საქართველოს ტექნიკური უნივერსიტეტი

ბადრი მეფარიშვილი, გულნარა ჯანელიძე

განაწილებული მონაცემთა ბაზების მართვის სისტემა Oracle

(ლაბორატორიული პრაქტიკუმი)

რეკომენდებულია სტუ-ის
სარედაქციო-
საგამომცემლო საბჭოს
მიერ. 28.10.2015, ოქმი №2

თბილისი
2016

განხილულია ლაბორატორიული სამუშაოების შესრულების მეთოდიკა მონაცემთა ბაზების მართვის სისტემა Oracle 10g XE გარემოში.

განკუთვნილია ინფორმატიკის დარგის სტუდენტებისათვის.

რეცენზენტები:

პროფესორი გია სურგულაძე

პროფესორი თინათინ კაიშაური

ISBN 978-9941-20-615-3

http//www.gtu.ge

ლაბორატორიული სამუშაოები	
N	თემის დასახელება და შინაარსი
1	Oracle 10g: მოკლე მიმოხილვა
2	მონაცემთა განსაზღვრების ენა
3	მონაცემთა მანიპულირების ენა
4	მუშაობა Object Browser -ში
5	მუშაობა Query Builder -ში
6	დანართის შემუშავება Oracle-ის გარემოში
7	*PL/SQL* ენის საფუძვლები
8	მუშაობა PL/SQL Developer-ში
9	კურსორები. პროცედურები და ფუნქციები.
10	ტრიგერები. პაკეტები. ტრანზაკციების მართვა
11	ფორმები
12	ანგარიშები
13	პროექტები
14	Oracle 10g XE ადმინისტრირება
15	მონაცემთა ბაზების უსაფრთხოება

შინაარსი

ზოგადი მითითებები

ლაბორატორიული სამუშაოების შესრულების თანამიმდევრობა

ლაბორატორიული ციკლის მიზანია მონაცემთა ბაზების აგება და საინფორმაციო სისტემის აპლიკაციის შემუშავება Oracle 10 g XE გარემოში SQL*Plus და PL/SQL ენების, აგრეთვე PL/SQL Developer -ის გამოყენებით.

ლაბორატორიული სამუშაოების ციკლის დაწყებამდე, საგნის სილაბუსის საშუალებით, სტუდენტი ეცნობა სამუშაოთა ჩამონათვალს და ძირითად მიზანს. ყოველ სამუშაო ადგილზე იქმნება მოცემული ჯგუფის საქაღალდე და მასში სტუდენტის ინდივიდუალური საქაღალდე, სადაც განთავსდება შესრულებული ლაბორატორიული სამუშაოების ოქმების ფაილები (doc გაფართოებით) ანუ ელექტრონული ოქმები, რომლებიც სემესტრის ბოლოს E-mail-ით გადაეგზავნება ლაბორატორიული მეცადინეობების მასწავლებელს და შემდგომი დაარქივებისათვის გადააქვთ CD-ზე.

წინასწარი მომზადება

ლაბორატორიული სამუშაოს აღწერა შედგება შემდეგი ნაწილებისაგან:

- სამუშაოს მიზანი;
- დავალება;
- დავალების შესრულების თანამიმდევრობა.

შესასრულებელი დავალება

ყოველი ლაბორატორიული სამუშაოს შესასრულებელი დავალება ფორმულირებულია „დავალება“ ნაწილში. სტუდენტი წინასწარ, დამოუკიდებლად უნდა გაეცნოს დავალებას და თუ გაუჩნდება შეკითხვები, ისინი უნდა დასვას სამუშაოს დაწყებამდე.

დავალების შესრულება

დავალების შესრულებამდე მოწმდება სტუდენტის მიერ დავალების წინასწარი მომზადების ხარისხი და, შესაბამისად, გადაწყდება დავალების შესასრულებლად სტუდენტის კომპიუტერთან დაშვების საკითხი.

ლაბორატორიული სამუშაო სრულდება აღწერაში მოცემული პრაქტიკული მოქმედებების თანამიმდევრობის საფუძველზე. მიღებული შედეგების ეკრანზე წარმოდგენის შემდეგ საჭიროა ტექსტური ფაილების სახით მათი ასლების შექმნა და შენახვა სტუდენტის ინდივიდუალურ საქაღალდეში, თარიღის მითითებით.

ლაბორატორიული სამუშაოს ანგარიში

ლაბორატორიული სამუშაოს ელექტრონული ანგარიში უნდა შეიცავდეს შემდეგ პუნქტებს:

- ლაბორატორიული სამუშაოს თემა;
- შესასრულებელი ინდივიდუალური დავალება;
- ტექსტურ ფაილში გადატანილი მიღებული შედეგების ასლები;
- დასკვნა.

ლაბორატორიული სამუშაო 1

Oracle 10g: მოკლე მიმოხილვა

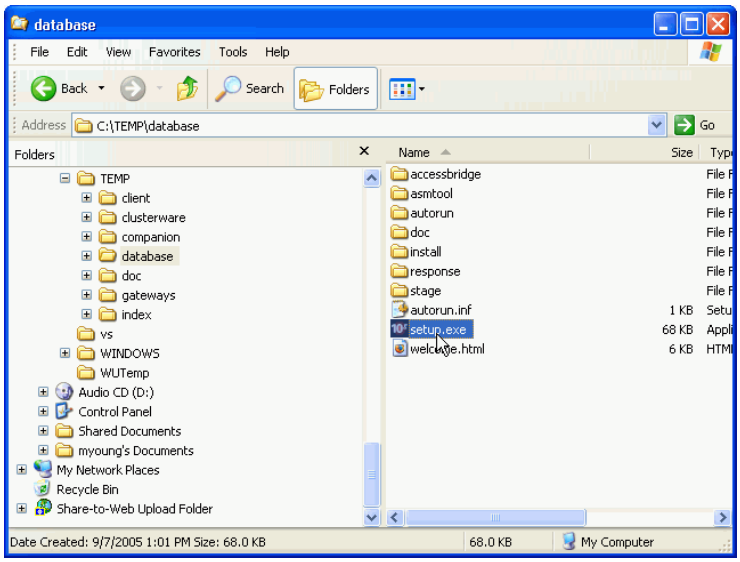
სამუშაოს მიზანი:

1. Oracle 10g-ს ინსტალაციის შესწავლა;
2. Oracle Database Express Edition (Oracle Database XE)-ის მიმოხილვა;
3. Oracle Database XE-ს Web - ინტერფეისის გაცნობა.

Oracle 10*g*-ს ინსტალაცია

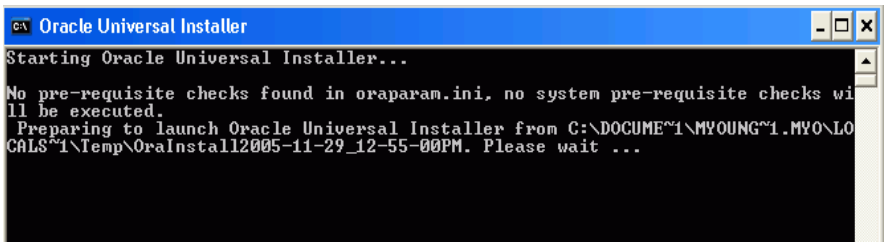
Oracle software-ის ინსტალაციისათვის, საჭიროა Oracle Universal Installer -ის გამოყენება.

1. ინსტალაციისათვის საჭიროა DVD-დან ფაილების განარქივება დირექტორიაში და **setup.exe** ფაილზე ორჯერ დაწკაპუნება.



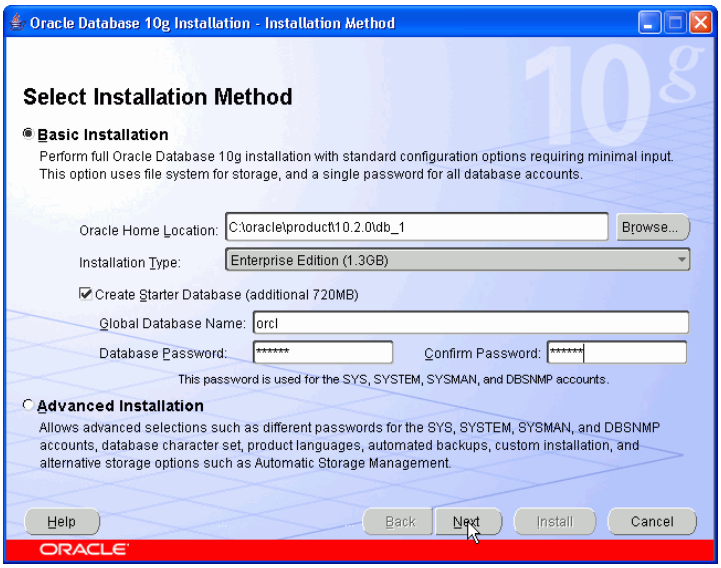
ნახ.1.1

2. Oracle Universal Installer-ის გაშვება.



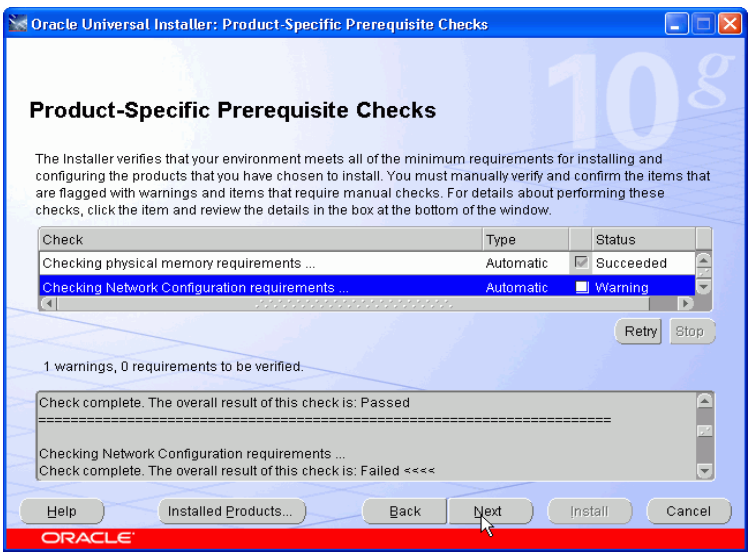
ნახ.1.2

3. დავადასტუროთ საბაზისო ინსტალაცია (basic installation). Global Database Name ველში შევიტანოთ **orcl,** ხოლო Database Password ველში — **oracle** და დავადასტუროთ Confirm Password. შემდეგ ვაწკაპუნებთ **Next**-ზე.



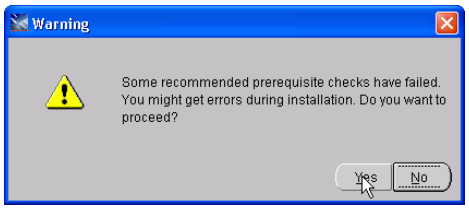
ნახ.1.3

4. თუ ინსტალაცია და კონფიგურაციის პროდუქტები შეიცავს გარკვეულ შეცდომებს, გაგრძელებამდე საჭიროა მათი კორექცია. შემდეგ ვაწკაპუნებთ **Next**.



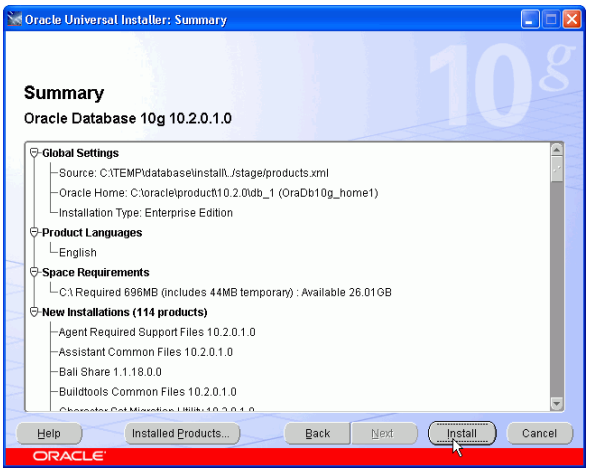
ნახ.1.4

5. თუ ვიღებთ რაიმე გაფრთხილებას, შეგვიძლია გავაგრძელოთ. ვაწკაპუნებთ **Yes**.



