID
1		Taylor		80

3. Το τμήμα HR θέλει να βρει τους υπαλλήλους με τον μεγαλύτερο και τον μικρότερο μισθό. Τροποποιήστε το ερώτημα lab\_02\_01.sql έτσι ώστε να προβάλετε το όνομα και τον μισθό όλων των υπαλλήλων των οποίων ο μισθός ΔΕΝ βρίσκεται μεταξύ 5.000 και 12.000. Αποθηκεύστε το ερώτημα ως lab\_02\_03.sql.

	AZ	LAST_NAME	AZ	SALARY
1		King		24000
2		Kochhar		17000
3		De Haan		17000
4		Lorentz		4200
5		Rajs		3500
6		Davies		3100
7		Matos		2600
8		Vargas		2500
9		Whalen		4400
10		Hartstein		13000

4. Δημιουργήστε ένα ερώτημα στο οποίο θα προβάλετε το επώνυμο, τον αριθμό και την ημερομηνία έναρξης όλων των υπαλλήλων με επώνυμο Matos και Taylor. Ταξινομήστε τα αποτελέσματα με αύξουσα σειρά κατά την ημερομηνία έναρξης.

	A Z LAST_NAME	A Z JOB_ID	HIRE_DATE
1	Matos	ST_CLERK	15-MAR-98
2	Taylor	SA_REP	24-MAR-98

5. Εμφανίστε το επώνυμο και το αριθμό τμήματος όλων των υπαλλήλων των τμημάτων 20 ή 50 σε αύξουσα σειρά κατά επώνυμο.

	A Z LAST_NAME	A Z DEPARTMENT_ID
1	Davies	50
2	Fay	20
3	Hartstein	20
4	Matos	50
5	Mourgos	50
6	Rajs	50
7	Vargas	50

6. Τροποποιήστε το ερώτημα lab\_02\_03.sql έτσι ώστε στο αποτέλεσμα σας να εμφανίζεται το επώνυμο και ο μισθός των υπαλλήλων των οποίων οι αποδοχές είναι μεταξύ 5.000 και 12.000 και οι οποίοι βρίσκονται στο τμήμα 20 ή 50. Ονομάστε τα πεδία *Employee* και *Monthly Salary* αντίστοιχα και σώστε το νέο ερώτημα με όνομα lab\_02\_06.sql.

	A Z Employee	A Z Monthly Salary
1	Fay	6000
2	Mourgos	5800

7. Το τμήμα HR θέλει το επώνυμο και την ημερομηνία πρόσληψης όλων των υπαλλήλων που προσλήφθηκαν το 2004.

	A Z LAST_NAME	HIRE_DATE
1	Higgins	07-JUN-94
2	Gietz	07-JUN-94

8. Εμφανίστε το επώνυμο και τον τίτλο εργασίας εκείνων των υπαλλήλων που δεν έχουν μάνατζερ.

	A Z LAST_NAME	A Z JOB_ID
1	King	AD_PRES

9. Εμφανίστε τα επώνυμα των υπαλλήλων που έχουν ταυτόχρονα "a" και "e" στο επώνυμο τους.

	A Z LAST_NAME
1	Davies
2	De Haan
3	Hartstein
4	Whalen



## 2.3 Συναθροιστικές Συναρτήσεις και Ομαδοποίηση εγγραφών

### 2.3.1 Γενικά

Οι συναθροιστικές συναρτήσεις είναι συναρτήσεις που παίρνουν ως είσοδο ένα σύνολο από τιμές και επιστρέφουν μια μόνο τιμή.

The diagram illustrates the use of the MAX aggregate function. On the left, the 'EMPLOYEES' table is shown with columns DEPARTMENT\_ID and SALARY. The SALARY column is highlighted with a red box. An orange arrow points from this column to a query result box on the right. The query result box shows the result of the query 'MAX(SALARY)' as 24000.

	DEPARTMENT_ID	SALARY
1	90	24000
2	90	17000
3	90	17000
4	60	9000
5	60	6000
6	60	4200
7	50	5800
8	50	3500
9	50	3100
10	50	2600
...		
18	20	6000
19	110	12000
20	110	8300

Maximum salary in EMPLOYEES table

MAX(SALARY)
24000

Η SQL προσφέρει πέντε ενσωματωμένες συναθροιστικές συναρτήσεις

- α. Μέσος όρος: **avg**
- β. Ελάχιστο: **min**
- γ. Μέγιστο: **max**
- δ. Άθροισμα: **sum**
- ε. Καταμέτρηση: **count**

Η είσοδος στο **sum** και στο **avg** πρέπει να είναι μια συλλογή από αριθμούς, αλλά οι άλλοι τελεστές μπορούν να λειτουργήσουν σε συλλογές μη αριθμητικών τύπων δεδομένων, π.χ. σε συμβολοσειρές.

### 2.3.2 Σύνταξη Συναθροιστικών Συναρτήσεων

```
SELECT group_function(column), ...
FROM   table
[WHERE condition]
[ORDER BY column];
```

Οι συναθροιστικές συναρτήσεις τοποθετούνται αμέσως μετά το λεκτικό **SELECT** και μπορούν να χρησιμοποιούνται περισσότερες από μια χωρισμένες με κόμμα. Σημαντικό είναι να θυμόμαστε ότι ΟΛΕΣ οι συναθροιστικές συναρτήσεις αγνοούν τις **NULL** τιμές.

### 2.3.3 Χρήση των AVG και SUM συναρτήσεων

```
SELECT AVG(salary), MAX(salary),  
       MIN(salary), SUM(salary)  
FROM   employees  
WHERE  job_id LIKE '%REP%';
```

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

Οι συναθροιστικές συναρτήσεις τοποθετούνται αμέσως μετά το λεκτικό *SELECT* και μπορούν να χρησιμοποιούνται περισσότερες από μια χωρισμένες με κόμμα. Σημαντικό είναι να θυμόμαστε ότι ΟΛΕΣ οι συναθροιστικές συναρτήσεις αγνοούν τις *NULL* τιμές.

### 2.3.4 Χρήση των MIN και MAX συναρτήσεων

```
SELECT MIN(hire_date), MAX(hire_date)  
FROM   employees;
```

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	17-JUN-87	29-JAN-00

Οι συναρτήσεις *MIN* και *MAX* μπορούν να χρησιμοποιηθούν με αριθμητικά, αλφαριθμητικά και με ημερομηνίες. Στην παραπάνω εικόνα εμφανίζεται η μικρότερη και μεγαλύτερη ημερομηνία πρόσληψης, ενώ πιο κάτω εμφανίζονται το πρώτο και τελευταίο όνομα των υπαλλήλων με αλφαβητική σειρά

```
SELECT MIN(last_name), MAX(last_name)  
FROM   employees;
```

	MIN(LAST_NAME)	MAX(LAST_NAME)
1	Abel	Zlotkey

### 2.3.5 Χρήση της COUNT

Οι συνάρτηση *COUNT* έχει τρεις μορφές σύνταξης

```
COUNT (*)  
COUNT (expr)  
COUNT (DISTINCT expr)
```

και για κάθε μια ισχύουν τα παρακάτω. Η *COUNT(\*)* επιστρέφει τον αριθμό των γραμμών ενός πίνακα οι οποίες επαληθεύουν το *WHERE* σε περίπτωση που υπάρχει συμπεριλαμβανομένων και τον διπλότυπων εγγραφών καθώς και των εγγραφών που περιέχουν *NULL* τιμές σε κάποιο από τα πεδία τους. Σε αντίθεση το *COUNT(expr)* επιστρέφει τον αριθμό των όχι-NULL τιμών των πεδίων που καθορίζονται στο *expr*.

Το *COUNT(DISTINCT expr)* επιστρέφει τον αριθμό των γραμμών των μοναδικών όχι-NULL τιμών των πεδίων που καθορίζονται στο *expr*.

Παράδειγμα το παρακάτω ερώτημα επιστρέφει τον αριθμό των υπαλλήλων του τμήματος 50.

```
SELECT COUNT(*)  
FROM employees  
WHERE department_id = 50;
```

	COUNT(*)
1	5

ενώ στο ακόλουθο ερώτημα επιστρέφεται ο αριθμός των όχι-NULL τιμών του πεδίου *commission\_pct* του τμήματος 80.

```
SELECT COUNT(commission_pct)  
FROM employees  
WHERE department_id = 80;
```

	COUNT(COMMISSION_PCT)
1	3

και στο πιο κάτω επιστρέφεται ο αριθμός των μοναδικών όχι-NULL τιμών του πεδίου *department\_id*.

```
SELECT COUNT(DISTINCT department_id)  
FROM employees;
```

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

### 2.3.6 Συναθροιστικές συναρτήσεις και NULL τιμές

Οι συναθροιστικές συναρτήσεις αγνοούν τις *NULL* τιμές που βρίσκονται στο πεδίο υπολογισμού, για παράδειγμα η μέση τιμή στο πιο κάτω ερώτημα υπολογίζεται με βάση τις τιμές εκείνων των εγγραφών οι οποίες στο πεδίο *commission\_pct* έχουν έγκυρη τιμή (όχι-*NULL* αριθμό).

```
SELECT AVG(commission_pct)  
FROM employees;
```

	AVG(COMMISSION_PCT)
1	0.2125

Για να μπορεί να γίνει χρήση και των εγγραφών οι οποίες περιέχουν *NULL* υπάρχει στην Oracle η συνάρτηση *NVL* με την οποία μπορούμε να χειριστούμε τα *NULLS*.

```
SELECT AVG(NVL(commission_pct, 0))  
FROM employees;
```

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

Στην περίπτωση αυτή όσες εγγραφές έχουν *NULL* στην τιμή της στήλης πεδίο *commission\_pct* υπολογίζονται σαν να έχουν 0.

Αντίστοιχες συναρτήσεις χειρισμού των *NULLS* υπάρχουν και σε άλλα συστήματα. Για παράδειγμα στον SQL Server υπάρχει η *ISNULL*, στην MySQL υπάρχει η *IFNULL*, στην Postgres υπάρχει η *COALESCE* κ.ο.κ.

### 2.3.7 Ομαδοποίηση δεδομένων

EMPLOYEES

1	DEPARTMENT_ID	2	SALARY	
1	10		4400	4400
2	20		13000	
3	20		6000	9500
4	50		5800	
5	50		2500	
6	50		2600	3500
7	50		3100	
8	50		3500	
9	60		4200	6400
10	60		6000	
11	60		9000	
12	80		11000	
13	80		10500	10033
14	80		8600	
***				
19	110		12000	
20	(null)		7000	

Average salary in  
EMPLOYEES table for  
each department

DEPARTMENT_ID	AVG(SALARY)	
1	10	4400
2	20	9500
3	50	3500
4	60	6400
5	80	10033.333333333333
6	90	19333.333333333333
7	110	10150
8	(null)	7000

Μέχρι στιγμής στα παραδείγματα που αναφερθήκαμε οι συναθροιστικές συναρτήσεις αναφέρθηκαν τον πίνακα σαν ένα μεγάλο σύνολο για την ομαδοποίηση των δεδομένων. Η συνθήκη αυτή δεν επαρκεί καθώς υπάρχουν περιπτώσεις όπου θέλουμε να συναθροίσουμε πολλές διαφορετικές ομάδες δεδομένων. Η απαίτηση αυτή μπορεί να καλυφθεί με την χρήση της έκφρασης *GROUP BY* που θα μελετήσουμε παρακάτω.

### 2.3.8 Ομαδοποίηση δεδομένων με χρήση της έκφρασης *GROUP BY*

```
SELECT    column, group_function(column)
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

Μπορούμε να κάνουμε χρήση της έκφρασης *GROUP BY* για να ομαδοποιήσουμε τις εγγραφές ενός πίνακα σε ομάδες και με την χρήση συναθροιστικών συναρτήσεων να εξάγουμε συγκεντρωτική πληροφορία για τις ομάδες αυτές π.χ.

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY  department_id;
```

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	90	19333.33333333333...
3	20	9500
4	110	10150
5	50	3500
6	80	10033.33333333333...
7	60	6400
8	10	4400

Στο πιο πάνω παράδειγμα υπολογίζεται ο μέσος μισθός (εφαρμογή συναθροιστικής συνάρτησης) ανά τμήμα (ομαδοποίηση). Κατά την χρήση του *GROUP BY* η σειρά εκτέλεσης έχει ως εξής:

- Η έκφραση *SELECT* ορίζει τα πεδία που πρέπει να ανακτηθούν και είναι:
  - Ο αριθμός του τμήματος στον πίνακα *EMPLOYEES*
  - Ο μέσος όρος των μισθών των ομάδων έτσι όπως καθορίζονται στην έκφραση *GROUP BY*
- Η έκφραση *FROM* καθορίζει τους πίνακες τους οποίους θα χρησιμοποιήσει το ερώτημα: *EMPLOYEES*
- Η έκφραση *WHERE* καθορίζει τους τις εγγραφές που θα ανακτηθούν, όμως στο παράδειγμα μας επειδή απουσιάζει θα ανακτηθούν όλες οι εγγραφές.
- Η έκφραση *GROUP BY* καθορίζει τον τρόπο με τον οποίο πρέπει να ανακτηθούν οι εγγραφές. Πιο συγκεκριμένα τα αποτελέσματα ομαδοποιούνται κατά αριθμό τμήματος (*department\_id*) έτσι ώστε να εφαρμοστεί η συνάρτηση *AVG* στο πεδίο του μισθού (*salary*) για τον υπολογισμό του μέσου μισθού.

Να σημειωθεί ότι η χρήση της *GROUP BY* σε κάποιο πεδίο δεν απαιτεί το πεδίο αυτό να βρίσκεται στην έκφραση του *SELECT*. Για παράδειγμα όπως φαίνεται στο παρακάτω ερώτημα επιστρέφεται ο μέσος μισθός ανά τμήμα χωρίς όμως να φαίνεται ο αριθμός τμήματος. Στην πράξη βέβαια το αποτέλεσμα αυτό δεν είναι καθόλου πρακτικό.

```
SELECT    AVG(salary)
FROM      employees
GROUP BY department_id ;
```

	AVG(SALARY)
1	7000
2	19333.333333333333333333333333...
3	9500
4	10150
5	3500
6	10033.333333333333333333333333...
7	6400
8	4400

### 2.3.9 Χρήση της έκφρασης *GROUP BY* με περισσότερα του ενός πεδία

EMPLOYEES			Add the salaries in the EMPLOYEES table for each job, grouped by department.		
DEPARTMENT_ID	JOB_ID	SALARY	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10 AD_ASST	4400	1	10 AD_ASST	4400
2	20 MK_MAN	13000	2	20 MK_MAN	13000
3	20 MK_REP	6000	3	20 MK_REP	6000
4	50 ST_MAN	5800	4	50 ST_CLERK	11700
5	50 ST_CLERK	2500	5	50 ST_MAN	5800
6	50 ST_CLERK	2600	6	60 IT_PROG	19200
7	50 ST_CLERK	3100	7	80 SA_MAN	10500
8	50 ST_CLERK	3500	8	80 SA_REP	19600
9	60 IT_PROG	4200	9	90 AD_PRES	24000
10	60 IT_PROG	6000	10	90 AD_VP	34000
11	60 IT_PROG	9000	11	110 AC_ACCOUNT	8300
12	80 SA_REP	11000	12	110 AC_MGR	12000
13	80 SA_MAN	10500	13	(null) SA_REP	7000
14	80 SA_REP	8600			
...					
19	110 AC_MGR	12000			
20	(null) SA_REP	7000			

Υπάρχουν περιπτώσεις όπου απαιτείται να δούμε συναθροίσεις για ομαδοποιήσεις μέσα σε ομαδοποιήσεις. Στην παραπάνω εικόνα εμφανίζεται ο συνολικός μισθός κάθε εργασίας ανά τμήμα. Ο πίνακας *EMPLOYEE* ομαδοποιείται αρχικά ανά τμήμα και στην συνέχεια μέσα στην ομαδοποίηση αυτή γίνεται ομαδοποίηση ανά εργασία. Για παράδειγμα στο τμήμα 50 ομαδοποιούνται οι υπάλληλοι με εργασία *ST\_CLERK* και υπολογίζεται για αυτούς ο συνολικός μισθός.