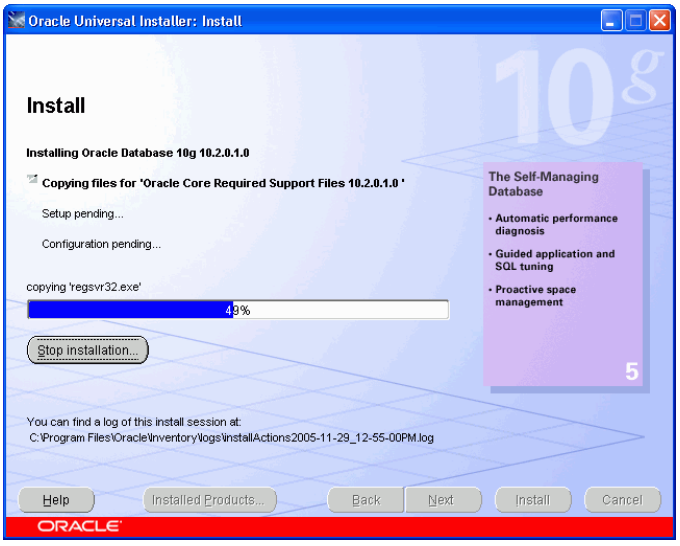
ნახ.1.5

6. Summary ფანჯარაში ვამოწმებთ თუ რა უნდა დაინსტალირდეს. შემდეგ ვაწკაპუნებთ **Install**.



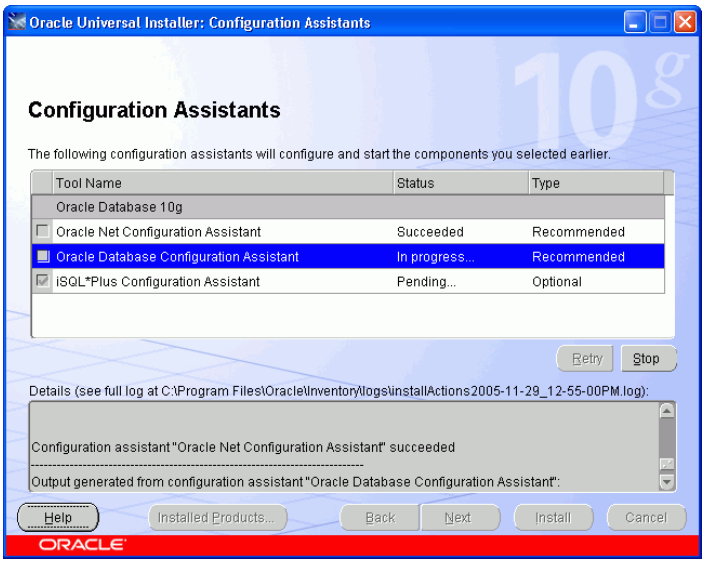
ნახ.1.6

7. ჩნდება პროცესის მსვლელობის ფანჯარა.



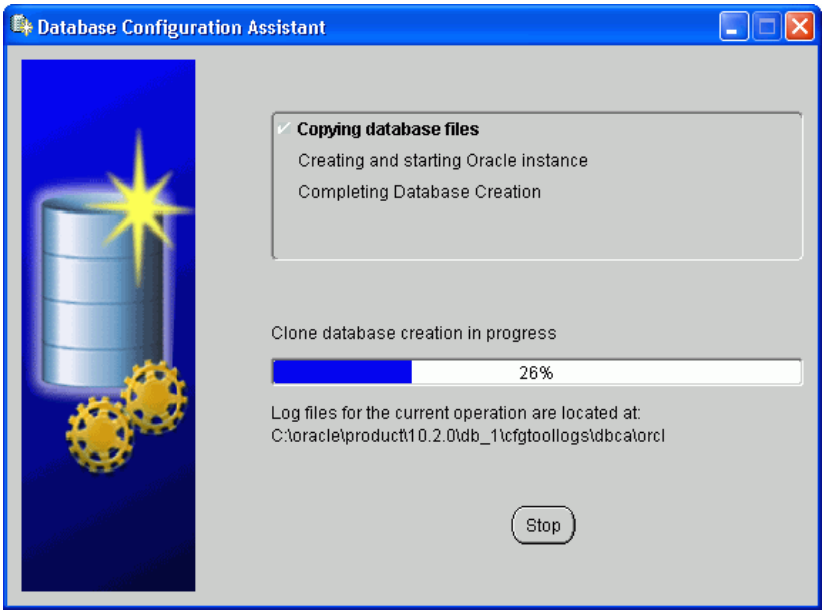
ნახ.1.7

8. ჩნდება Configuration Assistants ფანჯარა.



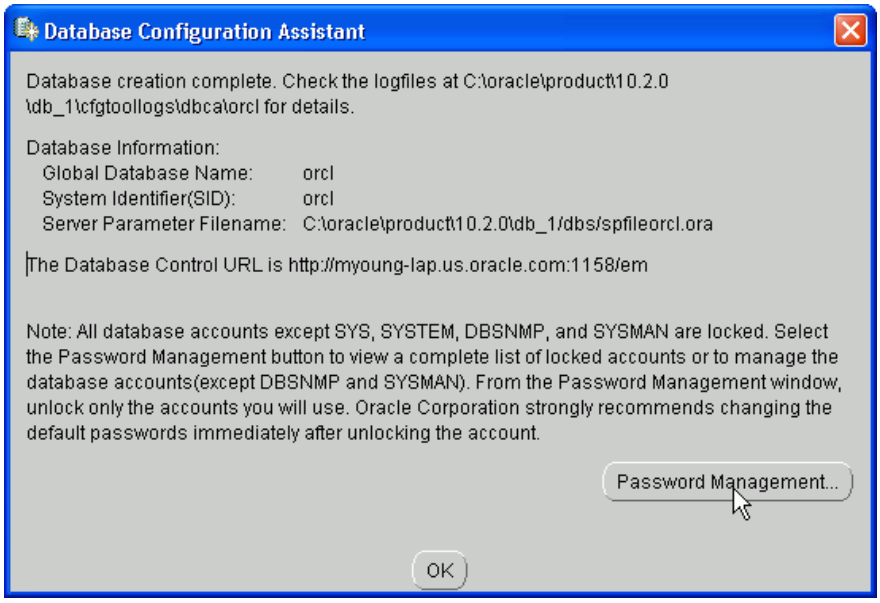
ნახ.1.8

9. ჩვენი მონაცემთა ბაზის სისტემა იქმნება.



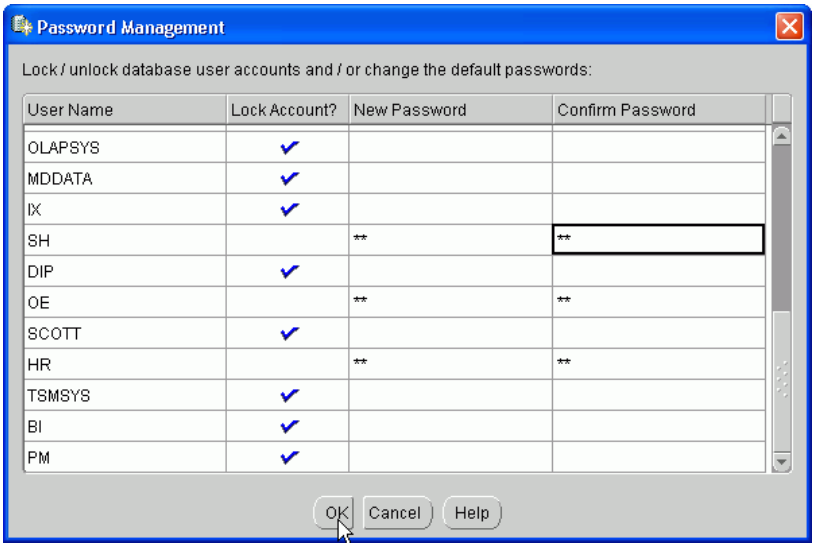
ნახ.1.9

10. შემდეგ ჩვენთვის სასურველი მომხმარებლების დებლოკირებისათვის ვაწკაპუნებთ **Password Management**.



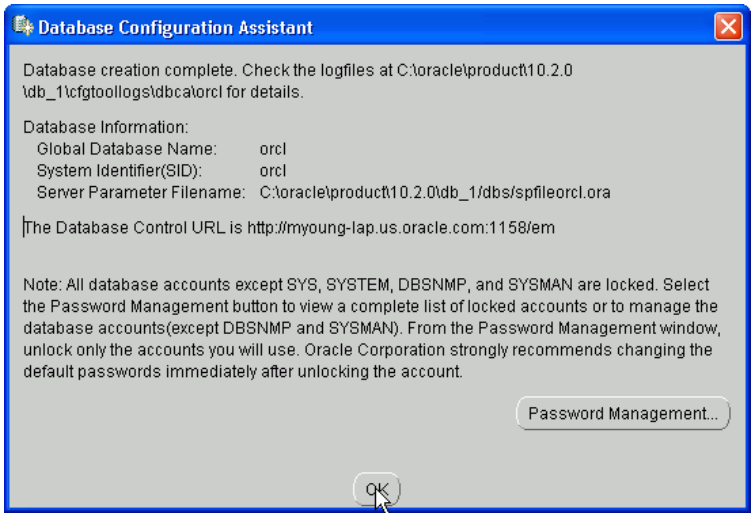
ნახ.1.10

11. **SH**, **OE** და **HR** მომხმარებელთა განბლოკირების შემდეგ დაწკაპუნებით მოვხსნათ მონიშვნა Lock Account სვეტში. შეგვაქვს მომხმარებლის იგივე სახელი New Password და Confirm Password ველებში. მაგალითად, **SH** მომხმარებლის განბლოკირებისათვის **SH** ველებში შეგვაქვს New Password და Confirm Password. შემდეგ, ვაწკაპუნებთ **OK.**



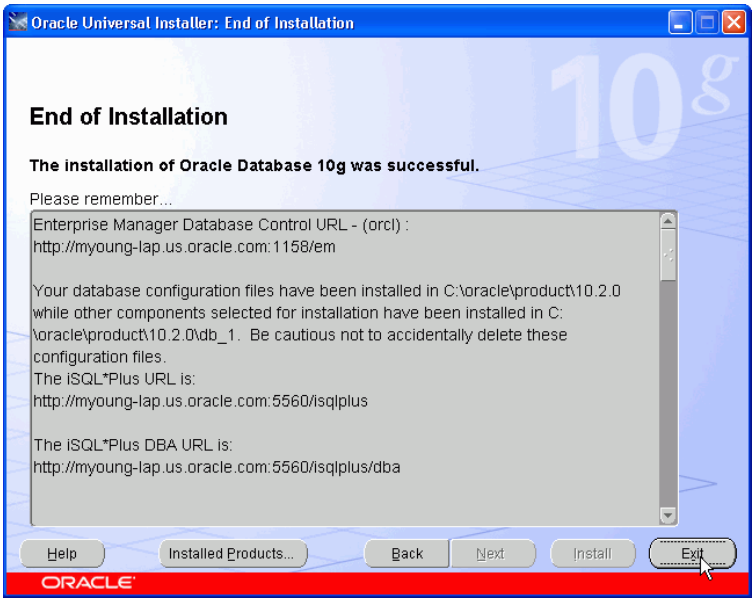
ნახ.1.11

12. კვლავ ვაწკაპუნებთ **OK**.



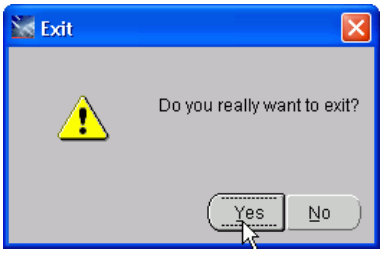
ნახ.1.12

13. ვაწკაპუნებთ **Exit**.



ნახ.1.13

14. ვაწკაპუნებთ **Yes** გამოსვლის დასტურისათვის.

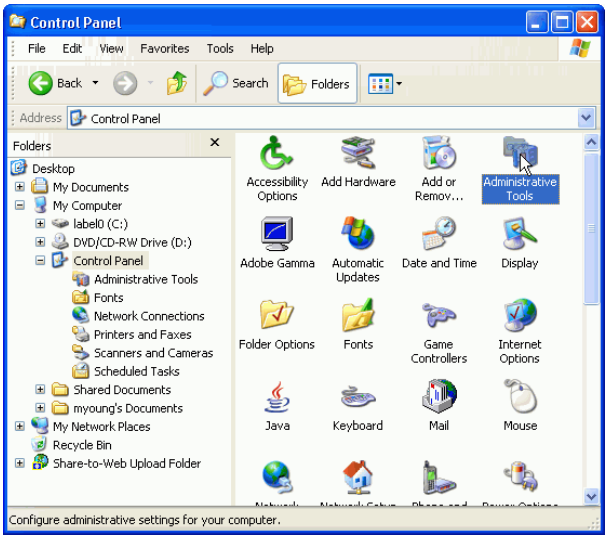


ნახ.1.14

ინსტალაციის შემდგომი ამოცანები

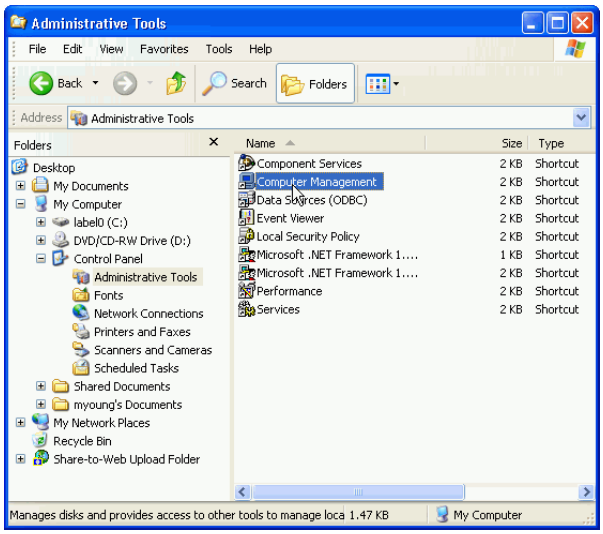
DB Control-ის დამატებითი ფუნქციონირებისათვის, როგორიცაა მონაცემთა ბაზის გაშვება და შეჩერება, ოპერაციული სისტემა უნდა მიუერთდეს ORA_DBA ჯგუფს ანუ უნდა გახდეს მისი ნაწილი, რისთვისაც უნდა შესრულდეს შემდეგი ნაბიჯები:

1. ვაწკაპუნებთ **Start** > **Settings** > **Control Panel**. ვირჩევთ **Administrative Tools**.