Η σύνταξη του *GROUP BY* με περισσότερα του ενός πεδία για την δημιουργία ομάδων και υποομάδων στα δεδομένα δείχνεται παρακάτω. Γίνεται με την τοποθέτηση των ονομάτων των στηλών που μας ενδιαφέρουν διαχωρισμένα με κόμμα αμέσως μετά το *GROUP BY*. Πρόσθετα μπορούμε να καθορίσουμε την σειρά ταξινόμησης των αποτελεσμάτων χρησιμοποιώντας κάποια στήλη του *GROUP BY* στην έκφραση του *ORDER BY*.

SELECT department_id dept_id, job_id, SUM(salary)		
FROM employees		
GROUP BY department_id, job_id		
ORDER BY department_id;		

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10 AD_ASST	4400
2	20 MK_MAN	13000
3	20 MK_REP	6000
4	50 ST_CLERK	11700
5	50 ST_MAN	5800
6	60 IT_PROG	19200
7	80 SA_MAN	10500
8	80 SA_REP	19600
9	90 AD_PRES	24000
10	90 AD_VP	34000
11	110 AC_ACCOUNT	8300
12	110 AC_MGR	12000
13	(null) SA_REP	7000

Στο *SELECT* του παραδείγματος αρχικά οι εγγραφές ομαδοποιούνται ανά τμήμα (*department\_id*) και στην συνέχεια ανά εργασία (*job\_id*) σε κάθε τμήμα. Με αυτό τον τρόπο η συνάρτηση *SUM* εφαρμόζεται στο πεδίο *salary* για όλες τις εργασίες σε κάθε τμήμα.

### 2.3.10 Σφάλματα και περιορισμοί στην εφαρμογή ομαδοποιήσεων

Κατά την σύνταξη ερωτημάτων με συναθροιστικές συναρτήσεις και μεμονωμένες στήλες στο *SELECT*, είναι απαραίτητη η ύπαρξη του *GROUP BY* που θα περιλαμβάνει τις μεμονωμένες στήλες. Στην περίπτωση που λείπει το *GROUP BY* συνήθως εμφανίζεται μήνυμα σφάλματος “*not a single-group function*”. Π.χ.

```
SELECT department_id, COUNT(last_name)
FROM employees;
```

ORA-00937: not a single-group group function  
00937. 00000 - "not a single-group group function"

**A GROUP BY clause must be added to count the last names for each department\_id.**

Σφάλμα επίσης έχουμε στην περίπτωση που εμφανίζονται ονόματα στηλών στο τμήμα του *SELECT* μαζί με τις συναθροιστικές συναρτήσεις και αυτές οι στήλες δεν εμφανίζονται στο τμήμα *GROUP BY* του ερωτήματος. Π.χ.

```
SELECT department_id, job_id, COUNT(last_name)
FROM employees
GROUP BY department_id;
```

ORA-00979: not a GROUP BY expression  
00979. 00000 - "not a GROUP BY expression"

**Either add job\_id in the GROUP BY or remove the job\_id column from the SELECT list.**

Η χρήση των συναθροιστικών συναρτήσεων ΔΕΝ μπορεί να γίνει στο τμήμα του *WHERE* ώστε να περιορίσουμε ομάδες εγγραφών. Π.χ.

```
SELECT department_id, AVG(salary)
FROM employees
WHERE AVG(salary) > 8000
GROUP BY department_id;
```

ORA 00934: group function is not allowed here

An error was encountered performing the requested operation:  
ORA-00934: group function is not allowed here  
00934. 00000 - "group function is not allowed here"  
Cause:  
Action:  
Error at Line:3 Column:9

**Cannot use the WHERE clause to restrict groups**

Για τον σκοπό αυτό υπάρχει η έκφραση *HAVING*. Για να εφαρμόσουμε λοιπόν επιτυχημένα το ερώτημα της παραπάνω εικόνας, με το οποίο προσπαθούμε να περιορίσουμε τα αποτελέσματα σε εκείνα μόνο τα τμήματα με μέσο μισθό μεγαλύτερο των 8.000, η σωστή σύνταξη έχει ως εξής

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 8000;
```

	DEPARTMENT_ID	AVG(SALARY)
1	90	19333.333333333333...
2	20	9500
3	110	10150
4	80	10033.333333333333...

### 2.3.11 Περιορισμός αποτελεσμάτων ομαδοποιήσεων με το HAVING

EMPLOYEES		
	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	5800
5	50	2500
6	50	2600
7	50	3100
8	50	3500
9	60	4200
10	60	6000
11	60	9000
12	80	11000
13	80	10500
14	80	8600
...		
18	110	8300
19	110	10000
20	(null)	7000

The maximum salary per department when it is greater than \$10,000

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	80	11000
3	90	24000
4	110	12000

Με την χρήση του *HAVING* μπορούμε να περιορίσουμε τα αποτελέσματα των ομαδοποιήσεων όπως ακριβώς και με την χρήση του *WHERE* στις εγγραφές. Έτσι για να βρούμε τον μεγαλύτερο μισθό ανά τμήμα για αυτά τα τμήματα που έχουν μέγιστο μισθό μεγαλύτερο από 10.000 πρέπει να γίνουν τα ακόλουθα:

1. Να βρούμε τον μέγιστο μισθό για κάθε τμήμα (ομαδοποίηση)
2. Να περιορίσουμε τις ομάδες μόνο για εκείνα τα τμήματα με μέγιστο μισθό μεγαλύτερο των 10.000

Παρατηρήστε ότι σύμφωνα με τον πρόχειρο αυτό αλγόριθμο οι ομαδοποιήσεις και οι υπολογισμοί των συναθροιστικών συναρτήσεων που τυχόν συμμετέχουν εφαρμόζονται πρώτα στα δεδομένα και στην συνέχεια τα αποτελέσματα περιορίζονται με το *HAVING*. Η σύνταξη του *HAVING* έχει ως εξής

```
SELECT    column, group_function
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[HAVING   group_condition]
[ORDER BY column];
```

με την έκφραση *group\_condition* να εμφανίζει μόνο εκείνες τις ομάδες των αποτελεσμάτων για τις οποίες είναι αληθής.



### 2.3.12 Παραδείγματα χρήσης του HAVING

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING max(salary)>10000;
```

	DEPARTMENT_ID	AVG(SALARY)
1	90	19333.333333333333...
2	20	9500
3	110	10150
4	80	10033.333333333333...

Με το πιο πάνω ερώτημα επιστρέφονται ΜΟΝΟ εκείνα τα τμήματα και ο μέσος μισθός των υπαλλήλων τους για τα οποία ο μέγιστος μισθός είναι μεγαλύτερος από 10.000. Πιο κάτω εμφανίζεται η εργασία καθώς και το σύνολο των μισθών για όσες εργασίες έχουν μισθολόγιο πάνω από 13.000. Εδώ εξαιρούνται οι *sales representatives* (%REP%) ενώ η ταξινόμηση των αποτελεσμάτων γίνεται με βάση το σύνολο των μισθών.

```
SELECT job_id, SUM(salary) PAYROLL
FROM employees
WHERE job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING SUM(salary) > 13000
ORDER BY SUM(salary);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

### 2.3.13 Φωλευμένες συναθροιστικές συναρτήσεις

```
SELECT MAX(AVG(salary))
FROM employees
GROUP BY department_id;
```

	MAX(AVG(SALARY))
1	19333.333333333333...

Όπως δείχνεται στην εικόνα μπορεί να γίνει εμφώλευση συναθροιστικών συναρτήσεων (μέχρι 2). Στο παράδειγμα υπολογίζεται ο μέσος μισθός κάθε τμήματος και στην συνέχεια εμφανίζεται ο μέγιστος μέσος μισθός.

### 2.3.14 Ασκήσεις

1. Οι συναθροιστικές συναρτήσεις παράγουν ένα αποτέλεσμα ανά ομάδα (True/False).

- Οι συναθροιστικές συναρτήσεις λαμβάνουν υπόψη τα *NULLS* κατά τον υπολογισμό του αποτελέσματος (True/False).
- Το τμήμα HR χρειάζεται μια αναφορά στην οποία θα εμφανίζεται ο μεγαλύτερος, ο μικρότερος, το σύνολο και ο μέσος μισθός όλων των υπαλλήλων. Ονομάστε τα αντίστοιχα πεδία με *Maximum*, *Minimum*, *Sum* και *Average*. Σώστε το ερώτημα σας σαν lab\_05\_04.sql.

MAXIMUM	MINIMUM	SUM	AVERAGE
24000	2100	691416	6461,83178

- Τροποποιήστε το ερώτημα lab\_05\_04.sql έτσι ώστε να επιστρέφει το μεγαλύτερο, το μικρότερο, το σύνολο, το μέσο μισθό και την εργασία όλων των υπαλλήλων ανά εργασία. Σώστε το νέο ερώτημα με όνομα lab\_05\_05.sql

MAXIMUM	MINIMUM	SUM	AVERAGE	JOB_ID
9000	4200	28800	5760	IT_PROG
12008	12008	12008	12008	AC_MGR
8300	8300	8300	8300	AC_ACCOUNT
8200	5800	36400	7280	ST_MAN
11000	11000	11000	11000	PU_MAN
4400	4400	4400	4400	AD_ASST
17000	17000	34000	17000	AD_VP
4200	2500	64300	3215	SH_CLERK
9000	6900	39600	7920	FI_ACCOUNT
12008	12008	12008	12008	FI_MGR
3100	2500	13900	2780	PU_CLERK
14000	10500	61000	12200	SA_MAN
13000	13000	13000	13000	MK_MAN
10000	10000	10000	10000	PR_REP
24000	24000	24000	24000	AD_PRES
11500	6100	250500	8350	SA_REP
6000	6000	6000	6000	MK_REP
3600	2100	55700	2785	ST_CLERK
6500	6500	6500	6500	HR_REP

- Βρείτε τον αριθμό των υπαλλήλων με την ίδια εργασία.

JOB_ID	COUNT(*)
AC_ACCOUNT	1
AC_MGR	1
AD_ASST	1
AD_PRES	1
AD_VP	2
FI_ACCOUNT	5
FI_MGR	1
HR_REP	1
IT_PROG	5
MK_MAN	1
MK_REP	1
PR_REP	1
PU_CLERK	5
PU_MAN	1
SA_MAN	5
SA_REP	30
SH_CLERK	20
ST_CLERK	20
ST_MAN	5

19 rows selected.

- Βρείτε τον αριθμό των managers οι οποίοι δεν έχουν manager και ονομάστε το πεδίο που θα παραχθεί *Number of Managers*. **Σημείωση:** Χρησιμοποιήστε την στήλη *manager\_id* για να βρείτε τους managers.

Number of Managers
1

7. Βρείτε την διαφορά μεταξύ του υψηλότερου και χαμηλότερου μισθού. Ονομάστε το πεδίο *DIFFERENCE*.

```
DIFFERENCE
-----
21900
```

8. Δημιουργήστε ένα ερώτημα με το οποίο θα φαίνονται ο αριθμός του μάνατζερ και ο μισθός του χαμηλότερα αμειβόμενου υπαλλήλου αυτού του μάνατζερ. Εξαιρέστε τους υπαλλήλους που δεν έχουν μάνατζερ καθώς και τις περιπτώσεις όπου ο χαμηλότερος μισθός είναι μικρότερος από 6.000. Τέλος ταξινομήστε το αποτέλεσμα ανά μισθό.

```
MANAGER_ID MIN(SALARY)
-----
148        6100
147        6200
149        6200
108        6900
146        7000
145        7000
205        8300
102        9000

8 rows selected.
```

9. **[Extra Challenge]** Εμφανίστε τον αριθμό των υπαλλήλων και το έτος για τους υπαλλήλους εκείνους οι οποίοι προσελήφθησαν το 2005, 2006, 2007 και 2008. Ονομάστε τα πεδία *number\_of\_employees* και *the\_year* αντίστοιχα. **Σημείωση:** Μπορείτε (στην Oracle) να απομονώσετε το έτος από μια ημερομηνία ως εξής *select extract(year from hire\_date) the\_year from employees*.