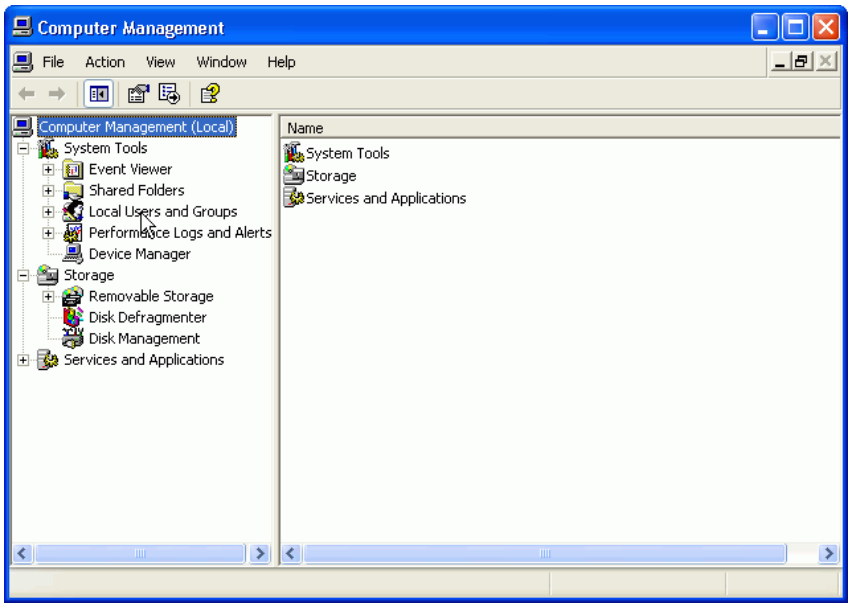
ნახ.1.15

2. ვირჩევთ **Computer Management**.



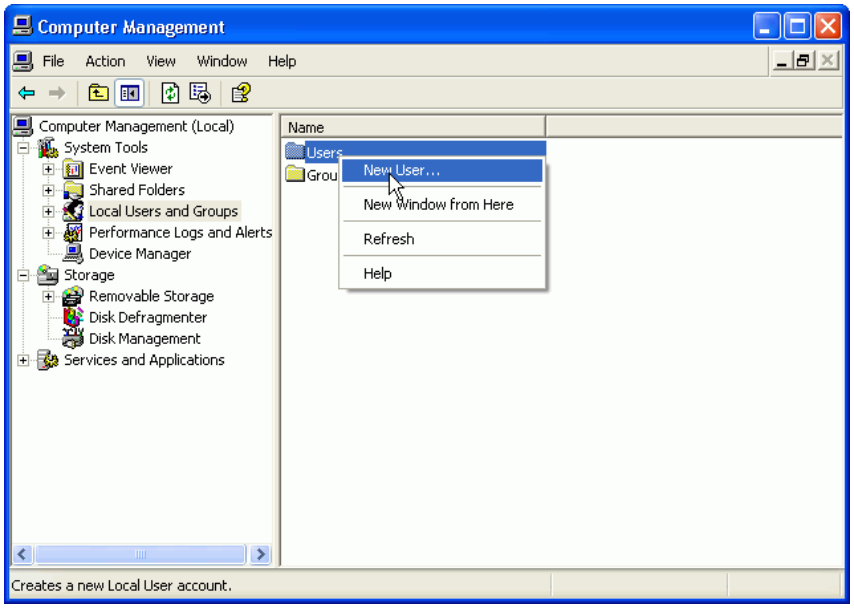
ნახ.1.16

3. ვირჩევთ **Local Users და Groups**.



ნახ.1.17

4. მარჯვენა პანელზე right ვაწკაპუნებთ **Users**. ვირჩევთ **New User ...**



ნახ.1.18

5. ფანჯარაში New User შეგვაქვს:

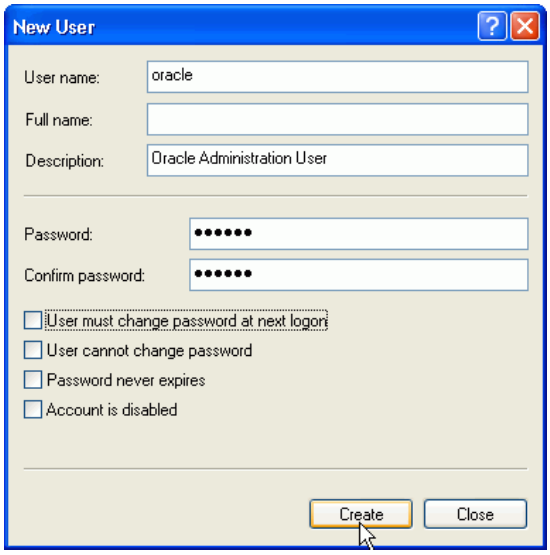
User name: **oracle**

Description: **Oracle Administration User**

Password: <a secure password>

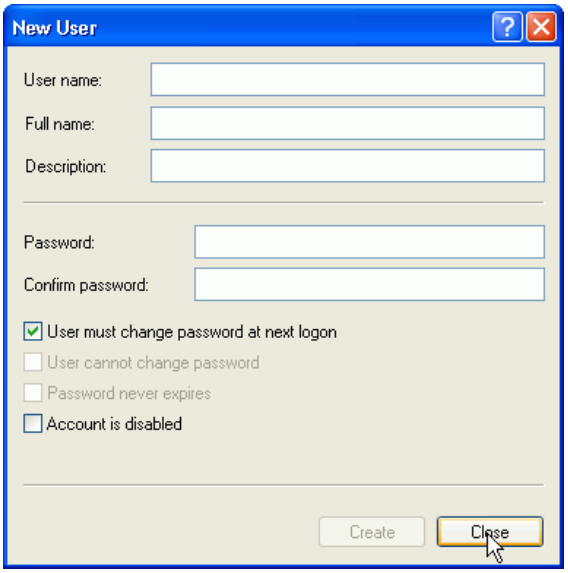
Confirm: <a secure password>

ამ მაგალითისათვის password სახით **oracle** არის გამოყენებული. არჩევანის გაუქმებისათვის (მოხსნისათვის) ვნიშნავთ **User must change password at next logon** და ვაწკაპუნებთ **Create**.



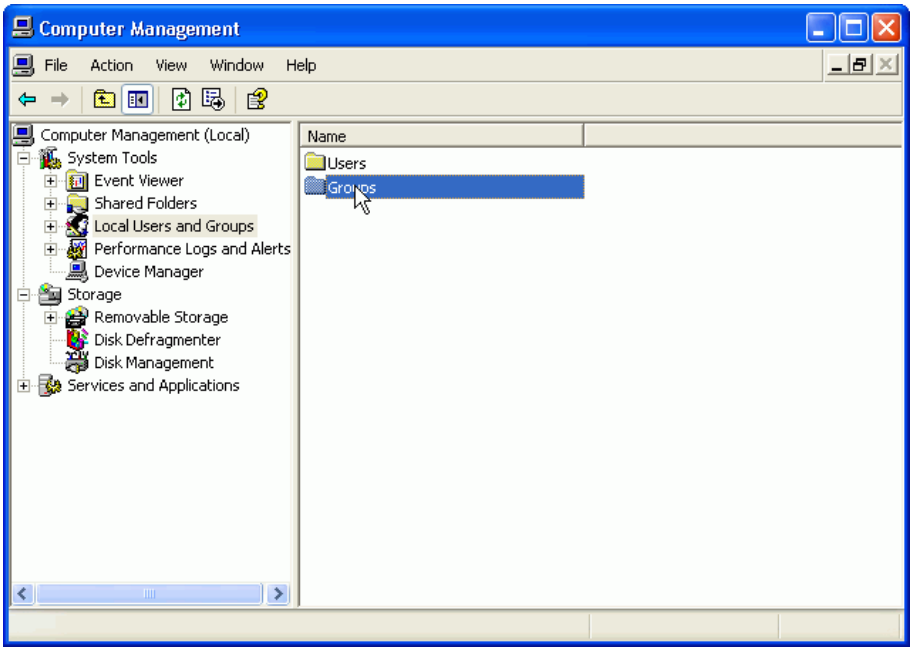
ნახ.1.19

6. ჩნდება სხვა New User ფანჯარა. დახურვისათვის ვაწკაპუნებთ **Close**.



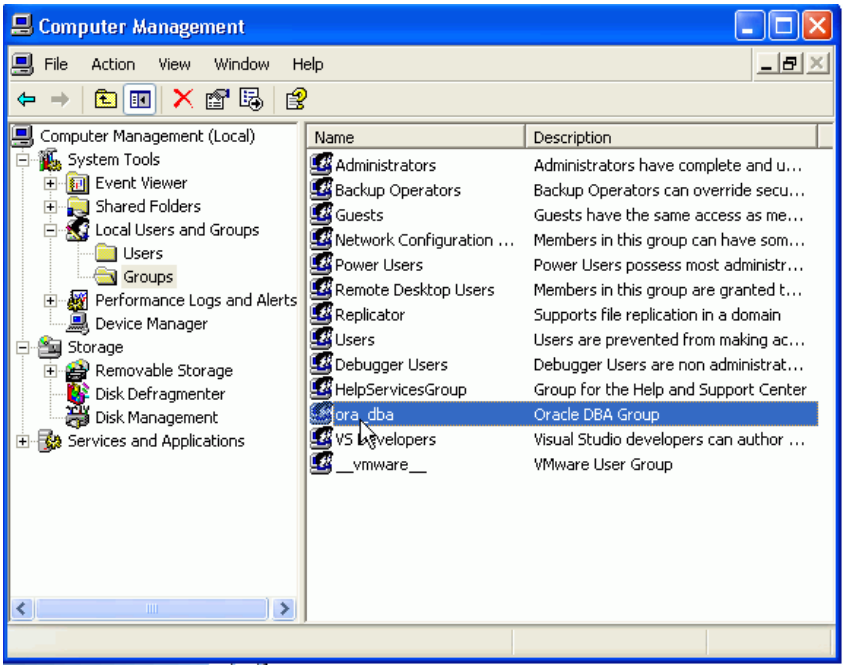
ნახ.1.20

7. ორჯერ ვაწკაპუნებთ **Groups**.



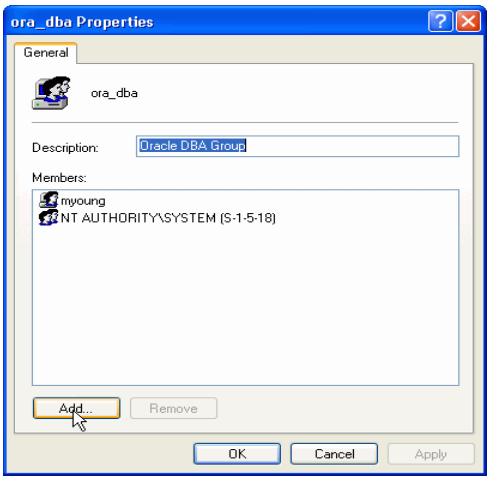
ნახ.1.21

8. ორჯერ ვაწკაპუნებთ **ora_dba**.



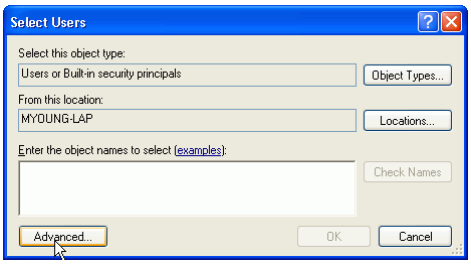
ნახ.1.22

9. ora_dba Properties ეკრანზე ვაწკაპუნებთ **Add**.