```
SQL> select extract(year from hire_date) the_year from employees;

THE_YEAR
-----
2003
2005
2001
2006
2007
2005
2006
2007
2002
2002
2005

NUMBER_OF_EMPLOYEES THE_YEAR
-----
29                2005
24                2006
19                2007
11                2008
```

## 2.4 Σύνδεση πινάκων

### 2.4.1 Γενικά

Σε πολλές περιπτώσεις υπάρχουν απαιτήσεις που προϋποθέτουν τον συνδυασμό των δεδομένων από διαφορετικούς πίνακες. Για παράδειγμα, στην παρακάτω εικόνα δημιουργείτε μια αναφορά η οποία συνδυάζει τους πίνακες *EMPLOYEES* και *DEPARTMENTS* έτσι ώστε να εμφανίσει το *ID* από τον πίνακα *EMPLOYEES*, το *department id* από τον πίνακα *EMPLOYEES* ή *DEPARTMENTS* και το *department name* από τον πίνακα *DEPARTMENTS*. Για να εμφανιστεί αυτή η πληροφορία θα πρέπει να χρησιμοποιήσουμε ένα τρόπο σύνδεσης αυτών των πινάκων.

EMPLOYEES				DEPARTMENTS			
EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID		DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	
1	King	90		1	Administration	1700	
2	Kochhar	90		2	Marketing	1800	
3	De Haan	90		3	Shipping	1500	
...				4	IT	1400	
16	Fay	20		5	Sales	2500	
19	Whalen	110		6	Executive	1700	
20	Seliz	110		7	Accounting	1700	
				8	Contracting	1700	

EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	200	10 Administration
2	201	20 Marketing
3	202	20 Marketing
4	124	50 Shipping
5	144	50 Shipping
...		
18	205	110 Accounting
19	206	110 Accounting

Η αντίστοιχη θεωρία παρουσιάστηκε στην Σχεσιακή Άλγεβρα στην ενότητα της σύνδεσης των πινάκων.

#### 2.4.2 Σύνταξη των ερωτημάτων σύνδεσης πινάκων (SQL:1999 πρότυπο)

Η σύνταξη για την σύνδεση πινάκων σύμφωνα με το πρότυπο SQL:1999 δείχνεται παρακάτω όπου το

```
SELECT table1.column, table2.column
FROM table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
ON (table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
ON (table1.column_name = table2.column_name)] |
[CROSS JOIN table2];
```

*table1.column* δηλώνει τον πίνακα και την στήλη από την οποία θα αντληθούν δεδομένα

το *NATURAL JOIN* συνδέει δύο πίνακες βασιζόμενο στο ίδιο όνομα στήλης

το *JOIN table2 USING column\_name* δηλώνει μια σύνδεση ισότητας βασιζόμενη στο όνομα της στήλης

το *JOIN table2 ON table1.column\_name=table2.column\_name* πραγματοποιεί σύνδεση ισότητας βασιζόμενη στην συνθήκη της έκφρασης ON

τα *LEFT/RIGHT/FULL OUTER* χρησιμοποιούνται για τις εξωτερικές συνδέσεις

το *CROSS JOIN* χρησιμοποιείται για την παραγωγή καρτεσιανού γινομένου

#### 2.4.3 Μετονομασία κατά την σύνδεση πινάκων

Κατά την σύνδεση δύο ή περισσότερων πινάκων δημιουργείται η ανάγκη προσδιορισμού των στηλών με την χρήση του ονόματος του πίνακα από τον οποίο προέρχονται ως πρόθεμα για την αποφυγή συγχύσεων. Για παράδειγμα κατά την σύνδεση των πινάκων

*DEPARTMENTS* και *EMPLOYEES* χωρίς την χρήση προθέματος στην στήλη *DEPARTMENT\_ID* δεν θα ήταν κατανοητό από πού προέρχεται. Η ανάγκη αυτή δεν υπάρχει όταν στους πίνακες που χρησιμοποιούμε δεν υπάρχουν στήλες με κοινά ονόματα.

Μια άλλη δυνατότητα που μπορούμε να χρησιμοποιήσουμε κατά την εκτέλεση ενός ερωτήματος, και όχι μόνο κατά την σύνδεση πινάκων, είναι αυτή της μετονομασίας ενός πίνακα. Η ενέργεια αυτή είναι ιδιαίτερα χρήσιμη όταν τα ονόματα των πινάκων είναι μεγάλα. Π.χ.

```
SQL> select e.employee_id, d.department_name
2   from employees e JOIN departments d
3   using (department_id);
```

εδώ ο πίνακας *EMPLOYEES* μετονομάζεται σε *e* και ο πίνακας *DEPARTMENTS* σε *d* ενώ ταυτόχρονα γίνεται χρήση προθεμάτων στις στήλες που επιλέγονται.

#### 2.4.4 Φυσική σύνδεση πινάκων

Με την φυσική σύνδεση μπορούν να συνδεθούν δύο πίνακες με βάση τις στήλες τους οι οποίες έχουν κοινά ονόματα και τύπους. Η ενέργεια αυτή μπορεί να γίνει με την χρήση της έκφρασης *NATURAL JOIN* και επιστρέφει όλες εκείνες τις εγγραφές που έχουν ίδιες τιμές σε όλες τις κοινές στήλες.

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	60	IT	1400	Southlake
2	50	Shipping	1500	South San Francisco
3	10	Administration	1700	Seattle
4	90	Executive	1700	Seattle
5	110	Accounting	1700	Seattle
6	190	Contracting	1700	Seattle
7	20	Marketing	1800	Toronto
8	80	Sales	2500	Oxford

Στο παραπάνω παράδειγμα ο πίνακας *LOCATIONS* συνδέεται με φυσική σύνδεση με τον πίνακα *DEPARTMENTS* μέσω της μοναδικής κοινής στήλης τους, την *LOCATION\_ID*. Κατά την φυσική σύνδεση μπορεί να γίνει ταυτόχρονα χρήση και άλλων εκφράσεων που έχουμε δει όπως π.χ του *WHERE*

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations
WHERE  department_id IN (20, 50);
```

#### 2.4.5 Σύνδεση πινάκων με την χρήση του USING

Κατά την σύνδεση πινάκων όπου υπάρχουν στήλες με κοινά ονόματα αλλά διαφορετικούς τύπους, η σύνδεση μπορεί να γίνει με την χρήση της έκφρασης *USING* με την οποία καθορίζουμε ποια (μόνο μια) από τις στήλες θα συμμετέχει. Για παράδειγμα συνδέουμε τους πίνακες *EMPLOYEES* και *DEPARTMENTS* μέσω του *department\_id*.

```
SELECT employee_id, last_name,
       location_id, department_id
FROM   employees JOIN departments
      USING (department_id);
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200 Whalen	1700	10
2	201 Hartstein	1800	20
3	202 Fay	1800	20
4	124 Mourgou	1500	50
5	144 Vargas	1500	50
6	143 Matos	1500	50
7	142 Davies	1500	50
8	141 Raju	1500	50
9	107 Lorentz	1400	60
10	104 Ernst	1400	60
...			
19	205 Higgins	1700	110

#### 2.4.6 Σύνδεση πινάκων με την χρήση του ON

Κατά την σύνδεση πινάκων με την χρήση της έκφρασης *ON* μας δίνεται η δυνατότητα να ορίσουμε εκείνες τις στήλες που συμμετέχουν στην σύνδεση αλλά και ορίσουμε και άλλου είδους σύνδεση εκτός από σύνδεση ισότητας. Επίσης βοηθά στο να γίνεται το ερώτημα ευκολότερα κατανοητό.

Στην περίπτωση όπου οι συνδεόμενοι πίνακες έχουν ίδια ονόματα στηλών είναι απαραίτητη η χρήση προθέματος με τον όνομα ή την μετονομασία του πίνακα. Μια τέτοια περίπτωση δείχνεται πιο κάτω και αφορά το πεδίο *department\_id*.

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
      ON (e.department_id = d.department_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200 Whalen	10	10	1700
2	201 Hartstein	20	20	1800
3	202 Fay	20	20	1800
4	124 Mourgou	50	50	1500
5	144 Vargas	50	50	1500
6	143 Matos	50	50	1500
7	142 Davies	50	50	1500
8	141 Raju	50	50	1500
9	107 Lorentz	60	60	1400
10	104 Ernst	60	60	1400

Πέρα από αυτό όμως, με το *ON* δίνεται η δυνατότητα να γίνει σύνδεση σε στήλες με διαφορετικά ονόματα, αλλά και σύνδεση σε περισσότερους από δύο πίνακες. Για παράδειγμα συνδέουμε τους πίνακες *EMPLOYEES* και *DEPARTMENTS* μέσω του πεδίου

*department\_id* και τους πίνακες *DEPARTMENTS* και *LOCATIONS* για να ανακτήσουμε την πληροφορία που θέλουμε από κάθε πίνακα.

Σύμφωνα με το πρότυπο SQL:1999 οι συνδέσεις πραγματοποιούνται από αριστερά προς τα δεξιά οπότε η πρώτη σύνδεση είναι *EMPLOYEES JOIN DEPARTMENTS*. Η δεύτερη σύνδεση μπορεί να δημιουργήσει σύνδεση μεταξύ οποιονδήποτε διαθέσιμων πινάκων, στην περίπτωση μας μεταξύ *DEPARTMENTS* και *LOCATIONS*.

```
SELECT employee_id, city, department_name
FROM employees e
JOIN departments d
ON d.department_id = e.department_id
JOIN locations l
ON d.location_id = l.location_id;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100 Seattle	Executive
2	101 Seattle	Executive
3	102 Seattle	Executive
4	103 Southlake	IT
5	104 Southlake	IT
6	107 Southlake	IT
7	124 South San Francisco	Shipping
8	141 South San Francisco	Shipping

...

Με το ερώτημα αυτό βρίσκουμε το ID κάθε υπαλλήλου, το όνομα του τμήματος που ανήκει καθώς και το όνομα της πόλης στην οποία ανήκει το τμήμα. Το ίδιο αποτέλεσμα με την χρήση του *USING* θα μπορούσε να παραχθεί με το ερώτημα:

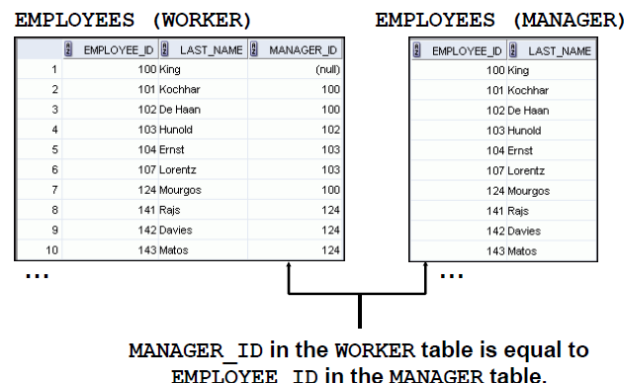
```
SELECT e.employee_id, l.city, d.department_name
FROM employees e
JOIN departments d
USING (department_id)
JOIN locations l
USING (location_id)
```

Στη σύνταξη της σύνδεσης μπορούμε επίσης να χρησιμοποιήσουμε το *WHERE* ή το *AND* για την εφαρμογή πρόσθετων συνθηκών που μας ενδιαφέρουν. Για παράδειγμα για να βρούμε την πληροφορία μόνο για τους υπαλλήλους με μάντζερ = 149 μπορούμε να γράψουμε μια από τις παρακάτω ισοδύναμες εκφράσεις.

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM employees e JOIN departments d
ON (e.department_id = d.department_id)
AND e.manager_id = 149 ;
```

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM employees e JOIN departments d
ON (e.department_id = d.department_id)
WHERE e.manager_id = 149 ;
```

## 2.4.7 Σύνδεση πίνακα με τον εαυτό του με την χρήση του ON



Σε κάποιες περιπτώσεις για να εξάγουμε την πληροφορία που θέλουμε είναι απαραίτητη η σύνδεση ενός πίνακα με τον εαυτό του. Για παράδειγμα για να βρούμε το όνομα του μάνατζερ κάθε εργαζόμενου είναι απαραίτητο να συνδέσουμε τον πίνακα *EMPLOYEES* με τον εαυτό του. Έτσι στην περίπτωση που θέλουμε να βρούμε τον μάνατζερ του υπαλλήλου *Lorentz's* θα χρειαστεί:

- Να βρούμε τον *Lorentz* με αναζήτηση στο πεδίο *LAST\_NAME*
- Να βρούμε τον αριθμό του μάνατζερ του από το πεδίο *MANAGER\_ID*. Είναι 103.
- Να βρούμε το όνομα του υπαλλήλου (μάνατζερ) με αριθμό, *EMPLOYEE\_ID=103*, από το πεδίο *LAST\_NAME*. Τελικά ο *Hunold* είναι ο μάνατζερ του *Lorentz*.

Όπως γίνεται κατανοητό με την διαδικασία αυτή προσπελαύνουμε τον πίνακα δύο φορές και αυτό μπορεί να γίνει με την χρήση του *ON* όπως φαίνεται στο παράδειγμα μας.

```
SQL> select worker.last_name employee, manager.last_name manager
2  from employees worker JOIN employees manager
3  on worker.manager_id = manager.employee_id
4  and worker.last_name='Lorentz';
```

EMPLOYEE	MANAGER
Lorentz	Hunold

## 2.4.8 Συνδέσεις μεταξύ πινάκων με την χρήση και άλλων τελεστών

\*Πριν την εκτέλεση της ενότητας αυτής συνδεθείτε μέσα από τον *SQL Developer* σαν χρήστης *HR* στην ΒΔ και πάρτε με αντιγραφή – επικόλληση τις πιο κάτω εντολές:

REM \*\*\*\*\*

REM Create the JOB\_GRADES table that will show different SALARY GRADES

REM depending on employee's SALARY RANGE

Prompt \*\*\*\*\* Creating JOB\_GRADES table ....



```
CREATE TABLE job_grades (  
    grade_level          CHAR(1),  
    lowest_sal           NUMBER(8,2) NOT NULL,  
    highest_sal          NUMBER(8,2) NOT NULL  
);  
  
ALTER TABLE job_grades  
ADD CONSTRAINT jobgrades_grade_pk PRIMARY KEY (grade);  
  
REM *****insert data into the JOB_GRADES table  
  
Prompt ***** Populating JOB_GRADES table ....  
  
INSERT INTO job_grades VALUES  
    ('A'  
    ,1000  
    ,2999  
    );  
  
INSERT INTO job_grades VALUES  
    ('B'  
    ,3000  
    ,5999  
    );  
  
INSERT INTO job_grades VALUES  
    ('C'  
    ,6000  
    ,9999  
    );
```

```
INSERT INTO job_grades VALUES
```

```
    ('D'  
  
    ,10000  
  
    ,14999  
  
    );
```

```
INSERT INTO job_grades VALUES
```

```
    ('E'  
  
    ,15000  
  
    ,24999  
  
    );
```

```
INSERT INTO job_grades VALUES
```

```
    ('F'  
  
    ,25000  
  
    ,40000  
  
    );
```

```
COMMIT;
```

```
REM *****
```

```
Prompt ***** END OF SCRIPT ....
```

EMPLOYEES

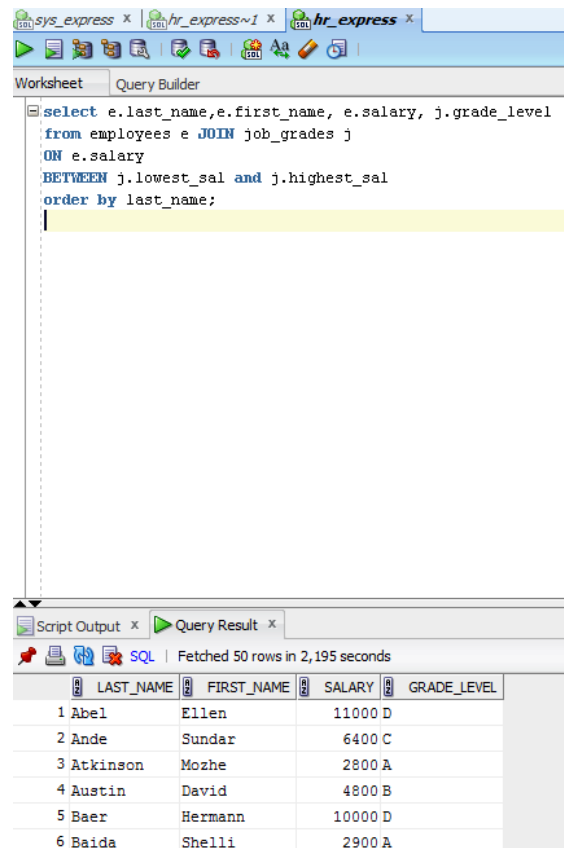
	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hunold	9000
5	Ernst	6000
6	Lorentz	4200
7	Mourgos	5800
8	Rajs	3500
9	Pavlov	3400

JOB\_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

Αφού εκτελέσετε τις πιο πάνω εντολές θα μελετήσετε την δυνατότητα της σύνδεσης πινάκων με την χρήση και τελεστών πέρα από την ισότητα (noequijoin). Για το παράδειγμα μας θα χρησιμοποιήσουμε στους πίνακες *EMPLOYEES* και *JOB\_GRADES* με σκοπό να βαθμολογήσουμε κάθε υπάλληλο βασιζόμενοι στην κλίμακα που προκύπτει από την ταξινόμηση του μισθού του μεταξύ *LOWEST\_SAL* και *HIGHEST\_SAL* (πίνακας

*JOB\_GRADES*). Για να γίνει αυτή η σύγκριση δεν χρησιμοποιούμε τον τελεστή της ισότητας (=) όπως δείχνεται παρακάτω αλλά τον τελεστή *BETWEEN*.