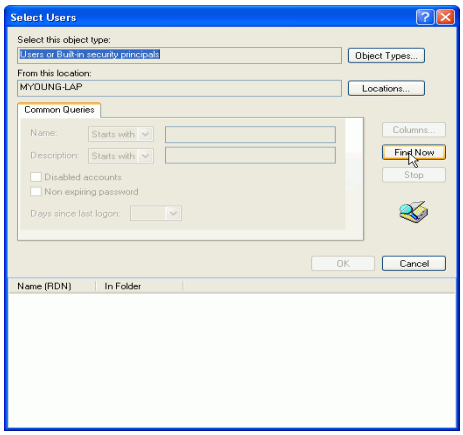
ნახ.1.23

10. Select Users ეკრანზე ვაწკაპუნებთ **Advanced**.



ნახ.1.24

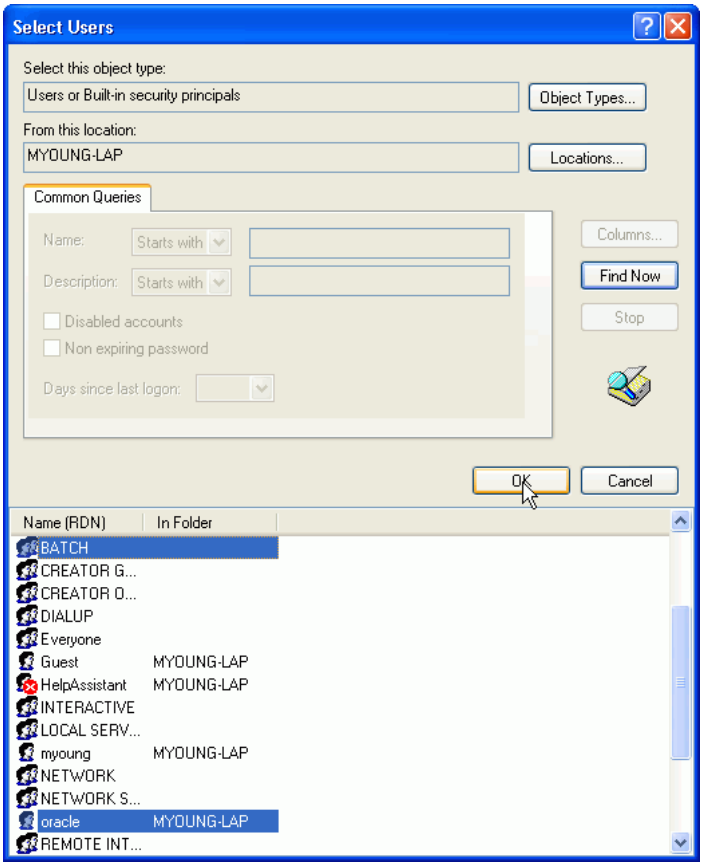
11. ვაწკაპუნებთ **Find Now**.



ნახ.1.25

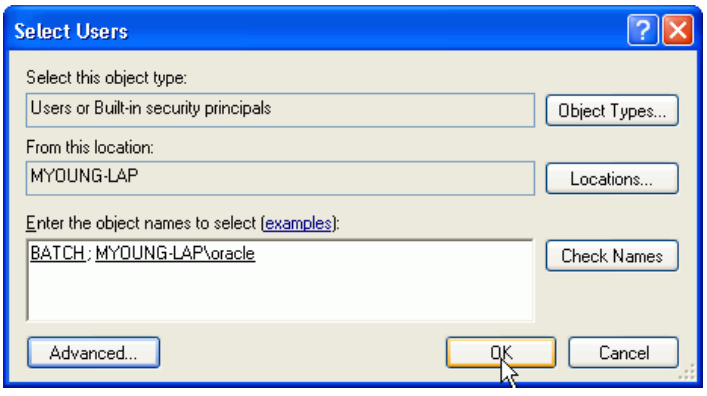
12. ფანჯრის ქვედა ნაწილში სიიდან Shift-ით ვირჩევთ **BATCH** -ს და

oracle მომხმარებელს. შემდეგ ვაწკაპუნებთ **OK**.



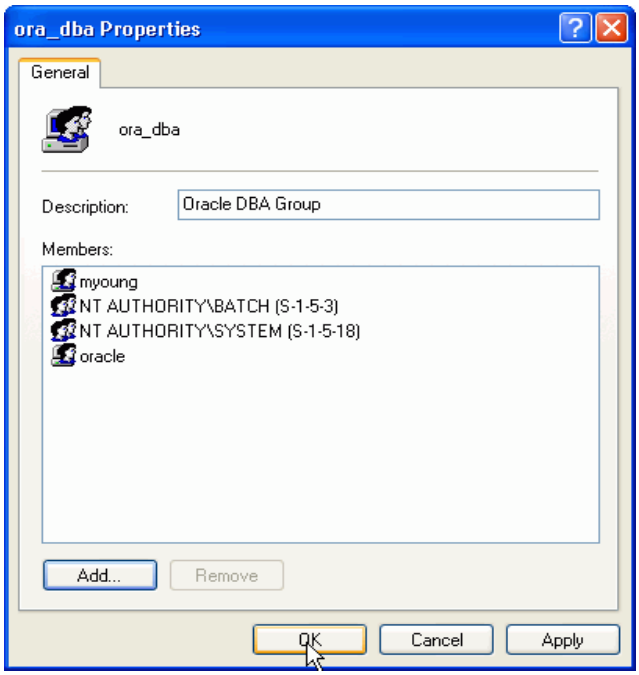
ნახ.1.26

13. ვაწკაპუნებთ **OK**.



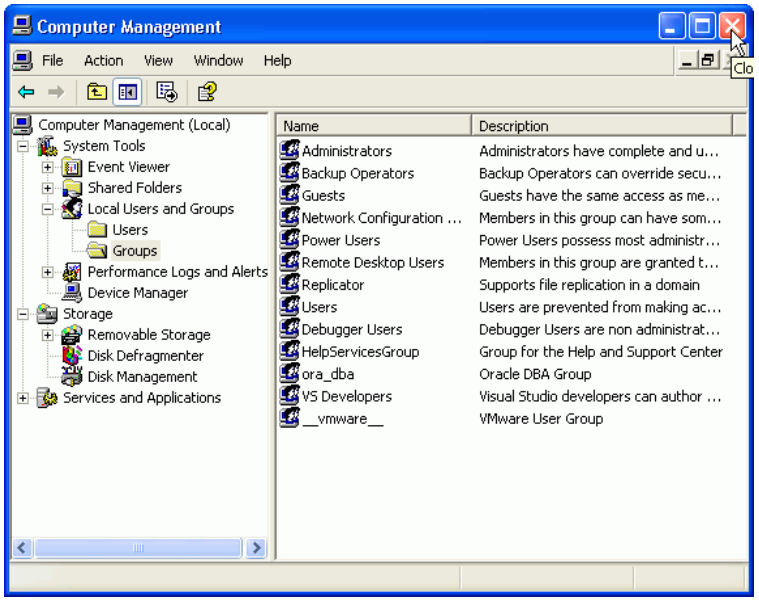
ნახ.1.27

14. ვაწკაპუნებთ **OK**.



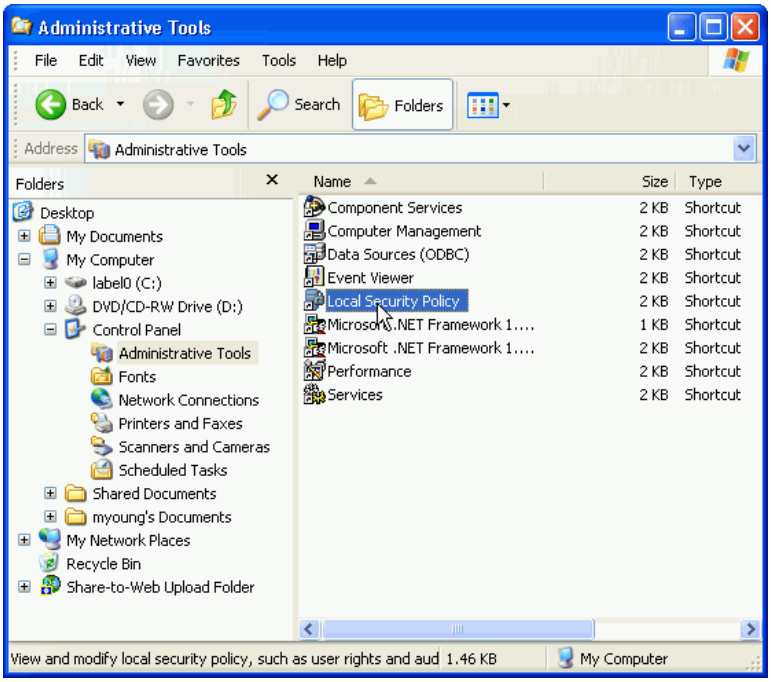
ნახ.1.28

15. დავხუროთ ფანჯარა Computer Management.



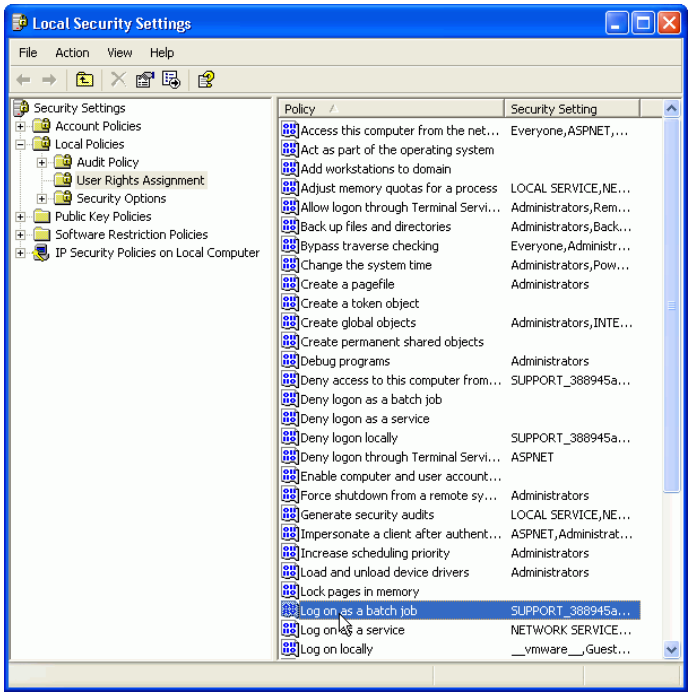
ნახ.1.29

16. თუ საჭიროა Security Policy Log-ის დავალებათა პაკეტის დამატება, ორჯერ ვაწკაპუნებთ **Local Security Policy**-ზე.



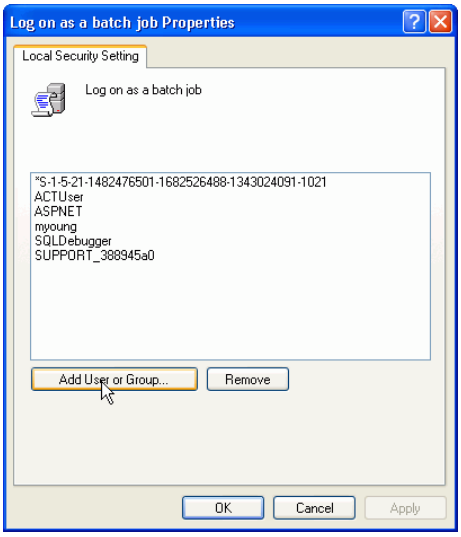
ნახ.1.30

17. ორჯერ ვაწკაპუნებთ **Log on as a batch job**-ზე.



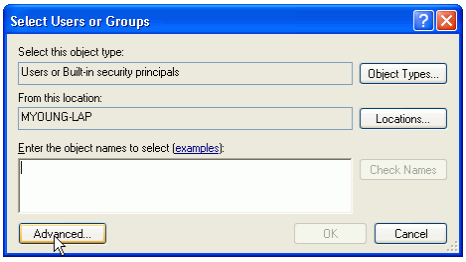
ნახ.1.31

18. ვაწკაპუნებთ **Add User or Group ...**



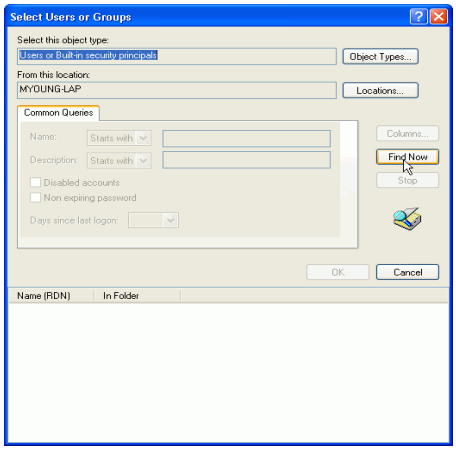
ნახ.1.32

19. ვაწკაპუნებთ **Advanced...**



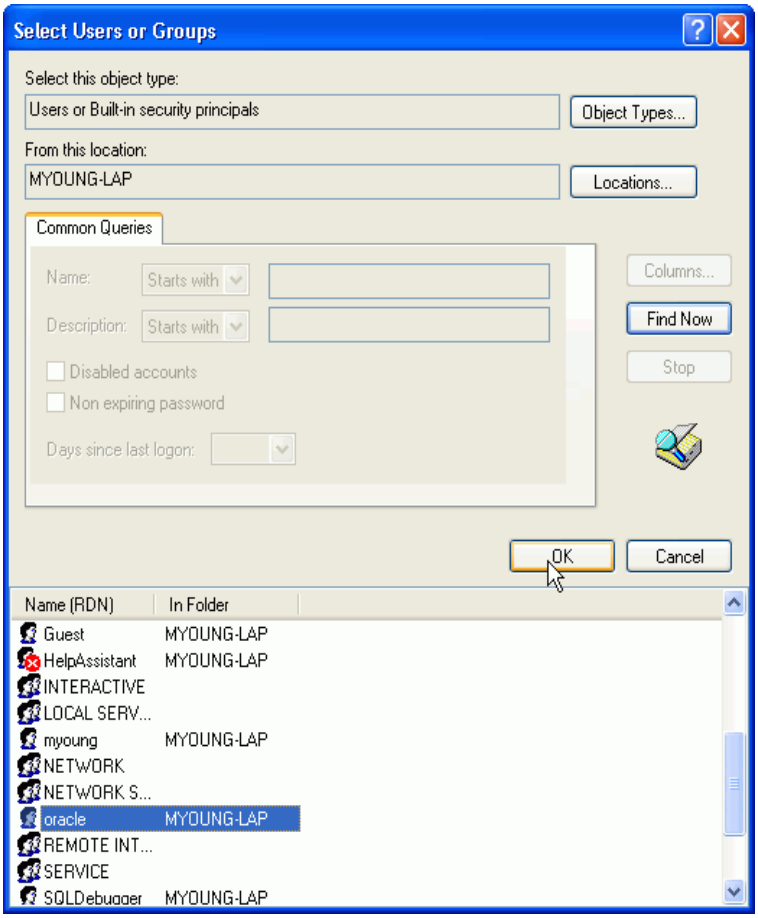
ნახ.1.33

20. ვაწკაპუნებთ **Find Now**.



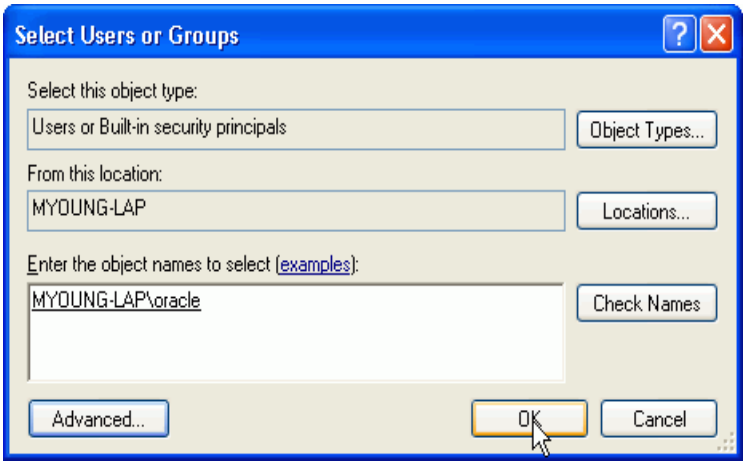
ნახ.1.34

21. სიიდან ვირჩევთ **oracle** და ვაწკაპუნებთ **OK**.



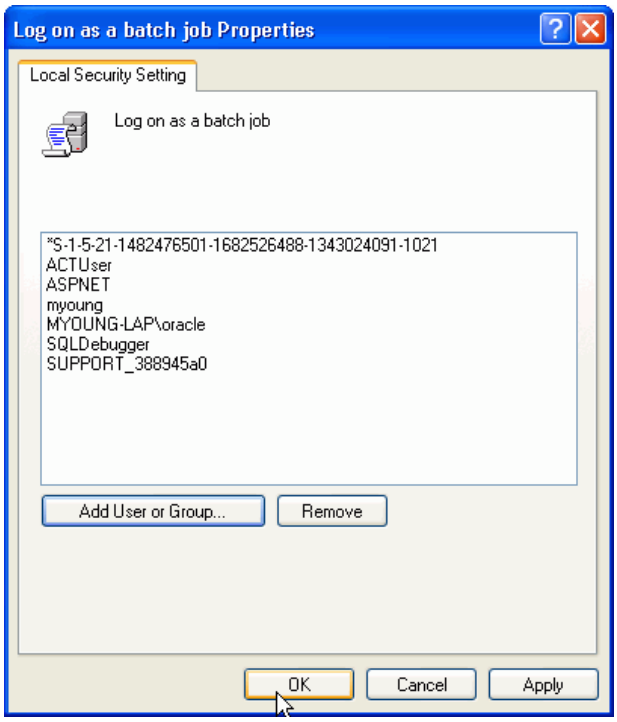
ნახ.1.35

22. ვაწკაპუნებთ **OK**.



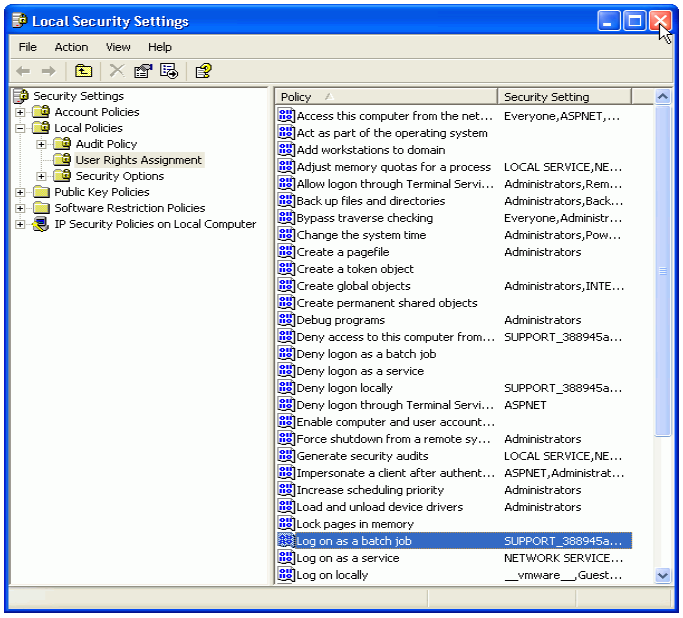
ნახ.1.36

23. კვლავ ვაწკაპუნებთ **OK**.



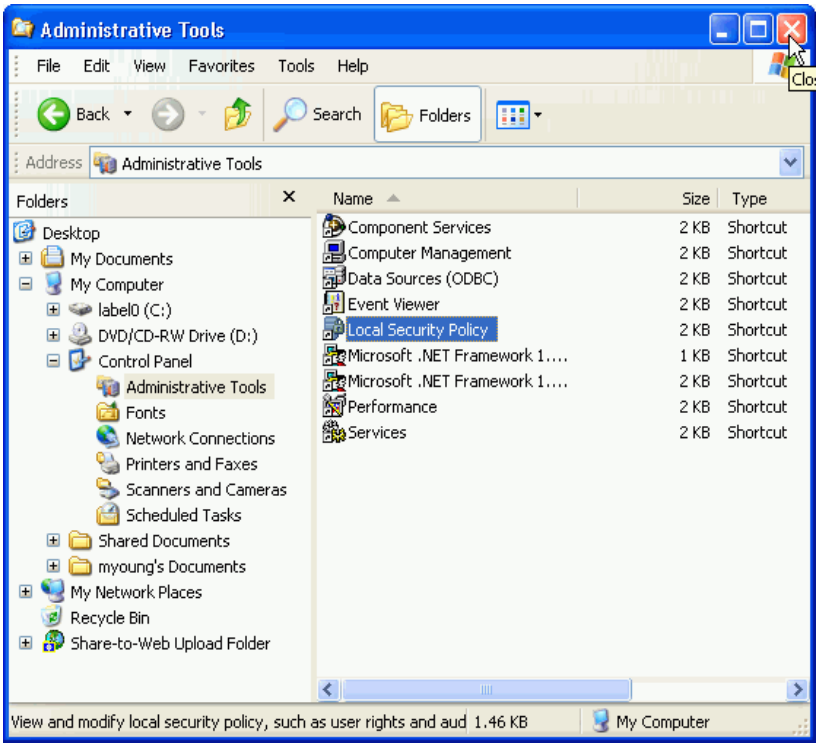
ნახ.1.37

24. დავხუროთ ფანჯარა Local Security Policy.



ნახ.1.38

25. დავხუროთ ფანჯარა Administrative Tools.

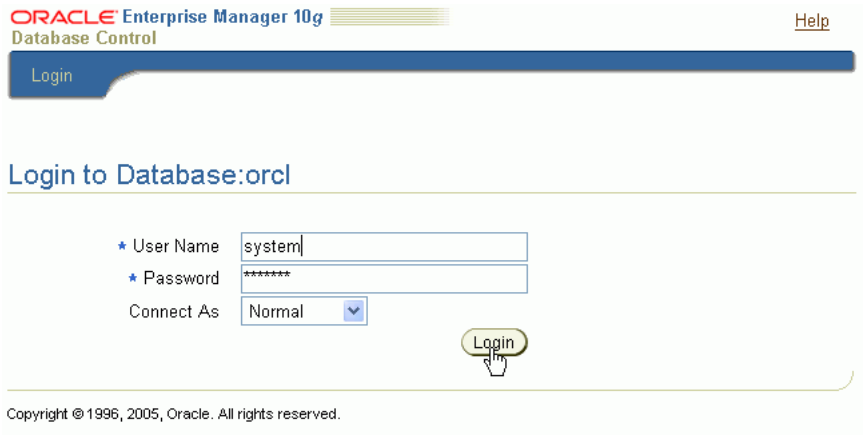


ნახ.1.39

ინსტალაციის ტესტირება

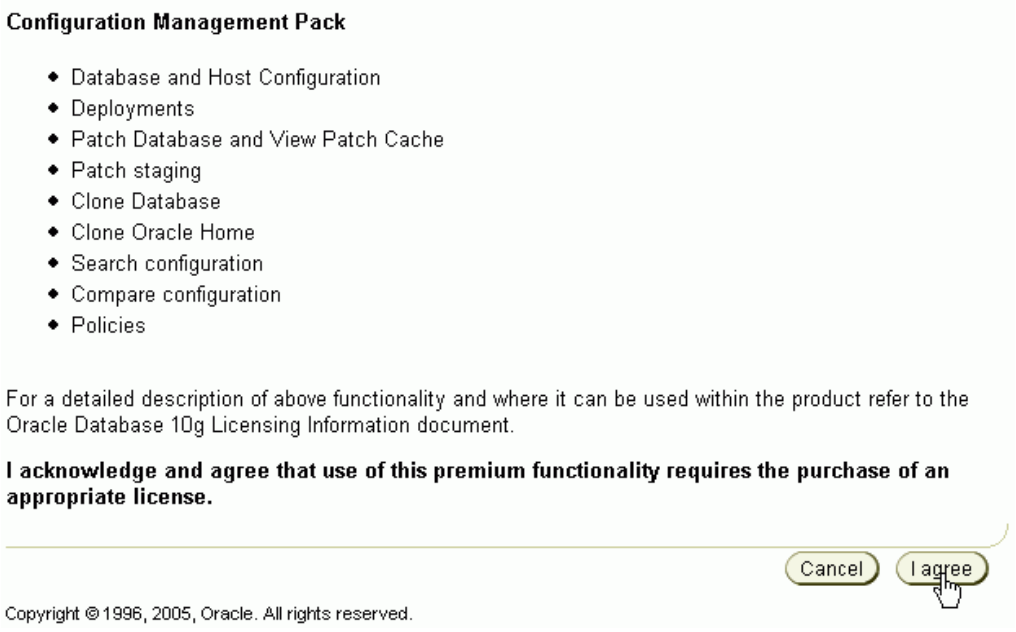
სრული და წარმატებით ინსტალაციის შემოწმებისათვის უნდა შესრულდეს შემდეგი:

1. წინა სექციიდან გამოსვლისათვის დაწკაპუნებით ბრაუზერი გახსნის Database Control URL-ს, რომელიც უკვე გაშვებულია. თუ ეს არ მოხდა, მაშინ გახსენით ბრაუზერი და შეიტანეთ შემდეგი URL:
   http://<hostname>:1158/em
   შეიტანეთ **system** როგორც User Name და **oracle** როგორც Password-ი და ვაწკაპუნებთ **Login**-ზე.



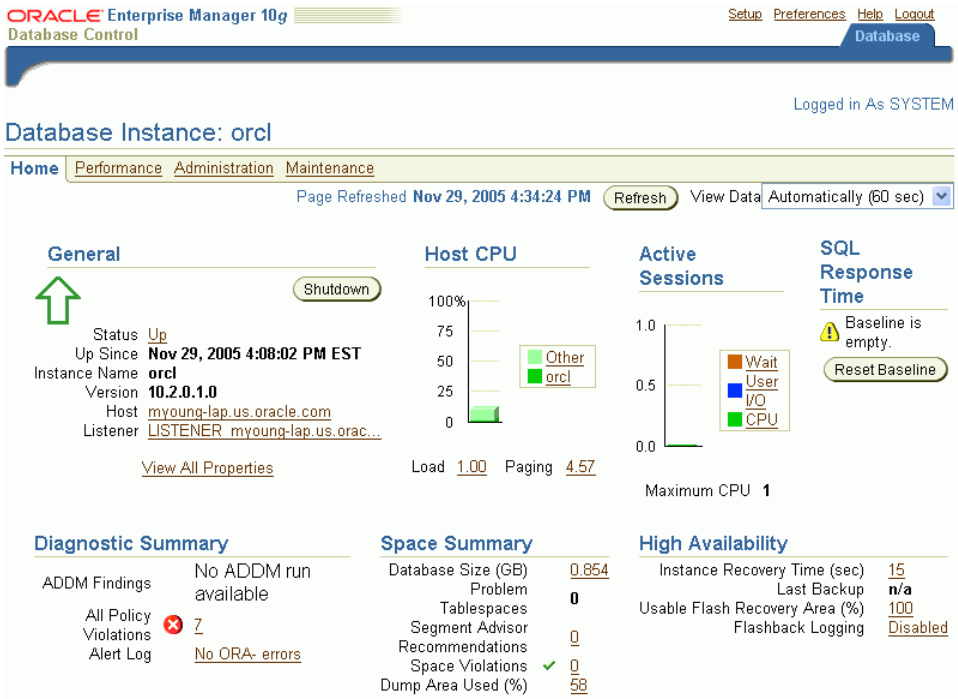
ნახ.1.40

2. გამოჩნდება Licensing ფანჯარა. Scroll-ით გადავახვიოთ ქვევით და ვაწკაპუნებთ **I agree**-ზე.