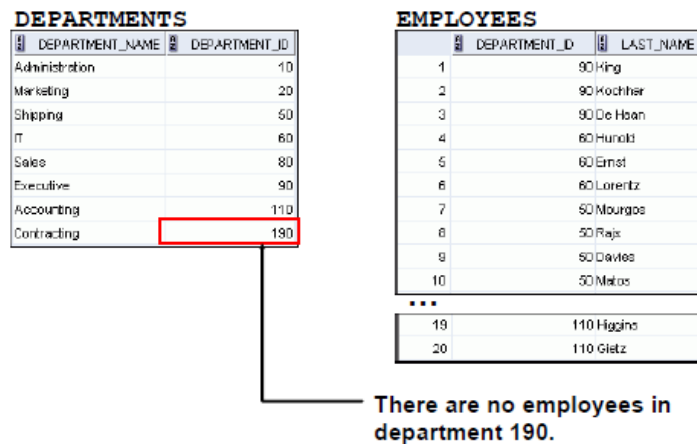
```
select e.last_name,e.first_name, e.salary, j.grade_level
from employees e JOIN job_grades j
ON e.salary
BETWEEN j.lowest_sal and j.highest_sal
order by last_name;
```

	LAST_NAME	FIRST_NAME	SALARY	GRADE_LEVEL
1	Abel	Ellen	11000	D
2	Ade	Sundar	6400	C
3	Atkinson	Mozhe	2800	A
4	Austin	David	4800	B
5	Baer	Hermann	10000	D
6	Baida	Shelli	2900	A

Από το αποτέλεσμα που προκύπτει βλέπουμε ότι κάθε υπάλληλος εμφανίζεται ΜΟΝΟ μία φορά στο αποτέλεσμα χωρίς να επαναλαμβάνεται. Αυτό συμβαίνει για δύο λόγους:

- Καμία εγγραφή με βάση τις τιμές της στα πεδία *LOWEST\_SAL*, *HIGHEST\_SAL* στον πίνακα *JOB\_GRADES* δεν υπερκαλύπτει την άλλη. Με αυτό τον τρόπο ο μισθός ενός υπαλλήλου μπορεί να βρίσκεται στα όρια που ορίζονται από μια μόνο εγγραφή.
- Οι μισθοί όλων των υπαλλήλων βρίσκονται μεταξύ των ορίων του πίνακα *JOB\_GRADES*. Αυτό σημαίνει ότι κανένας υπάλληλος δεν κερδίζει λιγότερα από την χαμηλότερη τιμή του *LOWEST\_SAL* ή περισσότερα από την υψηλότερη τιμή του *HIGHEST\_SAL*.

#### 2.4.9 Συνδέσεις μεταξύ πινάκων με εξωτερικές συνδέσεις



Στην περίπτωση που συνδέσουμε τους πίνακες *EMPLOYEES* και *DEPARTMENTS* το τμήμα με τον νούμερο π.χ. 190 δεν θα εμφανιστεί καθώς δεν υπάρχει κανένας υπάλληλος σε αυτό. Για την αποφυγή απώλειας πληροφορίας μπορούμε να χρησιμοποιήσουμε την εξωτερική σύνδεση ως τρόπο σύνδεσης πινάκων.

*SELECT DISTINCT department\_id FROM employees e NATURAL JOIN departments d ;*

The screenshot shows the result of a SQL query in a database interface. The window title is "Script Output x Query Res". The query is "SQL | All Rows Fr". The result is a table with one column, "DEPARTMENT\_ID", and eight rows of data.

DEPARTMENT_ID
1
100
2
30
3
90
4
20
5
110
6
50
7
80
8
60





#### 2.4.10 Αριστερή εξωτερική σύνδεση

Worksheet

Query Builder

```
select e.last_name, e.department_id, d.department_name
from employees e left outer join departments d
on (e.department_id = d.department_id);
```

Query Result x

    SQL | All Rows Fetched: 107 in 0,01 seconds

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
93	Errazuriz	80	Sales
94	Partners	80	Sales
95	Russell	80	Sales
96	De Haan	90	Executive
97	Kochhar	90	Executive
98	King	90	Executive
99	Popp	100	Finance
100	Urman	100	Finance
101	Sciarra	100	Finance
102	Chen	100	Finance
103	Faviet	100	Finance
104	Greenberg	100	Finance
105	Gietz	110	Accounting
106	Higgins	110	Accounting
107	Grant	(null)	(null)

Με το ερώτημα αυτό εμφανίζονται όλες οι εγγραφές του πίνακα *EMPLOYEES* ακόμη και αν δεν υπάρχει το τμήμα του υπαλλήλου στον πίνακα *DEPARTMENTS*.





#### 2.4.11 Δεξιά εξωτερική σύνδεση

Worksheet

Query Builder

```
select e.last_name, e.department_id, d.department_name
from employees e right outer join departments d
on (e.department_id = d.department_id);
```

Query Result x

    SQL | All Rows Fetched: 122 in 0,054 seconds

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
100	Urman	100	Finance
101	Chen	100	Finance
102	Faviet	100	Finance
103	Sciarra	100	Finance
104	Greenberg	100	Finance
105	Gietz	110	Accounting
106	Higgins	110	Accounting
107	(null)	(null)	Treasury
108	(null)	(null)	Corporate Tax
109	(null)	(null)	Control And Cr...
110	(null)	(null)	Shareholder Se...
111	(null)	(null)	Benefits
112	(null)	(null)	Manufacturing
113	(null)	(null)	Construction
114	(null)	(null)	Contracting

Με το ερώτημα αυτό εμφανίζονται όλες οι εγγραφές του πίνακα *DEPARTMENTS* ακόμη και αν δεν έχει υπάλληλο από τον πίνακα *EMPLOYEES*.

### 2.4.12 Πλήρης εξωτερική σύνδεση

```

select e.last_name, e.department_id, d.department_name
from employees e full outer join departments d
on (e.department_id = d.department_id)
order by 2 nulls first,3 nulls first;

```

Query Result x

SQL | Fetched 50 rows in 0,006 seconds

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Grant	(null)	(null)
2	(null)	(null)	Benefits
3	(null)	(null)	Construction
4	(null)	(null)	Contracting
5	(null)	(null)	Control And Credit
6	(null)	(null)	Corporate Tax
7	(null)	(null)	Government Sales
8	(null)	(null)	IT Helpdesk
9	(null)	(null)	IT Support

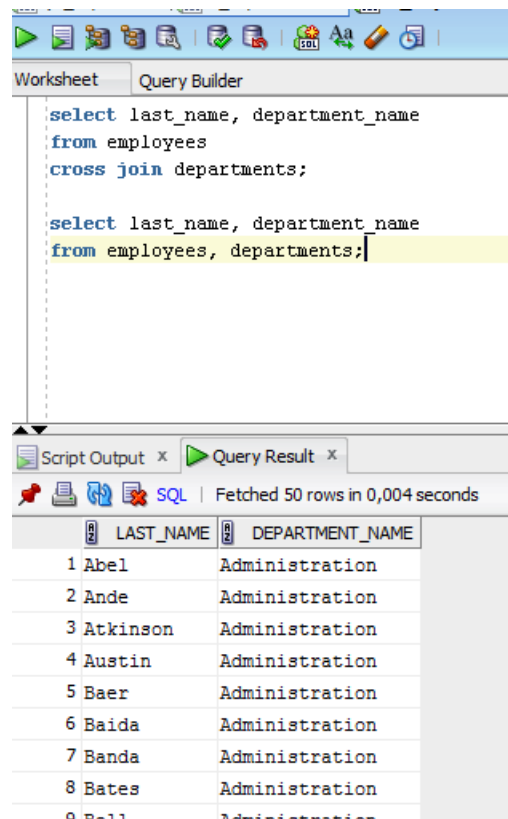
Με το ερώτημα αυτό εμφανίζονται τόσο όλες οι εγγραφές του πίνακα *DEPARTMENTS* όσο και όλες οι εγγραφές του *EMPLOYEES* με την συμπλήρωση NULL όπου χρειάζεται.

### 2.4.13 Καρτεσιανό γινόμενο

Σε περιπτώσεις όπου η συνθήκη της συνένωσης είναι εσφαλμένη ή παραλείπεται εσκεμμένα το αποτέλεσμα είναι η δημιουργία καρτεσιανού γινομένου όπου όλες οι γραμμές ενός πίνακα συνδυάζονται με όλες τις γραμμές του άλλου.

Το καρτεσιανό γινόμενο παράγει ένα μεγάλο αποτέλεσμα εγγραφών που σχεδόν πάντα ΔΕΝ έχει κάποια χρησιμότητα, εκτός από τις περιπτώσεις εκείνες που θέλουμε να εξομοιώσουμε την παραγωγή μεγάλου αριθμού δεδομένων.

Για την παραγωγή λοιπόν ενός καρτεσιανού γινομένου μεταξύ των πινάκων *EMPLOYEES* και *DEPARTMENTS* δείχνονται δύο ερωτήματα στην παρακάτω εικόνα τα οποία είναι ισοδύναμα. Στο πρώτο γίνεται χρήση της έκφρασης *CROSS JOIN*, ενώ στο δεύτερο παραλείπεται η συνθήκη σύνδεσης των πινάκων.



#### 2.4.14 Ασκήσεις

1. Γράψτε ένα ερώτημα για το τμήμα HR στο οποίο θα εμφανίζονται οι διευθύνσεις όλων των τμημάτων. Χρησιμοποιήστε τους πίνακες *LOCATIONS* και *COUNTRIES*. Στο ερώτημα να δείχνονται το *location\_id*, *street\_address*, *city*, *state\_province* και *country\_name*. Χρησιμοποιείστε το *NATURAL JOIN* για την σύνδεση των δύο πινάκων.

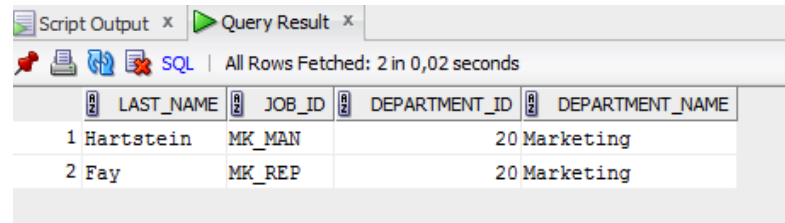
Script Output x Query Result x					
SQL   All Rows Fetched: 23 in 0,163 seconds					
	LOCATION_ID	STREET_ADDRESS	CITY	STATE_PROVINCE	COUNTRY_NAME
1	1000	1297 Via Cola di Rie	Roma	(null)	Italy
2	1100	93091 Calle della Testa	Venice	(null)	Italy
3	1200	2017 Shinjuku-ku	Tokyo	Tokyo Prefecture	Japan
4	1300	9450 Kamiya-cho	Hiroshima	(null)	Japan
5	1400	2014 Jabberwocky Rd	Southlake	Texas	United States of America
6	1500	2011 Tennessee Blvd	South San Francisco	California	United States of America

2. Το τμήμα HR θέλει να γνωρίζει το επίθετο, τον αριθμό τμήματος καθώς και το όνομα τμήματος όλων των υπαλλήλων.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Tobias	30	Purchasing
5	Colmenares	30	Purchasing
6	Baida	30	Purchasing
7	Raphaely	30	Purchasing
8	Khoo	30	Purchasing



- Εμφανίστε το επίθετο, το *job\_id*, τον αριθμό τμήματος καθώς και το όνομα τμήματος όλων των υπαλλήλων που εργάζονται στο Toronto. Χρησιμοποιήστε τους πίνακες *EMPLOYEES*, *DEPARTMENTS* και *LOCATIONS*

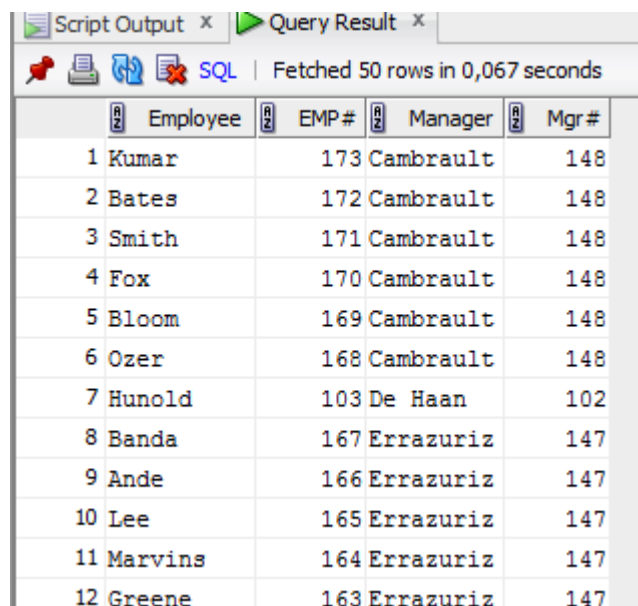


Script Output x Query Result x

SQL | All Rows Fetched: 2 in 0,02 seconds

	LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	Hartstein	MK_MAN	20	Marketing
2	Fay	MK_REP	20	Marketing

- Δημιουργήστε μια αναφορά για εμφανίσετε το επίθετο και τον αριθμό κάθε εργαζόμενου μαζί το επίθετο και τον αριθμό των μάνατζερ τους. Ονομάστε τα πεδία *Employee*, *Emp#*, *Manager* και *Mng#* αντίστοιχα. Σώστε το ερώτημα σας ως lab\_06\_04.sql

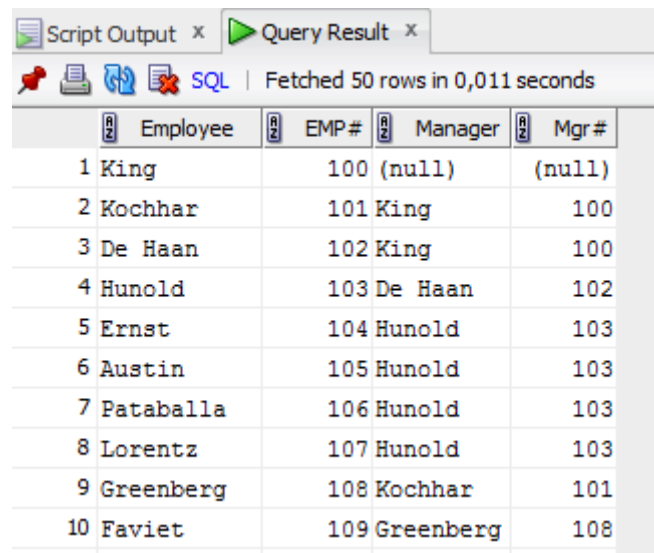


Script Output x Query Result x

SQL | Fetched 50 rows in 0,067 seconds

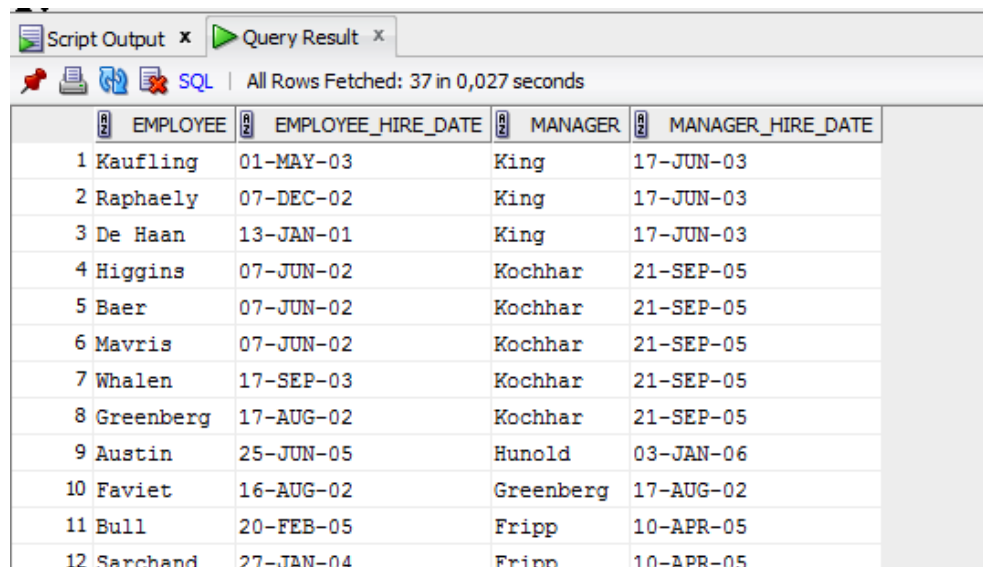
	Employee	EMP#	Manager	Mng#
1	Kumar	173	Cambrault	148
2	Bates	172	Cambrault	148
3	Smith	171	Cambrault	148
4	Fox	170	Cambrault	148
5	Bloom	169	Cambrault	148
6	Ozer	168	Cambrault	148
7	Hunold	103	De Haan	102
8	Banda	167	Errazuriz	147
9	Ande	166	Errazuriz	147
10	Lee	165	Errazuriz	147
11	Marvins	164	Errazuriz	147
12	Greene	163	Errazuriz	147

- Τροποποιήστε το ερώτημα lab\_06\_04.sql έτσι ώστε να εμφανίζεται και ο King, ο οποίος δεν έχει μάνατζερ. Ταξινομήστε το αποτέλεσμα κατά αριθμό υπαλλήλου. Σώστε το ερώτημα ως lab\_06\_05.sql



	Employee	EMP#	Manager	Mgr#
1	King	100	(null)	(null)
2	Kochhar	101	King	100
3	De Haan	102	King	100
4	Hunold	103	De Haan	102
5	Ernst	104	Hunold	103
6	Austin	105	Hunold	103
7	Pataballa	106	Hunold	103
8	Lorentz	107	Hunold	103
9	Greenberg	108	Kochhar	101
10	Faviet	109	Greenberg	108

6. **[Extra Challenge]** Εμφανίστε τα ονόματα, και την ημερομηνία πρόσληψης όλων των υπαλλήλων οι οποίοι προσελήφθησαν πριν από τους μάνατζερ τους μαζί με τα ονόματα των μάνατζερ και την ημερομηνία πρόσληψης αυτών.

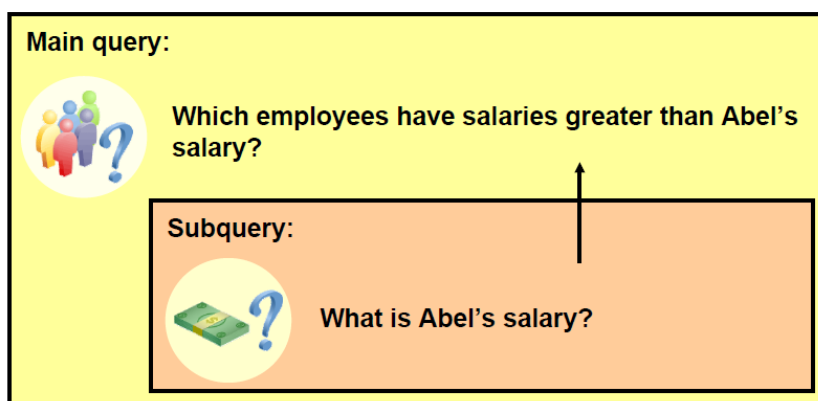


	EMPLOYEE	EMPLOYEE_HIRE_DATE	MANAGER	MANAGER_HIRE_DATE
1	Kaufling	01-MAY-03	King	17-JUN-03
2	Raphaely	07-DEC-02	King	17-JUN-03
3	De Haan	13-JAN-01	King	17-JUN-03
4	Higgins	07-JUN-02	Kochhar	21-SEP-05
5	Baer	07-JUN-02	Kochhar	21-SEP-05
6	Mavris	07-JUN-02	Kochhar	21-SEP-05
7	Whalen	17-SEP-03	Kochhar	21-SEP-05
8	Greenberg	17-AUG-02	Kochhar	21-SEP-05
9	Austin	25-JUN-05	Hunold	03-JAN-06
10	Faviet	16-AUG-02	Greenberg	17-AUG-02
11	Bull	20-FEB-05	Fripp	10-APR-05
12	Sarchand	27-JAN-04	Fripp	10-APR-05

## 2.5 Χρήση υποερωτημάτων

### 2.5.1 Γενικά

Ένα υποερώτημα είναι μια *select-from-where* έκφραση, η οποία είναι εμφωλευμένη σε μια άλλη ερώτηση. Σε πολλές περιπτώσεις υπάρχουν απαιτήσεις που προϋποθέτουν την χρήση υποερωτημάτων ώστε να παραχθεί το ζητούμενο αποτέλεσμα. Παράδειγμα ενός τέτοιου προβλήματος είναι “Ποιος υπάλληλος έχει μεγαλύτερο μισθό από τον μισθό της Abel;” (βλ. εικόνα παρακάτω).



### 2.5.2 Σύνταξη των υποερωτημάτων

Η σύνταξη για την χρήση υποερωτημάτων δείχνεται παρακάτω όπου το

```
SELECT  select_list
FROM    table
WHERE   expr operator
        (SELECT  select_list
         FROM    table);
```

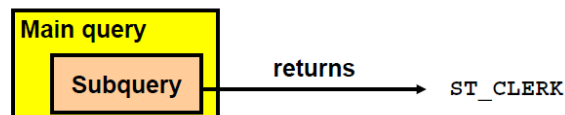
το υποερώτημα (εσωτερικό ερώτημα) εκτελείτε πριν από το κυρίως ερώτημα (εξωτερικό ερώτημα) και τα αποτελέσματα του στην συνέχεια χρησιμοποιούνται από το κυρίως ερώτημα. Για παράδειγμα στην πιο κάτω εικόνα πρώτα εκτελείτε το ερώτημα μέσα στις παρενθέσεις το οποίο μας επιστρέφει τον μισθό της Abel και στην συνέχεια το κυρίως ερώτημα.

```
SELECT last_name, salary
FROM   employees
WHERE  salary > 11000
        (SELECT salary
         FROM   employees
         WHERE  last_name = 'Abel');
```

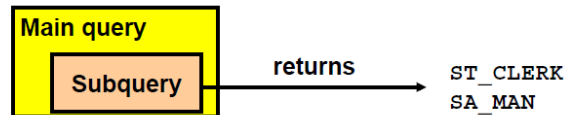
	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hartstein	13000
5	Higgins	12000

Τα υποερωτήματα εκτός από την χρήση τους στην έκφραση *WHERE* μπορούν να χρησιμοποιηθούν επίσης στο *SELECT*, στο *HAVING* και στο *FROM*. Θα πρέπει να περικλείονται από παρενθέσεις και σαν βέλτιστη πρακτική καλό είναι να τοποθετούνται στην δεξιά πλευρά της σύγκρισης ώστε να γίνεται ευκολότερη η ανάγνωση και η κατανόηση τους, επίσης δεν έχει νόημα η χρήση *ORDER BY* σε αυτά. Τέλος κατά περίπτωση, θα πρέπει να γίνεται και η χρήση των κατάλληλων τελεστών σύγκρισης, όπως θα δούμε, για υποερωτήματα μιας ή πολλαπλών γραμμών αντίστοιχα.

### Single-row subquery



### Multiple-row subquery



## 2.5.3 Υποερωτήματα μονής εγγραφής

Τα υποερωτήματα μονής εγγραφής επιστρέφουν μόνο μια εγγραφή και μπορούν να χρησιμοποιηθούν με τους αντίστοιχους τελεστές σύγκρισης μονής εγγραφής.

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Για παράδειγμα

```

SELECT last_name, job_id, salary
FROM employees
WHERE job_id = (SELECT job_id
                FROM employees
                WHERE last_name = 'Taylor')
AND salary > (SELECT salary
              FROM employees
              WHERE last_name = 'Taylor');
    
```

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

και τα δύο υποερωτήματα της εικόνας επιστρέφουν μια εγγραφή *SA\_REP*, 8600 αντίστοιχα και για αυτό χρησιμοποιούνται οι τελεστές σύγκρισης =, > .

Τα ερωτήματα γενικά μπορεί να χρησιμοποιούν πιο πολύπλοκές εκφράσεις για την παραγωγή του αποτελέσματος και τέτοιες περιπτώσεις χρήσης δείχνονται στα παραδείγματα που ακολουθούν. Στην πρώτη περίπτωση μέσα στο υποερώτημα

εμφανίζεται η συνάρτηση συνάθροισης *MIN* ενώ στη δεύτερη το παραγόμενο αποτέλεσμα χρησιμοποιείται σαν είσοδος στο *HAVING* (και όχι στο *WHERE*) για την παραγωγή των επιθυμητών εγγραφών.

```
SELECT last_name, job_id, salary
FROM employees
WHERE salary = (SELECT MIN(salary)
FROM employees);
```

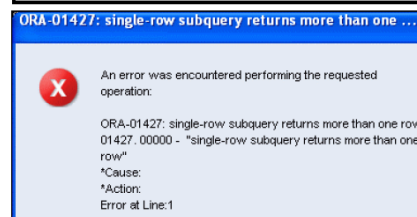
LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

```
SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) > (SELECT MIN(salary)
FROM employees
WHERE department_id = 50);
```

DEPARTMENT_ID	MIN(SALARY)
1	(null)
2	90
3	20
...	...
7	10
10	4400

Παρατηρούμε εδώ ότι υπάρχουν τελεστές μονής εγγραφής για υποερωτήματα μονής εγγραφής. Στην περίπτωση όμως που θέλουμε να επιστρέψουμε πολλαπλές εγγραφές ο συνδυασμός αυτός οδηγεί σε σφάλμα εκτέλεσης.

```
SELECT employee_id, last_name
FROM employees
WHERE salary = (SELECT MIN(salary)
FROM employees
GROUP BY department_id);
```



Single-row operator  
with multiple-row  
subquery

#### 2.5.4 Υποερωτήματα πολλαπλών εγγραφών

Η λύση στο πιο πάνω πρόβλημα είναι χρήση τελεστών σύγκρισης πολλαπλών εγγραφών για υποερωτήματα που επιστρέφουν πάνω από μια εγγραφή. Οι τελεστές αυτοί και οι λειτουργίες που πραγματοποιεί ο καθένας έχουν ως εξής:

Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. Compares a value to each value in a list or returned by a query. Evaluates to FALSE if the query returns no rows.
ALL	Must be preceded by =, !=, >, <, <=, >=. Compares a value to every value in a list or returned by a query. Evaluates to TRUE if the query returns no rows.

Ο τελεστής *IN* έχει μελετηθεί και στην παράγραφο 2.2.7 οπότε θα μελετήσουμε μόνο τους *ANY* και *ALL*.

### 2.5.5 Ο τελεστής *ALL*

Το *ALL* τοποθετείται μετά τον τελεστή σύγκρισης και πριν από το υποερώτημα. Οι συγκρίσεις με *ALL* αποδίδουν *TRUE*, αν ο τελεσταίος της σύγκρισης αποδίδει *TRUE* με όλες τις εγγραφές που επιστρέφει το υποερώτημα. Στην περίπτωση που η σύγκριση επιστρέψει *FALSE* για τουλάχιστον μια φορά, τότε η παράσταση επιστρέφει *FALSE*. Πρόσθετα αν το υποερώτημα επιστρέφει το κενό σύνολο τότε η σύγκριση αποδίδει *TRUE*, ενώ αν επιστραφεί *NULL*, τότε επιστρέφει *NULL*. Για παράδειγμα “Να βρεθούν οι υπάλληλοι των οποίων ο μισθός είναι μικρότερος από όλους τους μισθούς των υπαλλήλων με εργασία IT\_PROG”.

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL
      (SELECT salary
       FROM employees
       WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141	Rajs	ST_CLERK	3500
2	142	Davies	ST_CLERK	3100
3	143	Matos	ST_CLERK	2600
4	144	Vargas	ST_CLERK	2500

### 2.5.6 Ο τελεστής *ANY*

Το *ANY* τοποθετείται και αυτός μετά τον τελεστή σύγκρισης και πριν από το υποερώτημα. Οι συγκρίσεις με *ANY* αποδίδουν *TRUE*, αν ο τελεσταίος της σύγκρισης αποδίδει *TRUE* με οποιαδήποτε από τις εγγραφές που επιστρέφει το υποερώτημα. Στην περίπτωση που η σύγκριση επιστρέψει *FALSE* για όλες τις εγγραφές, τότε και μόνο τότε η παράσταση επιστρέφει *FALSE*. Πρόσθετα αν το υποερώτημα επιστρέφει το κενό σύνολο τότε η σύγκριση αποδίδει *FALSE*, ενώ αν επιστραφεί *NULL*, τότε επιστρέφει *NULL*. Για παράδειγμα “Να βρεθούν οι υπάλληλοι των οποίων ο μισθός είναι μικρότερος για οποιοδήποτε από τους μισθούς των υπαλλήλων με εργασία IT\_PROG”.