ნახ.1.41

3. გამოჩნდება Database Control Home Page. თქვენი ინსტალაცია წარმატებით არის შესრულებული.



ნახ.1.42

Oracle Database Express Edition (XE)-ის მიმოხილვა

სამუშაო მაგიდიდან Database Home Page წვდომისათვის Start მენიუში ვასრულებთ მოქმედებებს შემდეგი თანამიმდევრობით:

Start/ Programs (or All Programs)/Oracle Database 10g Express Edition/Go to Database Home Page.

Web-ბრაუზერის გამოყენების შემთხვევაში კი ვკრეფთ შემდეგ URL-ს:

http://127.0.0.1:*port*/apex,

სადაც: *port* არის TCP პორტის ნომერი HTTP შეერთებისათვის, რომლის მნიშვნელობა უსიტყვოდ არის 8080.

დაშორებული კომპიუტერიდან Web-ბრაუზერის შემთხვევაში Database Home Page წვდომისათვის გამოიყენება შემდეგი URL:

http://*host*:*port*/apex,

სადაც: *host* არის იმ კომპიუტერის სახელი ან IP მისამართი, რომელზეც დაინსტალირებულია Oracle Database XE.

როდესაც ჩნდება login გვერდი, სისტემაში დარეგისტრირებისათვის შეგვყავს მომხმარებლის სახელი და პაროლი.

სამუშაო მაგიდიდან მონაცემთა ბაზის გაშვებისათვის:

1. Oracle Database XE host კომპიუტერზე ვრეგისტრირდებით როგორც **Windows** ადმინისტრატორი, ვინაიდან იგი არის Administrator -ის ჯგუფის წევრი მომხმარებელი.

2. მონაცემთა ბაზის წვდომისათვის Start მენიუში ვაწკაპუნებთ შემდეგი თანამიმდევრობით: **Start/Programs (or All Programs)/Oracle Database 10g Express Edition/Start Database**.

ბრძანებითი სტრიქონიდან მონაცემთა ბაზის გაშვებისათვის:

1. Oracle Database XE host კომპიუტერზე ვრეგისტრირდებით როგორც ORA_DBA მომხმარებელთა ჯგუფის წევრი.

2. გავხსნათ ბრძანებების ფანჯარა.

3. SQL Command Line-ისა და მონაცემთა ბაზის დაკავშირებისათვის შევიტანოთ შემდეგი ბრძანება: SQLPLUS / AS SYSDBA

4. SQL Command Line -ში შეგვაქვს STARTUP ბრძანება.

თუ ბრძანება წარმატებით სრულდება, დისპლეიზე ჩნდება შემდეგი ინფორმაცია:

ORACLE instance started.
Total System Global Area 599785472 bytes
Fixed Size 1220804 bytes
Variable Size 180358972 bytes
Database Buffers 415236096 bytes
Redo Buffers 2969600 bytes
Database mounted.
Database opened.

5. შეგვაქვს შემდეგი SQL მოთხოვნა:

```
select count(*) from hr.employees;
The query results should look similar to the following:
COUNT(*)
----------
107
```

6. სისტემიდან გამოსვლისათვის SQL Command Line -ში შეგვაქვს EXIT ბრძანება.

მონაცემთა ბაზის დახურვა სამუშაო მაგიდიდან:

1. Oracle Database XE host კომპიუტერზე დავრეგისტრირდეთ როგორც **Windows** ადმინისტრატორი, ვინაიდან იგი არის Administrator -ის ჯგუფის წევრი მომხმარებელი.

2. მონაცემთა ბაზის წვდომისათვის Start მენიუში ვაწკაპუნებთ შემდეგი თანამიმდევრობით: **Start/Programs (or All Programs)/Oracle Database 10g Express Edition/ Stop Database**.

მონაცემთა ბაზის დახურვა ბრძანებითი სტრიქონის გამოყენებით:

1. Oracle Database XE host კომპიუტერზე ვრეგისტრირდებით როგორც ORA_DBA მომხმარებელთა ჯგუფის წევრი.

2. გავხსნათ ბრძანებების ფანჯარა.

3. SQL Command Line და მონაცემთა ბაზის დაკავშირებისათვის შევიტანოთ შემდეგი ბრძანება: SQLPLUS / AS SYSDBA

4. SQL Command Line -ში შეგვაქვს შემდეგი ბრძანება: SHUTDOWN IMMEDIATE.

თუ ბრძანება წარმატებით სრულდება, დისპლეიზე ჩნდება შემდეგი ინფორმაცია:

Database closed.

Database dismounted.

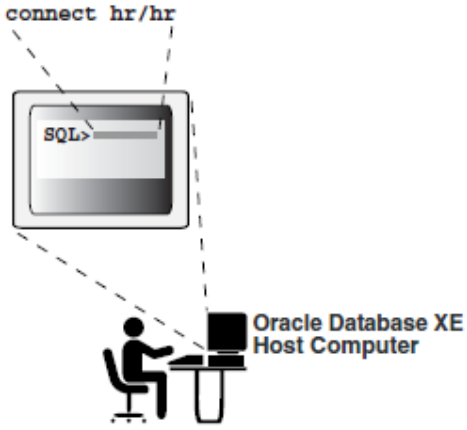
ORACLE instance shut down.

5. სისტემიდან გამოსვლისათვის SQL SQL Command Line -ში შეგვაქვს ბრძანება: EXIT.

შეერთებების შესახებ

განირჩევა Oracle Database XE host კომპიუტერთან შეერთებათა შემდეგი სახეობები:

ა) ლოკალური შეერთება, როდესაც SQL Command Line -ის მუშაობა ხდება იმავე host კომპიუტერზე, სადაც Oracle Database XE არის დაინსტალირებული. ამისათვის საკმარისია მხოლოდ მომხმარებლის სახელისა და პაროლის გამოყენება. მაგალითად, ნახ.1-ზე ნაჩვენებია ლოკალური შეერთების სქემა მომხმარებლის სახელის hr და პაროლის hr გამოყენებით.



ნახ.1.43

ბ) დაშორებული შეერთება, როდესაც SQL Command Line -ის მუშაობა ხდება Oracle Database XE host კომპიუტერისაგან დაშორებულ სხვა კომპიუტერზე, რომელსაც ქსელის მეშვეობით უკავშირდება. ამისათვის აუცილებელია დაშორებულ კომპიუტერზე შემდეგი Oracle კლიენტური პროგრამული უზრუნველყოფის დაინსტალირება:

- Oracle Database Express Edition Client (Oracle Database XE Client)
- Instant Client
- Oracle client software for Oracle Database Enterprise Edition or Stdoard Edition.

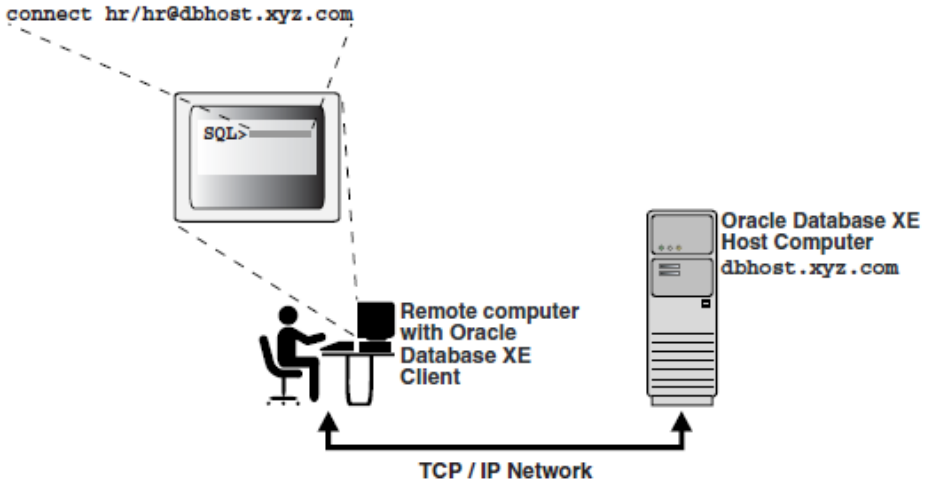
თანმიმდევრული შეერთების (Oracle Net connect string) შემთხვევაში გარდა მომხმარებლის სახელის და პაროლისა connect string შეიცავს host სახელს ან IP მისამართს. ამასთან, არ არის აუცილებელი TCP პორტის ნომერი და მონაცემთა ბაზის სერვერის სახელი. Oracle Net connect string გააჩნია შემდეგი ფორმატი:

username/*password*@[//]*host*[:*port*][/*service_name*]

სადაც:

- // არა აუცილებელია
- *host* host სახელი ან IP მისამართი კომპიუტერისა, რომელზეც მუშაობს Oracle Database XE.
- *port* (არა აუცილებელი) არის Oracle Net მიმღების TCP პორტის ნომერი. უსიტყვოდ მისი მნიშვნელობა არის 1521.
- *service_name* (არა აუცილებელი) არის მონაცემთა ბაზის სერვერის სახელი, რომელთანაც ხდება შეერთება.

ნახ. 1.44-ზე ნაჩვენებია database service (XE) -ის (1521) პორტზე დაშორებული შეერთების სქემა.



ნახ.1.44

Oracle XE Web-ინტერფეისის გაცნობა

მოცემული განყოფილება შეიცავს შემდეგ საკითხებს:

1) სისტემაში შესვლა როგორც მონაცემთა ბაზის ადმინისტრატორი

ამისათვის საჭიროა შემდეგი ბიჯების გაკეთება:

1. გავხსნათ მონაცემთა ბაზის Home Page რეგისტრაციის ფანჯარა:

Windows -ში **Start** მენიუში ვირჩევთ **Programs** (ან **All Programs**), შემდეგ **Oracle Database 10g Express Edition** და შემდეგ **Go To Database Home Page**.

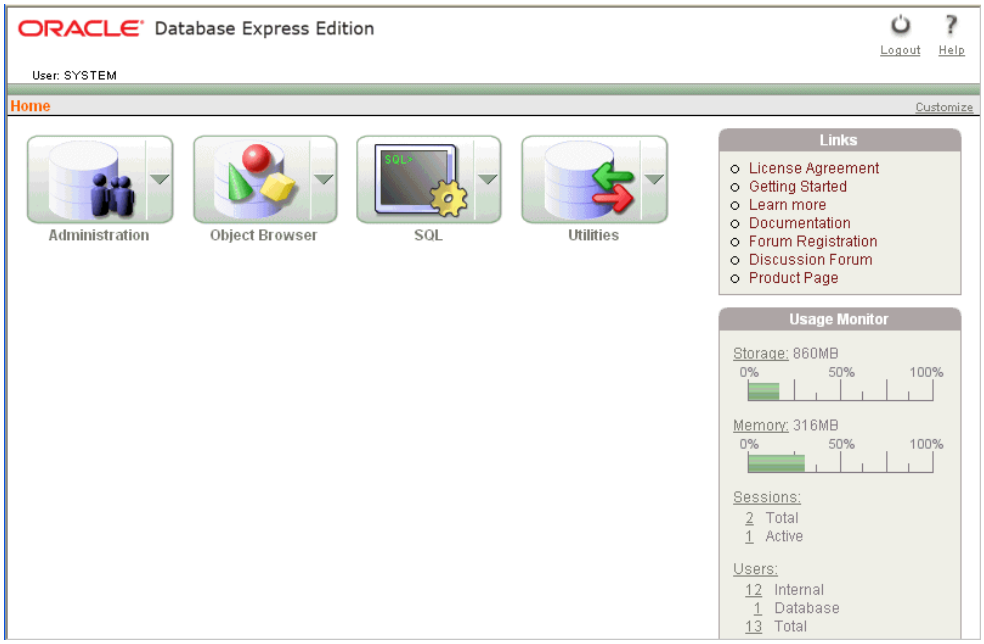
2. მონაცემთა ბაზის Home Page რეგისტრაციის ფანჯარაში შეგვაქვს შემდეგი ინფორმაცია:

Username: Enter system for the user name.

Password: password, რომელიც Oracle Database XE ინსტალაციის დროს იყო გამოყენებული.

3. ვაწკაპუნებთ **Login**.

ჩნდება მონაცემთა ბაზის Home Page რეგისტრაციის ფანჯარა.