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ANY
      (SELECT salary
       FROM employees
       WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
3	142	Davies	ST_CLERK	3100
4	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400
...				
9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

## 2.5.7 Ασκήσεις

1. Γράψτε ένα ερώτημα για το τμήμα HR στο οποίο θα εμφανίζεται το id, το επώνυμο και ο μισθός όλων των υπαλλήλων οι οποίοι έχουν αποδοχές μεγαλύτερες του μέσου όρου των αποδοχών όλων των υπαλλήλων. Ταξινομήστε κατά μισθό σε αύξουσα σειρά.

Query Result x

SQL | Fetched 50 rows in 0,02 seconds

	EMPLOYEE_ID	LAST_NAME	SALARY
1	203	Mavris	6500
2	123	Vollman	6500
3	165	Lee	6800
4	113	Popp	6900
5	155	Tuvault	7000
6	161	Sewall	7000
7	178	Grant	7000
8	164	Marvins	7200
9	172	Bates	7300
10	171	Smith	7400
11	154	Cambrault	7500
12	160	Doran	7500
13	111	Sciarra	7700

2. Το τμήμα HR θέλει να γνωρίζει το επίθετο, τον αριθμό τμήματος καθώς και το *JOB\_ID* όλων των υπαλλήλων με *LOCATION\_ID* του τμήματος να είναι 1700.

Query Result x

SQL | All Rows Fetched: 18 in 0,008 seconds

	LAST_NAME	DEPARTMENT_ID	JOB_ID
1	King	90	AD_PRES
2	Kochhar	90	AD_VP
3	De Haan	90	AD_VP
4	Greenberg	100	FI_MGR
5	Faviet	100	FI_ACCOUNT
6	Chen	100	FI_ACCOUNT
7	Sciarra	100	FI_ACCOUNT
8	Urman	100	FI_ACCOUNT
9	Popp	100	FI_ACCOUNT
10	Raphaely	30	PU_MAN
11	Khoo	30	PU_CLERK

3. Δείξτε τους υπαλλήλους των οποίων ο μισθός είναι μεγαλύτερος από τον οποιοδήποτε μισθό των υπαλλήλων του τμήματος 60.

Query Result x

SQL | Fetched 50 rows in 0,017 seconds

	LAST_NAME
1	King
2	Kochhar
3	De Haan
4	Russell
5	Partners
6	Hartstein
7	Greenberg
8	Higgins
9	Errazuriz
10	Ozer
11	Raphaely

4. **[Extra Challenge]** Εμφανίστε τα επίθετα, και τον μισθό όλων των υπαλλήλων οι οποίοι με μισθό μεγαλύτερο του μέσου μισθού και που εργάζονται στο τμήμα όπου κάποιος υπάλληλος έχει το γράμμα “u” στο επώνυμο του.

Query Result x

SQL | All Rows Fetched: 36 in 0,038 seconds

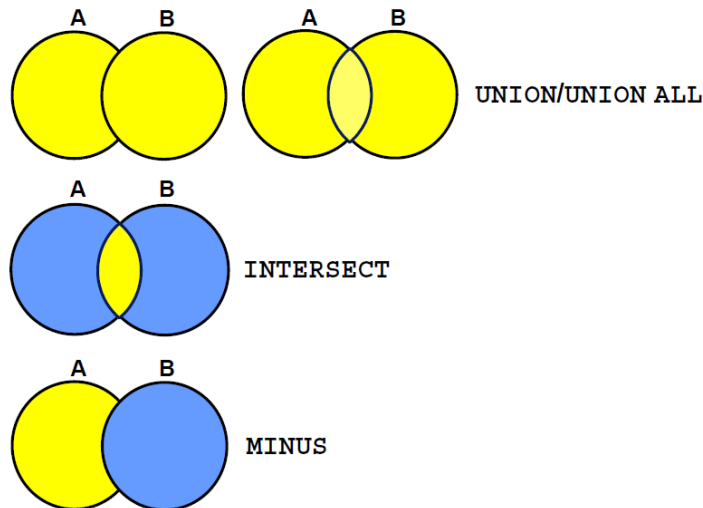
	LAST_NAME
1	Hunold
2	Vollman
3	Kaufling
4	Fripp
5	Weiss
6	Livingston
7	Taylor
8	Hutton
9	Abel
10	Bates
11	Smith



## 2.6 Τελεστές Συνόλων

### 2.6.1 Γενικά

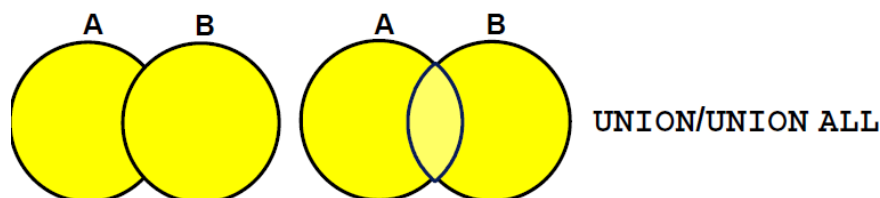
Οι τελεστές συνόλων που έχουμε στην διάθεση μας είναι οι *UNION/UNION ALL*, *INTERSECT* και *MINUS* και όπως γνωρίζουμε και από την θεωρία μπορούν να εφαρμοστούν μόνο σε συμβατές σχέσεις.



Πιο συγκεκριμένα θα πρέπει ο αριθμός των στηλών που καλούνται στην πρώτη έκφραση να είναι ίδιος με τον αριθμό των στηλών που καλούνται στις επόμενες εκφράσεις. Επίσης οι στήλες (πεδίο ορισμού) θα πρέπει να είναι συμβατές μεταξύ τους ενώ η χρήση του *ORDER BY* επιτρέπεται μόνο στο τέλος της συνολικής έκφρασης. Γενικά η σύνταξη της SQL για τους τελεστές συνόλων έχει ως εξής

```
SELECT select_list
FROM table1
WHERE expr
UNION/UNION ALL/INTERSECT/MINUS
SELECT select_list
FROM table2
WHERE expr
```

### 2.6.2 Ο τελεστής *UNION/UNION ALL*



Ο τελεστής *UNION* επιστρέφει τις όλες εγγραφές όλων των ερωτημάτων που ενώνονται εξαλείφοντας τις διπλές εγγραφές. Στην περίπτωση που θέλουμε αυτές να εμφανίζονται κάνουμε χρήση του τελεστή *UNION ALL*. Για παράδειγμα “Εμφάνισε την τρέχουσα και την προηγούμενη εργασία όλων των υπαλλήλων χωρίς διπλοεγγραφές”

```
SELECT employee_id, job_id
FROM employees
UNION
SELECT employee_id, job_id
FROM job_history;
```

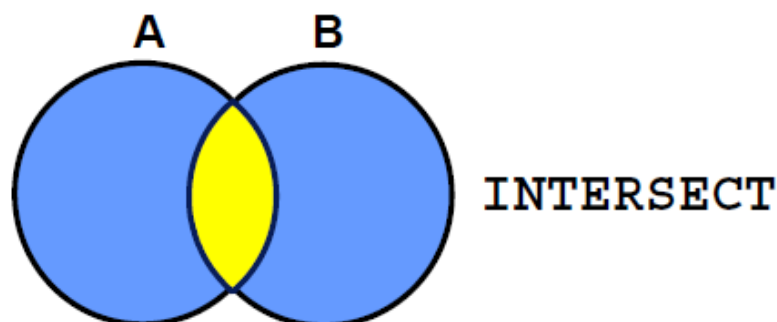
EMPLOYEE_ID	JOB_ID
1	100 AD_PRES
2	101 AC_ACCOUNT
...	
22	200 AC_ACCOUNT
23	200 AD_ASST
24	201 MK_MAN
...	

“Εμφάνισε την τρέχουσα και την προηγούμενη εργασία όλων των υπαλλήλων ”

```
SELECT employee_id, job_id, department_id
FROM employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM job_history
ORDER BY employee_id;
```

EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100 AD_PRES	90
...		
16	144 ST_CLERK	50
17	149 SA_MAN	80
18	174 SA_REP	80
19	176 SA_REP	80
20	176 SA_MAN	80
21	176 SA_REP	80
22	178 SA_REP	(null)
...		
30	206 AC_ACCOUNT	110

### 2.6.3 Ο τελεστής *INTERSECT*

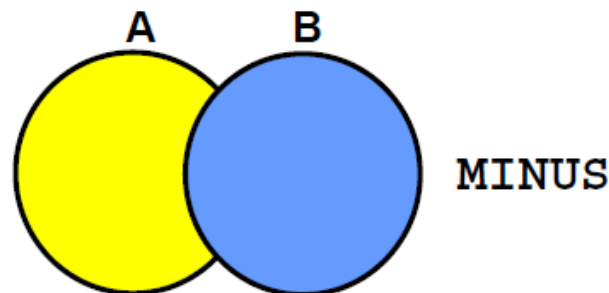


Ο τελεστής *INTERSECT* επιστρέφει τις κοινές εγγραφές των ερωτημάτων που ενώνονται και αντιστοιχεί στον τελεστή Τομή της Σχεσιακής Άλγεβρας. Για παράδειγμα “Εμφάνισε το *EMPLOYEE\_ID* και το *JOB\_ID* των εργαζομένων των οποίων η εργασία είναι ίδια (κοινή) με μια προηγούμενη (άλλαξαν εργασία αλλά ύστερα ξανάλλαξαν στην πρότερη εργασία τους)”

```
SELECT employee_id, job_id
FROM employees
INTERSECT
SELECT employee_id, job_id
FROM job_history;
```

	EMPLOYEE_ID	JOB_ID
1	176	SA_REP
2	200	AD_ASST

#### 2.6.4 Ο τελεστής MINUS



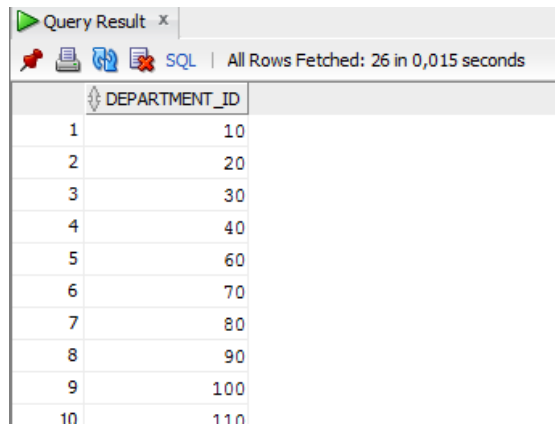
Ο τελεστής *MINUS* επιστρέφει όλες τις διακριτές εγγραφές του πρώτου μέρους οι οποίες όμως δεν βρίσκονται στο δεύτερο μέρος. Αντιστοιχεί στον τελεστή Αφαίρεση της Σχεσιακής Άλγεβρας. Για παράδειγμα “Εμφάνισε το *EMPLOYEE\_ID* των εργαζομένων των που δεν έχουν αλλάξει εργασία καμία φορά”

```
SELECT employee_id
FROM employees
MINUS
SELECT employee_id
FROM job_history;
```

	EMPLOYEE_ID
1	100
2	103
3	104
4	107
5	124
...	
14	205
15	206

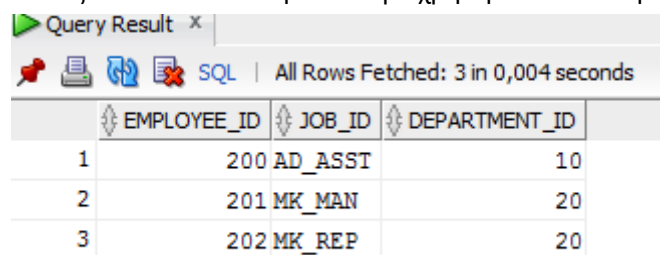
#### 2.6.5 Ασκήσεις

1. Το HR ενδιαφέρεται για τα ID των τμημάτων εκείνων τα οποία δεν περιέχουν το *JOB\_ID* “*ST\_CLERK*”. Να το γράψετε με τελεστές συνόλων.



	DEPARTMENT_ID	
1	10	
2	20	
3	30	
4	40	
5	60	
6	70	
7	80	
8	90	
9	100	
10	110	

2. Δημιουργήστε μια αναφορά με το *EMPLOYEE\_ID*, *JOB\_ID*, *DEPARTMENT\_ID* όλων των υπαλλήλων των τμημάτων με *DEPARTMENT\_ID* “10” και “20”. Να το γράψετε με τελεστές συνόλων. Επαληθεύστε με χρήση του τελεστή *IN*.



	EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	200	AD_ASST	10
2	201	MK_MAN	20
3	202	MK_REP	20

## 2.7 Χειρισμός Δεδομένων

### 2.7.1 Γενικά

REM \*\*\*\*\*

REM Create the COPY\_EMP table that will hold same data as EMPLOYEES table

Prompt \*\*\*\*\* Creating COPY\_EMP table ....

CREATE TABLE COPY\_EMP AS SELECT \* FROM EMPLOYEES;

COMMIT;

REM \*\*\*\*\*

Prompt \*\*\*\*\* END OF SCRIPT ....

Η Γλώσσα Χειρισμού Δεδομένων/Data Manipulation Language (DML) αποτελεί αναπόσπαστο τμήμα της SQL. Κατά την ενημέρωση, την προσθήκη ή την διαγραφή δεδομένων εκτελείτε πάντα μια DML εντολή. Μια λογική ομάδα από εντολές DML ονομάζεται συναλλαγή/transaction.

Για παράδειγμα κατά την συναλλαγή με την ΒΔ μιας τράπεζας, όταν κάποιος πελάτης μεταφέρει λεφτά από ένα λογαριασμό σε ένα άλλο, η συναλλαγή που κάνει μπορεί να

αποτελείτε από τρεις διαφορετικές ενέργειες: μείωση ποσού στον πρώτο λογαριασμό, αύξηση του ποσού στον δεύτερο λογαριασμό και καταγραφή της κίνησης στον κατάλογο των κινήσεων. Κατά την εκτέλεση αυτής της συναλλαγής πρέπει να διασφαλίζεται ότι στην περίπτωση που κάποια από τις ενέργειες αποτύχει θα πρέπει να αναιρεθεί το σύνολο των ενεργειών, δηλαδή όλη η συναλλαγή/transaction.

Στην παράγραφο αυτή παρουσιάζονται οι εντολές *INSERT*, *UPDATE*, *DELETE* της *SQL* οι οποίες αφορούν τις μεταβολές των εγγραφών, όποιος ενδιαφέρεται μπορεί να αναζητήσει και την εντολή *MERGE* η οποία σύμφωνα με το πρότυπο *SQL:2003* αποτελεί και αυτή εντολή μεταβολής δεδομένων.