ნახ.1.45

2) Sample User Account -ის განბლოკვა

აპლიკაციის შექმნისათვის საჭიროა დარეგისტრირება როგორც მონაცემთა ბაზის მომხმარებელი. Oracle Database XE შეიცავს მონაცემთა ბაზის ნიმუშს სახელწოდებით HR. აქ მომხმარებელს გააჩნია გარკვეული რაოდენობის ცხრილები ნიმუშის სქემაში, რომლებიც შეიძლება გამოყენებულ იქნას ფიქციური აპლიკაციის Human Resources დეპარტამენტის შექმნისათვის. მიუხედავად ამისა, უსაფრთხოების მიზნით მომხმარებლის ანგარიში ბლოკირებულია. ამიტომ ნიმუშის აპლიკაციის აგებამდე საჭირო იქნება მისი განბლოკვა, რისთვისაც:

1. დარწმუნდით, რომ ჯერ კიდევ დარეგისტრირებული ხართ როგორც მონაცემთა ბაზის ადმინისტრატორი.
2. ვაწკაპუნებთ იკონაზე **Administration** და შემდეგ ვაწკაპუნებთ **Database Users**.
3. ვაწკაპუნებთ **HR** სქემაზე HR მომხმარებლის შესახებ ინფორმაციის დისპლეიზე ასახვისათვის.

ნახ.1.46

4. მონაცემთა ბაზის მომხმარებლის მიერ მართვის განხორციელებისათვის შეგვაქვს შემდეგი ინფორმაცია:

5. **Password** და **Confirm Password**: შეგვაქვს პაროლი.

6. **Account Status**: ვირჩევთ **Unlocked**.

7. **Roles**: დავრწმუნდეთ, რომ ორივე **CONNECT** და **RESOURCE** გააქტიურებულია.

8. ვაწკაპუნებთ **Alter User**.

ახლა უკვე მზად ვართ ჩვენი პირველი აპლიკაციის შესაქმნელად.

3) დარეგისტრირება როგორც Sample User Account

ამისათვის:

1. მონაცემთა ბაზის ადმინისტრატორის ანგარიშიდან გამოსვლისათვის ვაწკაპუნებთ **Logout**-ზე Database Home Page -ის მარჯვენა ზედა კუთხეში.
2. ფანჯარაში ვაწკაპუნებთ **Login**.
3. Login ფანჯარაში შეგვაქვს ორივე: მომხმარებლის სახელი და პაროლი.
4. ვაწკაპუნებთ **Login**.

გამოჩნდება მონაცემთა ბაზის Home Page.

4) მარტივი აპლიკაციის შექმნა

აპლიკაციის შექმნა არის მარტივი გზა მონაცემთა ბაზის მონაცემების ნახვისა და რედაქტირებისათვის. შევქმნათ აპლიკაცია ცხრილისათვის EMPLOYEES, რომელიც არის HR სქემის ნაწილი. ამისათვის:

1. მონაცემთა ბაზის Home Page -ზე ვაწკაპუნებთ იკონაზე **Application Builder**.
2. ვაწკაპუნებთ ღილაკზე **Create**.
3. ვირჩევთ **Create Application** და ვაწკაპუნებთ **Next**.
4. აპლიკაციის შექმნისათვის შეგვაქვს:

   Name: Enter MyApp.

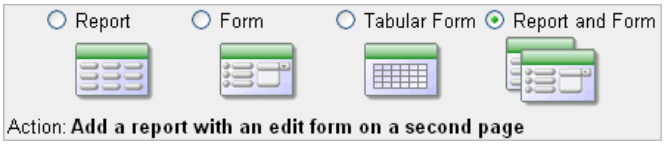
   ვეთანხმებით ყველა შენახულ (დარჩენილ) defaults.

   ვაწკაპუნებთ **Next**.

შემდეგ ვამატებთ გვერდებს თქვენს აპლიკაციას.

5. ყოველ დამატებულ გვერდზე:

   გვერდის ტიპის განსაზღვრისათვის ვირჩევთ **Report და Form**.



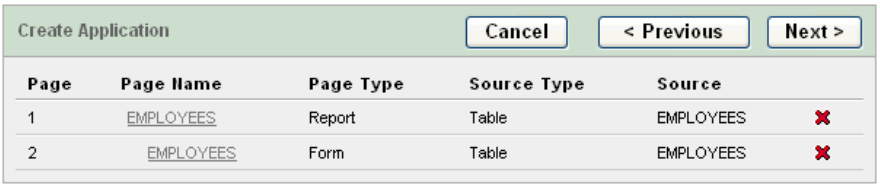
ნახ.1.47

ვნახოთ, რომ **Action** აღწერს თქვენ მიერ დამატებული გვერდის ტიპს.

შემდეგ ველში **ცხრილი Name**, ვაწკაპუნებთ ზემოთ მიმართულ ისარს და შემდეგ Search Dialog ფანჯრიდან ვირჩევთ **EMPLOYEES**.

ვაწკაპუნებთ **Add Page**.

ორი ახალი გვერდი ჩნდება გვერდის თავში, Create Application-ის ქვეშ.



Page	Page Name	Page Type	Source Type	Source	
1	EMPLOYEES	Report	Table	EMPLOYEES	✖
2	EMPLOYEES	Form	Table	EMPLOYEES	✖

ნახ.1.48

ვაწკაპუნებთ **Next**.

6. პანელზე Tabs ვეთანხმებით პირობას (**One Level of Tabs**) და ვაწკაპუნებთ **Next**.

7. პანელზე Shared Components ვეთანხმებით პირობას (**No**) და ვაწკაპუნებთ **Next**.

ეს ოპცია გააქტიურებულია სხვა აპლიკაციიდან საერთო კომპონენტების იმპორტის თვალსაზრისით. საერთო კომპონენტები წარმოადგენენ ერთობლივ ელემენტებს, რომლებიც შეიძლება დისპლეიზე აისახოს ან გამოყენებულ იქნას აპლიკაციის შიგნით ნებისმიერ გვერდზე.

8. სქემის და მომხმარებლის მიერ მიზანშეწონილი ენის აუთენტიფიკაციისათვის ვიღებთ defaults და ვაწკაპუნებთ **Next**.

9. თემებისათვის ვირჩევთ **Theme 2** ვაწკაპუნებთ **Next**.

თემები წარმოადგენს ნიმუშების კოლექციას, რომელიც შეიძლება გამოვიყენოთ მთელი აპლიკაციის დაგეგმვისა და სტილის განსაზღვრისათვის.

10. ვადასტურებთ ჩვენს შერჩეულს. წინა wizard page-ში დაბრუნებისათვის ვაწკაპუნებთ **Previous**. შემდეგ ვაწკაპუნებთ **Create**.

რის შემდეგაც გვერდის თავში გამოჩნდება შემდეგი შეტყობინება:
Application created successfully.

5) თქვენი ახალი აპლიკაციის შესრულებაზე გაშვება

1. ვაწკაპუნებთ იკონაზე **Run Application**.

ნახ.1.49

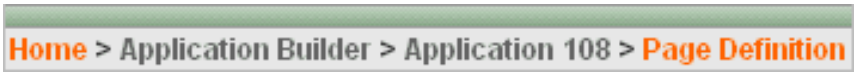
2. რეგისტრაციის გვერდზე შეგვაქვს hr ორივე: **User Name** და **Password**. თქვენი დანართი გამოჩნდება ცხრილით EMPLOYEES.
3. შევისწავლოთ ჩვენი აპლიკაცია.

ჩვენ სურვილისამებრ შეგვიძლია EMPLOYEES ცხრილის მოთხოვნის ფორმირება და შესრულება, რისთისაც გვერდის ქვედა ნაწილში ვიყენებთ Developer ინსტრუმენტების სტრიქონს.

ნახ.1.50

ეს უკანასკნელი არის სწრაფი გზა მიმდინარე გვერდის რედაქტი-რებისათვის, ახალი გვერდის, მართვის ან კომპონენტის შექმნისათვის, სესიის მდგომარეობის ნახვისა და სხვა. აპლიკაციიდან გასვლისა და Application Builder- ში დაბრუნებისათვის Developer ინსტრუმენტების სტრიქონზე ვაწკაპუნებთ **Edit Page 1**.

4. მონაცემთა ბაზის Home Page-ში დაბრუნებისათვის, გვერდის თავში ვირჩევთ **Home** შემდეგ ფრაგმენტებს.



ნახ.1.51

შედეგად ვიღებთ ჩვენს პირველ აპლიკაციას Oracle Database XE-ის გამოყენებით.

ლაბორატორიული სამუშაო 2

მონაცემთა განსაზღვრების ენის (Data Definition Language -DDL) შესწავლა

სამუშაოს მიზანი:

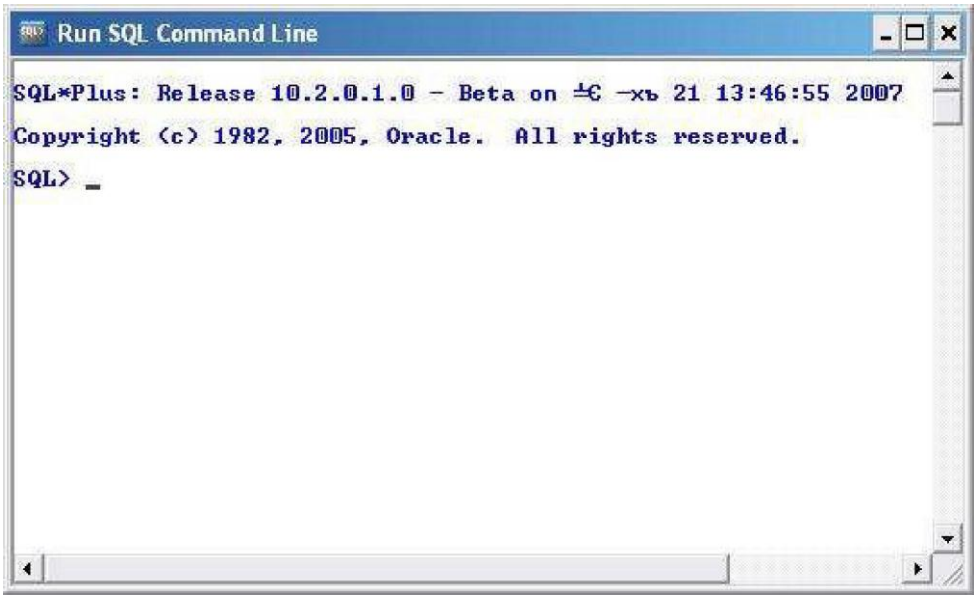
1. SQL*Plus-თან მუშაობის დაწყება;

2. მონაცემთა განსაზღვრების ენის (Data Definition Language - DDL) ბრძანებების შესწავლა.

SQL*Plus-თან მუშაობის დაწყება

SQL*Plus იხსნება შემდეგი ბრძანებით:

Start ► Programs ► Oracle Database 10g Express Edition ► Run SQL Command Line და ვაწკაპუნებთ.

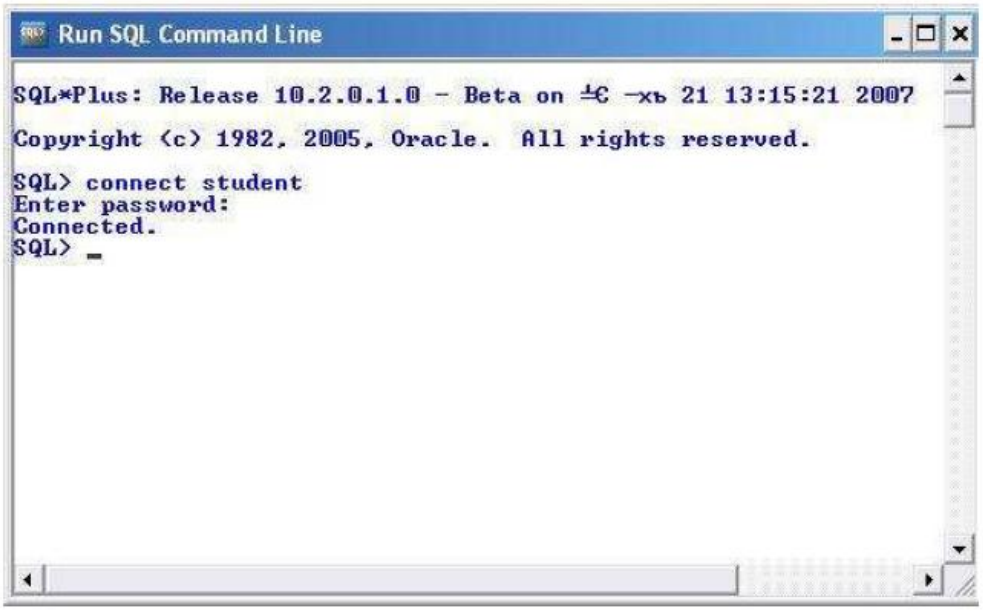


ნახ.2.1

მონაცემთა ბაზასთან შეერთებისათვის საჭიროა ავკრიფოთ:

connect მომხმარებლის სახელი

გამოდის შემდეგი შეტყობინება.