### 2.7.2 Εισαγωγή Δεδομένων - Η εντολή *INSERT*

<b>DEPARTMENTS</b>				70   Public Relations	100	1700	<b>New row</b>
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID				
10	Administration	200	1700				
20	Marketing	201	1800				
50	Shipping	124	1500				
60	IT	103	1400				
80	Sales	149	2500				
90	Executive	100	1700				
110	Accounting	205	1700				
190	Contracting		1700				

Insert a new row into the DEPARTMENTS table.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700
70	Public Relations	100	1700

Στην παραπάνω εικόνα δείχνεται η εισαγωγή ενός νέου τμήματος στον πίνακα *DEPARTMENTS*. Η εντολή της *SQL* για την εισαγωγή δεδομένων στα ΣΔΒΔ είναι η *INSERT* και η σύνταξη της έχει ως εξής:

```
INSERT INTO table [(column [, column...])]
VALUES (value [, value...]);
```

όπου *table* είναι το όνομα του πίνακα, *column* είναι το όνομα της στήλης στην οποία θα εισάγουμε τιμή και *value* είναι η τιμή την οποία θέλουμε να εισάγουμε. Στην περίπτωση όπου θέλουμε να εισάγουμε μια νέα γραμμή σε ένα πίνακα η οποία περιέχει τιμές για όλα τα πεδία του πίνακα δεν είναι απαραίτητο να γράψουμε τα ονόματα όλων των στηλών αρκεί οι τιμές να δίνονται με την ίδια σειρά με την οποία έχουν οριστεί και οι στήλες στον πίνακα. Εάν θέλουμε να εισάγουμε συμβολοσειρές ή ημερομηνίες αυτές πρέπει να περικλείονται σε μονά εισαγωγικά π.χ.

```
INSERT INTO departments(department_id, department_name,
                        manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
1 row created.
```

Στην περίπτωση που δεν δηλωθεί η τιμή μιας στήλης τότε είτε άμεσα είτε έμμεσα λαμβάνει “τιμή” *NULL*. Η άμεση εισαγωγή *NULL* μπορεί να γίνει παραλείποντας την στήλη από την λίστα των στηλών ενώ έμμεσα γίνεται δηλώνοντας *NULL* στις τιμές των αντίστοιχων στηλών. Για παράδειγμα:

```
INSERT INTO departments (department_id,
                        department_name )
VALUES (30, 'Purchasing');
1 row created.
```

```
INSERT INTO departments
VALUES (100, 'Finance', NULL, NULL);
1 row created.
```

Κατά την εισαγωγή νέων δεδομένων ελέγχονται όλοι περιορισμοί ακεραιότητας π.χ. περιορισμοί πεδίου τιμών, περιορισμοί αναφορικής ακεραιότητας (ξένο κλειδί), περιορισμοί ακεραιότητας οντότητας (πρωτεύων κλειδί), κ.α. Στην περίπτωση που υπάρχει παραβίαση κάποιου περιορισμού τότε η συναλλαγή αναιρείται και η εγγραφή δεν εισάγεται.

### 2.7.3 Ενημέρωση Δεδομένων - Η εντολή *UPDATE*

#### EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMM.
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	80	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	80	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	80	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

Update rows in the **EMPLOYEES** table.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMM.
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	80	30
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	80	30
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	80	30
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

Στην παραπάνω εικόνα δείχνεται η ενημέρωση των αριθμών των τμημάτων ορισμένων υπαλλήλων στον πίνακα *EMPLOYEES*. Η εντολή της SQL για την ενημέρωση δεδομένων στα ΣΔΒΔ είναι η *UPDATE* και η σύνταξη της έχει ως εξής:

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

όπου *table* είναι το όνομα του πίνακα, *column* είναι το όνομα της στήλης στην οποία θα εισάγουμε τιμή, *value* είναι η τιμή την οποία θέλουμε να εισάγουμε και *condition* είναι η

συνθήκη με την οποία καθορίζουμε ποια/ες εγγραφές θα ενημερωθούν. Στην περίπτωση που παραλείψουμε το *WHERE* ενημερώνονται όλες οι εγγραφές του πίνακα.

```
UPDATE copy_emp
SET department_id = 110;
22 rows updated.
```

Κατά την ενημέρωση εγγραφών μπορεί να γίνει χρήση εμφωλευμένων υποερωτημάτων για την ενημέρωση των τιμών των αντίστοιχων στηλών. Με το πιο κάτω παράδειγμα, με υποερωτήματα, ενημερώνεται το *JOB\_ID*, *SALARY* του υπαλλήλου *114* έτσι να είναι όμοια με αυτά του *205*.

```
UPDATE employees
SET job_id = (SELECT job_id
               FROM employees
               WHERE employee_id = 205),
    salary = (SELECT salary
               FROM employees
               WHERE employee_id = 205)
WHERE employee_id = 114;
1 row updated.
```

Η σύνταξη αυτής της περίπτωσης έχει ως εξής:

```
UPDATE table
SET column =
    (SELECT column
     FROM table
     WHERE condition)
[ ,
  column =
    (SELECT column
     FROM table
     WHERE condition)]
[WHERE condition] ;
```

Μια παραλλαγή της πιο πάνω περίπτωσης είναι η χρήση υποερωτημάτων στην παράσταση της συνθήκης – *WHERE*. Εδώ ενημερώνεται το *DEPARTMENT\_ID* όλων εκείνων των οποίων το *JOB\_ID* είναι ίδιο με το *JOB\_ID* του υπαλλήλου *200*.

```
UPDATE copy_emp
SET department_id = (SELECT department_id
                     FROM employees
                     WHERE employee_id = 100)
WHERE job_id = (SELECT job_id
                 FROM employees
                 WHERE employee_id = 200);
1 row updated.
```

Κατά όμοιο τρόπο στην περίπτωση που παραβιαστεί κάποιος περιορισμός ακεραιότητας τότε εμφανίζεται μήνυμα σφάλματος και η ενημέρωση αναιρείται.

```
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110;
```

```
UPDATE employees
*
ERROR at line 1:
ORA-02291: integrity constraint (HR.EMP_DEPT_FK)
violated - parent key not found
```

#### 2.7.4 Διαγραφή Δεδομένων - Η εντολή DELETE

##### DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
70	Public Relations	100	1700
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400
100	Finance		
80	Sales	149	2500

##### Delete a row from the DEPARTMENTS table.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
70	Public Relations	100	1700
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

Στην παραπάνω εικόνα δείχνεται η διαγραφή του τμήματος *FINANCE* από τον πίνακα *DEPARTMENTS*. Η εντολή της SQL για την διαγραφή δεδομένων στα ΣΔΒΔ είναι η *DELETE* και η σύνταξη της έχει ως εξής:

```
DELETE [FROM] table
[WHERE condition];
```

όπου *table* είναι το όνομα του πίνακα και *condition* είναι η συνθήκη με την οποία καθορίζουμε ποια/ες εγγραφές θα διαγραφούν. Στην περίπτωση που παραλείψουμε το *WHERE* διαγράφονται όλες οι εγγραφές του πίνακα.

```
DELETE FROM employees
WHERE employee_id = 114;
```

```
DELETE FROM copy_emp;
22 rows deleted.
```



Παρόμοια με την ενημέρωση μπορεί να γίνει χρήση εμφωλευμένων υποερωτημάτων για τον καθορισμό των εγγραφών που θα διαγραφούν από τον πίνακα. Με το πιο κάτω παράδειγμα, με υποερώτημα διαγράφονται όλοι υπάλληλοι των οποίων το *DEPARTMENT\_ID* είναι ίδιο με αυτό του τμήματος που έχουν την συμβολοσειρά *Public* στο όνομα του.

```
DELETE FROM employees
WHERE department_id =
      (SELECT department_id
       FROM departments
       WHERE department_name LIKE '%Public%');

1 row deleted.
```

Και κατά την διαγραφή ελέγχονται, με παρόμοιο τρόπο, οι περιορισμοί ακεραιότητας που τυχόν υπάρχουν.

### 2.7.5 *DEFAULT* τιμές

Σύμφωνα με το πρότυπο SQL:1999 μπορεί να γίνει η χρήση της έκφρασης *DEFAULT* κατά εισαγωγή ή κατά την ενημέρωση των εγγραφών έτσι ώστε η στήλη να λαμβάνει σαν τιμή αυτή που έχει οριστεί ως προκαθορισμένη. Για να γίνει καλύτερα αυτό κατανοητό παρακάτω εμφανίζεται η έκφραση δημιουργίας πίνακα (δεν έχουμε αναφερθεί σε αυτό ακόμη) και στην στήλη *City* ορίζεται ως προκαθορισμένη τιμή η *Αθήνα*.

```
CREATE TABLE Persons
(
  P_Id int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255) DEFAULT 'Αθήνα'
)
```

Στην περίπτωση που δεν ορίζεται τιμή και έχει οριστεί προκαθορισμένη τιμή για την στήλη εισάγετε η προκαθορισμένη τιμή (άμεσος τρόπος). Αν δεν έχει οριστεί προκαθορισμένη τιμή εισάγεται – όπως έχουμε δει – *NULL*. Η χρήση της λέξης *DEFAULT* μπορεί να γίνει με τον έμμεσο τρόπο όπως είπαμε στις εκφράσεις του *INSERT* και του *UPDATE*. Για παράδειγμα

```
INSERT INTO departments
  (department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);
```

```
UPDATE departments
SET manager_id = DEFAULT WHERE department_id = 10;
```

κατά την εισαγωγή, ενημέρωση της εγγραφής χρησιμοποιείται η προκαθορισμένη τιμή για την στήλη *MANAGER\_ID*.

### 2.7.6 Συναλλαγές με την ΒΔ

Η έννοια των συναλλαγών (transactions) παρέχει μεγαλύτερη ευελιξία κατά την μεταβολή των δεδομένων καθώς εγγυάται την ακεραιότητα τους σε περίπτωση οποιασδήποτε αποτυχίας (π.χ. του συστήματος, της διεργασίας του χρήστη κλπ). Όπως είπαμε στην προηγούμενη ενότητα ως συναλλαγή ορίζεται μια λογική ομάδα εντολών μεταβολής δεδομένων. Για παράδειγμα συναλλαγή αποτελεί η μεταφορά χρημάτων μεταξύ δύο λογαριασμών η οποία περιλαμβάνει αφαίρεση ποσού από τον ένα και προσθήκη ποσού στον άλλο. Σε αυτή την περίπτωση και οι δύο ενέργειες είτε θα πρέπει να επιτύχουν είτε θα πρέπει να αποτύχουν (ατομικότητα-atomicity).

Μια συναλλαγή με την ΒΔ ξεκινά όταν εκτελείτε η πρώτη DML εντολή και μπορεί να τερματιστεί είτε με την εντολή *COMMIT* είτε με την εντολή *ROLLBACK*<sup>1</sup>. Πιο συγκεκριμένα η εντολή *COMMIT* τερματίζει την συγκεκριμένη δοσοληψία κάνοντας τυχόν μεταβολές εγγραφών μόνιμες στην ΒΔ. Η εντολή *ROLLBACK* τερματίζει επίσης την δοσοληψία αναιρώντας όλες τις αλλαγές που έχουν γίνει. Μετά τον τερματισμό της συναλλαγής η αμέσως επόμενη εντολή ξεκινά μια νέα συναλλαγή.

Οι δύο παραπάνω εντολές μας προσφέρουν: ακεραιότητα δεδομένων, έλεγχο των αλλαγών πριν γίνουν μόνιμες στην ΒΔ, ομαδοποίηση ενεργειών. Για να γίνουν καλύτερα κατανοητές αυτές οι έννοιες στα παραδείγματα που ακολουθούν,

```
DELETE FROM employees
WHERE employee_id = 99999;
1 row deleted.

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 row inserted.
```

```
COMMIT;
Commit complete.
```

διαγράφεται μια εγγραφή από τον πίνακα *EMPLOYEES* και εισάγεται μια νέα γραμμή στον πίνακα *DEPARTMENTS* στην συνέχεια οι αλλαγές αυτές γίνονται μόνιμες με την χρήση του *COMMIT*. Ενώ στην αμέσως επόμενη εικόνα διαγράφονται, από λάθος, όλες οι εγγραφές του πίνακα *COPY\_EMP* αλλά η ενέργεια αυτή αναιρείται με την χρήση του *ROLLBACK*.

<sup>1</sup> Στην ουσία υπάρχουν πολλές περιπτώσεις που τερματίζουν μια συναλλαγή αλλά δεν είναι του παρόντος.

### 2.7.7 Ασκήσεις

Το τμήμα HR θέλει να δημιουργήσετε εντολές SQL για την εισαγωγή, ενημέρωση και διαγραφή των δεδομένων που θα αφορούν τους υπαλλήλους. Σαν πρότυπο θα χρησιμοποιήσετε τον πίνακα *MY\_EMPLOYEE*, πριν παραδώσετε τις εντολές στο HR. Για τον λόγο με αντιγραφή επικόλληση εκτελέστε τον παρακάτω κώδικα. Στην συνέχεια χρησιμοποιήστε την *DESCRIBE* για να δείτε την δομή του πίνακα.

REM \*\*\*\*\*

Prompt \*\*\*\*\* Creating MY\_EMPLOYEE table ....

CREATE TABLE my\_employee

(id NUMBER(4) CONSTRAINT my\_employee\_id\_nn NOT NULL,

last\_name VARCHAR2(25),

first\_name VARCHAR2(25),

userid VARCHAR2(8),

salary NUMBER(9,2));

REM \*\*\*\*\*

Prompt \*\*\*\*\* END OF SCRIPT ....

1. Δημιουργήστε μια εντολή *INSERT* για να εισάγετε την πρώτη εγγραφή στον πίνακα *MY\_EMPLOYEE* όπως αυτή δείχνετε στην εικόνα. Να μην αναφέρεται τις στήλες στην σύνταξη του *INSERT*.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

2. Δημιουργήστε μια εντολή *INSERT* για να εισάγετε την δεύτερη εγγραφή στον πίνακα *MY\_EMPLOYEE* όπως αυτή δείχνετε στην εικόνα. Αυτή τη φορά να αναφέρονται οι στήλες του πίνακα στην σύνταξη του *INSERT*.
3. Με αντίστοιχο τρόπο εισάγετε όλες τις εγγραφές και επιβεβαιώστε ότι έχουν εισαχθεί σωστά (*SELECT*).
4. Κάντε τις αλλαγές μόνιμες.
5. Αλλάξτε το όνομα του υπαλλήλου 3 σε *Drexler*.
6. Δώστε αύξηση σε όλους τους υπαλλήλους με μισθό μικρότερο από 900, 10 ευρώ. Κάντε τις αλλαγές μόνιμες.

7. Απολύστε τους υπαλλήλους με μισθό μεγαλύτερο από 1000.
8. Αλλάξατε γνώμη, αναιρέστε τις αλλαγές.

## 2.8 Δημιουργία πινάκων

### 2.8.1 Γενικά

Στην ενότητα αυτή θα δούμε την σύνταξη και την χρήση της εντολής δημιουργίας πινάκων καθώς και τις έννοιες των περιορισμών ακεραιότητας στην πράξη.

### 2.8.2 Η εντολή *CREATE TABLE*

Με την DDL εντολή *CREATE TABLE* μπορούμε να δημιουργήσουμε στην ΒΔ ένα νέο πίνακα. Η σύνταξη της έχει ως εξής:

```
CREATE TABLE [schema.] table
              (column datatype [DEFAULT expr] [, ...]);
```

όπου *schema* είναι το όνομα του χρήστη, *table* είναι το όνομα του πίνακα, *DEFAULT expr* καθορίζει την προκαθορισμένη τιμή, *column* είναι το όνομα της στήλης και *datatype* είναι ο τύπος της στήλης και το μέγεθος της.

Ένα παράδειγμα χρήσης του *DEFAULT* είδαμε στην παράγραφο 2.7.5 με την εντολή:

```
CREATE TABLE Persons
(
  P_Id int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255) DEFAULT 'Αθήνα'
)
```

Κατά το παράδειγμα αυτό δημιουργείται ο πίνακας με το όνομα *Persons* ο οποίος περιλαμβάνει πέντε στήλες, τις *P\_Id*, *LastName*, *FirstName*, *Address*, *City*. Σε κάθε μια από τις στήλες αυτές μπορεί να εισαχθεί ένας συγκεκριμένος τύπος δεδομένων<sup>2</sup> ο οποίος δηλώνεται ακριβώς δίπλα από το όνομα της στήλης και ο οποίος επιλέγεται ώστε να είναι συμβατός με το είδος της πληροφορίας που θέλουμε να αποθηκεύσουμε. Έτσι η στήλη *P\_Id* δηλώνεται σαν *int* (*integer*) δηλαδή ακέραιος, ενώ η στήλη *LastName* δηλώνεται σαν *varchar(255)* δηλαδή μεταβλητού μεγέθους χαρακτήρες με μέγιστο μέγεθος τα 255 bytes. Γενικότερα στα ΣΔΒΔ υπάρχουν υλοποιημένοι πολλοί βασικοί τύποι. Μια γενική κατηγοριοποίηση τους δείχνεται στον παρακάτω πίνακα

<sup>2</sup> Σύμφωνα με την θεωρία αναφερόμαστε στο πεδίο τιμών μιας ιδιότητας.  
Hands on 05(final)

Τύποι Δεδομένων	Δηλώσεις
λογικός	BOOLEAN
χαρακτήρας	CHAR, VARCHAR
δυαδικό ψηφίο	BIT, BIT VARYING
ακριβής αριθμητικός	NUMERIC, DECIMAL, INTEGER, SMALLINT
προσεγγιστικός αριθμητικός	FLOAT, REAL, DOUBLE PRECISION
ημερομηνία και ώρα	DATE, TIME, TIMESTAMP
διάστημα	INTERVAL

3

### 2.8.3 Περιορισμοί ακεραιότητας (*integrity constraints*)

Με τους περιορισμούς ακεραιότητας ελέγχεται η συνέπεια των δεδομένων. Παραδείγματα περιορισμών είναι η εφαρμογή συγκεκριμένων κανόνων κατά την εισαγωγή, ενημέρωση ή διαγραφή δεδομένων από ένα πίνακα ή η αποτροπή διαγραφής ενός πίνακα στην περίπτωση που υπάρχουν συσχετίσεις από άλλους πίνακες κ.α. Οι περιορισμοί μπορούν να διακριθούν σε:

*integrity constraints*: ορίζουν το πρωτεύον κλειδί και το κλειδί αναφοράς (primary and foreign keys)

*value constraints*: ορίζουν τιμές δεδομένων που επιτρέπεται να εισαχθούν στα δεδομένα και όταν οι τιμές αυτές πρέπει να είναι μοναδικές (unique) ή όχι κενό (not NULL)

*table constraints*: περιορίζουν τις τιμές που εισάγονται σε σχέση με όλες τις άλλες τιμές σε έναν πίνακα

*field constraints*: περιορίζουν τις τιμές που μπορούν να εισαχθούν σε ένα συγκεκριμένο πεδίο ενός πίνακα, ανεξάρτητα από τιμές που υπάρχουν σε άλλα πεδία

Ένας περιορισμός μπορεί να οριστεί κατά την δημιουργία του πίνακα, π.χ.

<sup>3</sup> Για περαιτέρω μελέτη των διαφορών μεταξύ των τύπων μπορείτε να δείτε εδώ <http://www.w3resource.com/sql/data-type.php> όπως και στο reference του εκάστοτε RDMS που χρησιμοποιείτε.  
Hands on 05(final)

ή εκ των υστέρων με ανάλογη σύνταξη.

```
ALTER TABLE table_name  
ADD CONSTRAINT constraint_name PRIMARY KEY (column1, column2, ...  
column_n);
```

Οι περιορισμοί μπορεί να αναφέρονται σε ολόκληρο τον πίνακα ή σε κάποια στήλη. Οι κυριότεροι περιορισμοί που είναι διαθέσιμοι στα ΣΔΒΔ είναι τα εξής:

Περιορισμός (constraint)	Περιγραφή
<b>NOT NULL</b>	Καθορίζει ότι τιμή της στήλης δεν μπορεί να είναι null
<b>UNIQUE</b>	Καθορίζει ότι η τιμή κάποιας στήλης ή συνδυασμού στηλών πρέπει να είναι μοναδική μεταξύ όλων των εγγραφών του πίνακα
<b>PRIMARY KEY</b>	Προσδιορίζει μοναδικά κάθε εγγραφή στον πίνακα
<b>FOREIGN KEY</b>	Εισάγει μια σχέση ξένου κλειδιού μεταξύ πινάκων

#### 2.8.4 Περιορισμός *NOT NULL*

Ο περιορισμός *NOT NULL* βεβαιώνει ότι μια στήλη δεν μπορεί να περιέχει *NULL* τιμές. Μπορεί να οριστεί μόνο σε επίπεδο στήλης, και στην εικόνα έχει εφαρμοστεί στα πεδία *LAST\_NAME* και *HIRE\_DATE* του πίνακα *EMPLOYEES*. Κατά την δημιουργία του

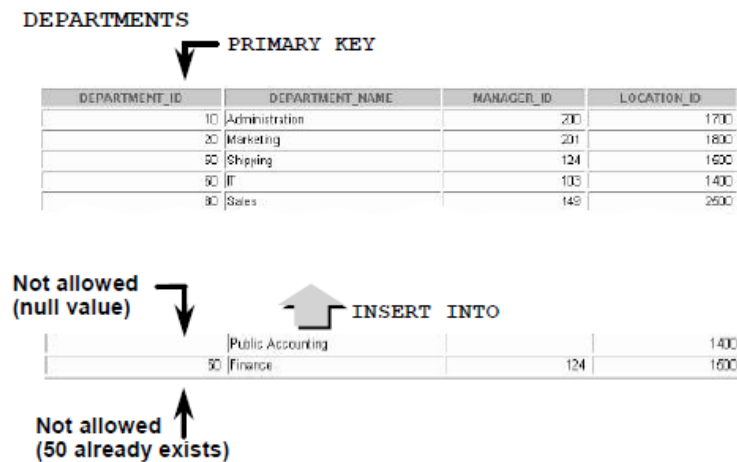
περιορισμού μπορούμε να ορίσουμε το όνομα που θέλουμε να έχει (εδώ *EMP\_LAST\_NAME\_NN*):

### 2.8.5 Περιορισμός *UNIQUE*

Ο περιορισμός *UNIQUE* βεβαιώνει ότι η τιμή σε μια στήλη ή σε συνδυασμό στηλών πρέπει να είναι μοναδική, δηλαδή δεν μπορούν δύο γραμμές του πίνακα να έχουν ίδιες τιμές στο/α συγκεκριμένο/α πεδίο/α. Μια ιδιαιτερότητα του *UNIQUE* είναι ότι επιτρέπει την ύπαρξη *NULL* τιμών καθώς τα *nulls* δεν θεωρείται ότι είναι ίσα με το οτιδήποτε. Στην εικόνα έχει εφαρμοστεί στο πεδίο *EMAIL* του πίνακα *EMPLOYEES* με όνομα *EMP\_EMAIL\_UK*.

```
CREATE TABLE employees(  
    employee_id      NUMBER(6) ,  
    last_name        VARCHAR2(25) NOT NULL ,  
    email            VARCHAR2(25) ,  
    salary           NUMBER(8,2) ,  
    commission_pct   NUMBER(2,2) ,  
    hire_date        DATE NOT NULL ,  
    ...  
    CONSTRAINT emp_email_uk UNIQUE(email));
```

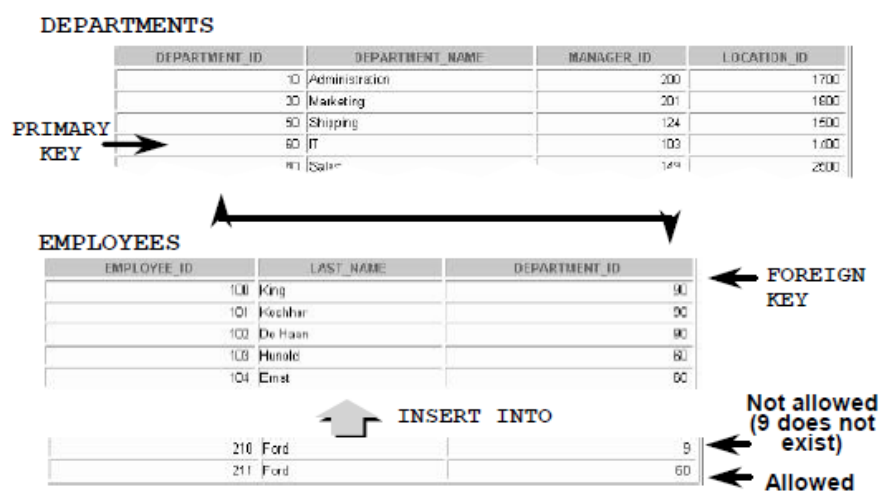
## 2.8.6 Περιορισμός PRIMARY KEY (πρωτεύοντος κλειδιού)



Ο περιορισμός αυτός δημιουργεί ένα πρωτεύον κλειδί στον πίνακα και μπορεί να περιλαμβάνει μια ή περισσότερες στήλες. Κάθε πίνακας μπορεί να έχει μόνο ένα πρωτεύον κλειδί το οποίο βεβαιώνει ότι η τιμή στη/ις στήλη/ες πρέπει να είναι μοναδική αλλά και να μην περιέχει/ουν *null* τιμές. Στην εικόνα έχει δημιουργηθεί το *DEPT\_ID\_PK* πρωτεύον κλειδί στο πεδίο *DEPARTMENT\_ID* του πίνακα *EMPLOYEES*.

```
CREATE TABLE departments (
  department_id      NUMBER(4),
  department_name     VARCHAR2(30)
  CONSTRAINT dept_name_nn NOT NULL,
  manager_id         NUMBER(6),
  location_id         NUMBER(4),
  CONSTRAINT dept_id_pk PRIMARY KEY(department_id));
```

## 2.8.7 Περιορισμός FOREIGN KEY (ξένου κλειδιού)





Διαφορετικοί πίνακες μιας σχεσιακής ΒΔ μπορούν να συσχετισθούν μέσω κοινών στηλών. Οι κανόνες “αναφορικής ακεραιότητας” είναι αυτοί που κανονίζουν και εξασφαλίζουν τη λειτουργία αυτών των συσχετίσεων. Ένας περιορισμός αναφορικής ακεραιότητας απαιτεί για κάθε γραμμή ενός πίνακα, η τιμή στο ξένο κλειδί να ταιριάζει με μια τιμή του γονικού κλειδιού. Σαν *ξένο κλειδί (foreign key)* ορίζεται η στήλη ή ομάδα στηλών που περιλαμβάνεται στον ορισμό του περιορισμού αναφορικής ακεραιότητας που αναφέρεται σε ένα *κλειδί αναφοράς (referenced key)*. Το κλειδί αναφοράς είναι το πρωτεύον κλειδί ή μια *unique* στήλη του πίνακα που αναφέρεται το ξένο κλειδί. Άλλοι ορισμοί που εισάγονται είναι αυτός του *dependent or child table* που είναι ο πίνακας που περιλαμβάνει το ξένο κλειδί δηλαδή, ο πίνακας που είναι εξαρτώμενος από τις τιμές που υπάρχουν στο αναφερόμενο κλειδί καθώς και ο *referenced or parent table* που είναι ο πίνακας στον οποίον αναφέρεται το ξένο κλειδί του πίνακα-παιδιού. Με βάση αυτού του πίνακα αποφασίζεται κατά πόσον επιτρέπονται ειδικές εισαγωγές ή ενημερώσεις στον πίνακα-παιδί. Μια ιδιαιτερότητα είναι ότι στο ξένο κλειδί επιτρέπεται να υπάρχουν τιμές *null*, ακόμη και αν δεν υπάρχουν στην αναφορά του. Στην εικόνα έχει δημιουργηθεί το *EMP\_DEPT\_FK* ξένο κλειδί στο πεδίο *DEPARTMENT\_ID* του πίνακα *EMPLOYEES* το οποίο αναφέρεται στον πεδίο *DEPARTMENT\_ID* του πίνακα *DEPARTMENTS*.

```
CREATE TABLE employees(
  employee_id      NUMBER(6) ,
  last_name        VARCHAR2(25) NOT NULL,
  email            VARCHAR2(25) ,
  salary           NUMBER(8,2) ,
  commission_pct   NUMBER(2,2) ,
  hire_date        DATE NOT NULL,
  ...
  department_id    NUMBER(4) ,
  CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
    REFERENCES departments(department_id) ,
  CONSTRAINT emp_email_uk UNIQUE(email));
```

### 2.8.8 Ασκήσεις

1. Ορίστε το πρωτεύον κλειδί *cp\_emp\_id\_pk* του πίνακα *COPY\_EMP* το πεδίο *ID*.
2. Δημιουργήστε τον πίνακα *DEPT* δίνοντας

*create table dept as select \* from departments;*

Στην συνέχεια ορίστε ως πρωτεύον κλειδί *dpt\_dptid\_pk* το πεδίο *DEPARTMENT\_ID* και προσθέστε ένα *unique constraint unique\_dpt\_nm* στο πεδίο *DEPARTMENT\_NAME*.

3. Προσθέστε ένα *foreign key* στον πίνακα *COPY\_EMP* από το πεδίο *DEPARTMENT\_ID* το οποίο θα ελέγχει ότι κάποιος υπάλληλος δεν μπορεί να ανατεθεί σε μη υπάρχον τμήμα του πίνακα *DEPT*. Όνομα *constraint cp\_emp\_dept\_id\_fk*.
4. Εισάγετε έναν νέο υπάλληλο ο οποίος ανήκει στο τμήμα 300; Τι παρατηρείτε και γιατί;
5. Δημιουργήστε ένα νέο πίνακα *COPY\_EMP2* σύμφωνα με τις παρακάτω απαιτήσεις:

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK Table				
FK Column				
Data type	NUMBER	VARCHAR2	VARCHAR2	NUMBER
Length	7	25	25	7

Name	Null?	Type
ID		NUMBER(7)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)