ნახ.2.2

მონაცემთა განსაზღვრების ენის (DDL) ბრძანებები

მოცემული ლაბორატორიული სამუშაო ეძღვნება SQL ენის ბრძანებებისა და SQL*Plus (Windows და კონსოლური ვერსიის) გარსის გამოყენების ჩვევების ათვისებას. განვიხილოთ SQL ენის ზოგიერთი ბრძანება:

CREATE ცხრილი — მონაცემთა ბაზის ცხრილის შექმნა;

INSERT INTO — ცხრილში მონაცემების (სტრიქონების) ჩასმა;

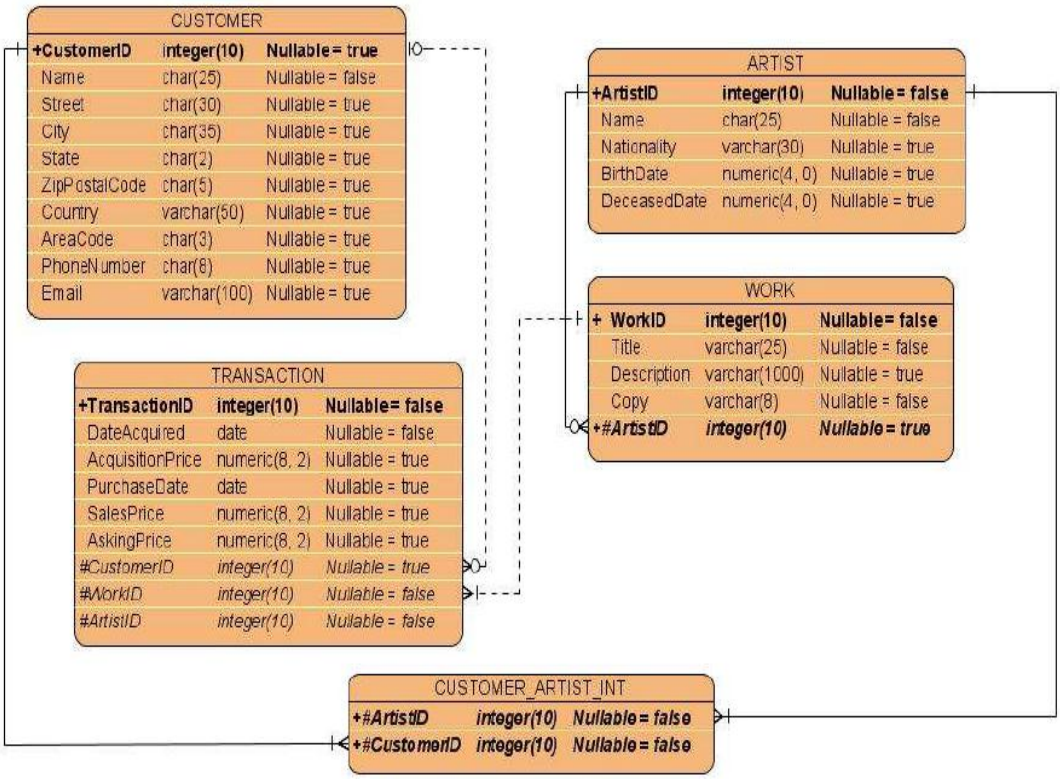
SELECT — ცხრილიდან მონაცემების (სტრიქონების) ამორჩევა;

DELETE — ცხრილიდან მონაცემების (სტრიქონების) ამოგდება;

DESCRIBE — ცხრილის აღწერა;

DROP ცხრილი — მონაცემთა ბაზიდან ცხრილის ამოგდება.

ცხრილების შექმნა. მაგალითისათვის განვიხილოთ შემდეგი მონაცემთა ბაზის ცხრილები შესაბამისი სტრუქტურებით, რომელთა შესაქმნელად გამოიყენება ბრძანება: create_ცხრილი.sql.



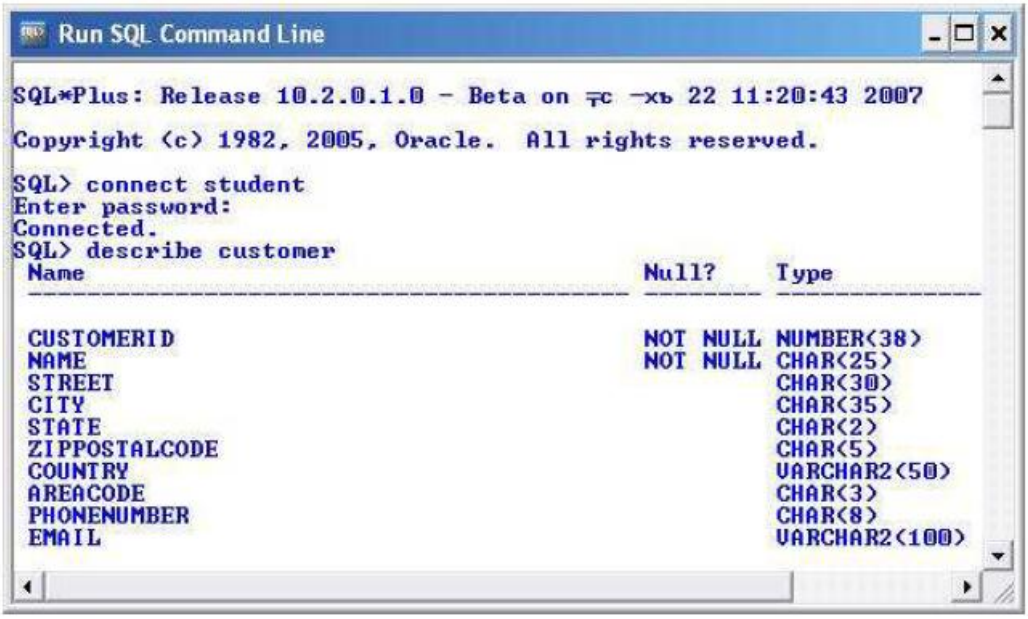
ნახ.2.3

```
CREATE TABLE CUSTOMER(
    CustomerID      int             NOT NULL,
    Name            char(25)        NOT NULL,
    Street          char(30)        NULL,
    City            char(35)        NULL,
    State           char(2)         NULL,
    ZipPostalCode   char(5)         NULL,
```

```
    Country         varchar(50)     NULL,
    AreaCode        char(3)         NULL,
    PhoneNumber     char(8)         NULL,
    Email           varchar(100)    NULL,
    CONSTRAINT  CustomerPK
                PRIMARY KEY (CustomerID));

CREATE TABLE ARTIST(
    ArtistID        int             NOT NULL,
    Name            char(25)        NOT NULL,
    Nationality     varchar(30)     NULL,
    BirthDate       numeric(4,0)    NULL,
    DeceasedDate    numeric(4,0)    NULL,
    CONSTRAINT  ArtistPK    PRIMARY KEY (ArtistID),
    CONSTRAINT  ArtistAK1 UNIQUE (Name),
    CONSTRAINT  NationalityValues CHECK (Nationality IN
                ('Canadian', 'English', 'French', 'German',
                'Mexican', 'Russian', 'Spanish', 'US')),
    CONSTRAINT  BirthValuesCheck CHECK (BirthDate < DeceasedDate),
    CONSTRAINT  ValidBirthYear CHECK ((BirthDate > 1000) AND
                (BirthDate < 2100)),
    CONSTRAINT  ValidDeathYear CHECK ((DeceasedDate > 1000) AND
                (DeceasedDate < 2100)));

        CREATE TABLE CUSTOMER_ARTIST_INT(
            ArtistID        int         NOT NULL,
            CustomerID      int         NOT NULL,
            CONSTRAINT  CustomerArtistPK
                        PRIMARY KEY (ArtistID, CustomerID),
            CONSTRAINT  Customer_Artist_Int_ArtistFK
                        FOREIGN KEY (ArtistID)
                        REFERENCES ARTIST (ArtistID)
                        ON DELETE CASCADE,
            CONSTRAINT  Customer_Artist_Int_CustomerFK
                        FOREIGN KEY (CustomerID)
                        REFERENCES CUSTOMER (CustomerID)

                        ON DELETE CASCADE);
```

```
CREATE TABLE WORK(
    WorkID      int         NOT NULL,
    Title           varchar(25)     NOT NULL,
    Description     varchar(1000)   NULL,
    Copy            varchar(8)      NOT NULL,
    ArtistID        int         NOT NULL,
    CONSTRAINT  WorkPK
                PRIMARY KEY (WorkID),
    CONSTRAINT  WorkAK1
                UNIQUE (Title, Copy),
    CONSTRAINT  ArtistFK
                FOREIGN KEY (ArtistID)
                REFERENCES ARTIST (ArtistID));


CREATE TABLE TRANSACTION(
    TransactionID   int         NOT NULL,
    DateAcquired    Date            NOT NULL,
    AcquisitionPrice    Numeric(8,2)    NULL,
    PurchaseDate    Date            NULL,
    SalesPrice      Numeric(8,2)    NULL,
    AskingPrice     Numeric(8,2)    NULL,
    CustomerID      int         NULL,
    WorkID      int         NOT NULL,
    CONSTRAINT  TransactionPK
                PRIMARY KEY (TransactionID),
    CONSTRAINT  SalesPriceRange
                CHECK ((SalesPrice > 1000) AND
                (SalesPrice <= 200000)),
    CONSTRAINT  ValidTransDate
                CHECK (DateAcquired <= PurchaseDate),
    CONSTRAINT  TransactionWorkFK
                FOREIGN KEY (WorkID)
                REFERENCES  WORK (WorkID),
    CONSTRAINT  TransactionCustomerFK
     FOREIGN KEY (CustomerID)
     REFERENCES  CUSTOMER (CustomerID));
```

ცხრილების მდგომარეობის შესამოწმებლად გამოიყენება ბრძანება DESCRIBE ანუ DESC.



ნახ.2.4

განხილული მაგალითის ყველა ცხრილს, გარდა CUSTOMER_ARTIST_INT გააჩნია ე.წ. სუროგატული გასაღებები, რომელთა პირდაპირი განსაზღვრა Oracle-ში არ ხდება. ამისათვის გამოიყენება ე.წ. მიმდევრობები (sequence) — ობიექტი, სადაც ხდება უნიკალურ რიცხვთა მიმდევრობის გენერაცია. შემდეგი ოპერატორი განსაზღვრავს მიმდევრობას სახელწოდებით CustID, რომელიც იწყება 100-დან და ყოველ გამოყენებაზე იზრდება 1-ით:

Create Sequence CustID Increment by 1 start with 100;

არსებობს მიმდევრობების ორი მნიშვნელოვანი მეთოდი:

- NextVal იძლევა მიმდევრობის შემდეგ მნიშვნელობას;

-CurrVal იძლევა მიმდევრობის მიმდინარე მნიშვნელობას.

ამგვარად, CustID.NextVal იძლევა CustID მიმდევრობის შემდეგ მნიშვნელობას:

```
INSERT INTO CUSTOMER
(CustomerID, Name, AreaCode, PhoneNumber)
VALUES (CustID.NextVal, 'Mary Jones', '350', '555-1234');
```

მონაცემთა შეტანა. მიმდევრობების მეშვეობით შესაძლებელია ცხრილების მონაცემებით შევსება, რისთვისაც გამოიყენება ბრძანება INSERT.

```
INSERT INTO ARTIST VALUES
(ArtistID.NextVal, 'Miro', 'Spanish', 1870, 1950);
INSERT INTO ARTIST VALUES
(ArtistID.NextVal, 'Kqoainsky', 'Russian', 1854, 1900);
INSERT INTO ARTIST VALUES
(ArtistID.NextVal, 'Frings', 'US', 1700, 1800);
INSERT INTO ARTIST VALUES
(ArtistID.NextVal, 'Klee', 'German', 1900, NULL);
INSERT INTO ARTIST VALUES
(ArtistID.NextVal, 'Moos', 'US', NULL, NULL);
INSERT INTO ARTIST VALUES
(ArtistID.NextVal, 'Tobey', 'US', NULL, NULL);
INSERT INTO ARTIST VALUES
(ArtistID.NextVal, 'Matisse', 'French', NULL, NULL);
INSERT INTO ARTIST VALUES
(ArtistID.NextVal, 'Chagall', 'French', NULL, NULL);
INSERT INTO CUSTOMER VALUES
(CustID.NextVal, 'Jeffrey Janes', '123 W. Elm St',
'Renton', 'WA', '98123', 'USA', '206', '555-1345',
'Customerl000@somewhere.com');
INSERT INTO CUSTOMER VALUES
(CustID.NextVal, 'David Smith', '813 Tumbleweed Lane',
'Lovelqa', 'CO', '80345', 'USA', '303', '555-5434',
'Customerl001@somewhere.com');
INSERT INTO CUSTOMER VALUES
(CustlD.NextVal, 'Tiffany Twilight', '88 - First Avenue',
'Langley', 'WA', '98114', 'USA', '206', '555-1000',
'Customerl015@somewhere.com');
INSERT INTO CUSTOMER VALUES
(CustlD.NextVal, 'Fred Smathers', '10899-88th Ave',
'Bainbridge Islqa', 'WA', '98108', 'USA', '206', '555-1234',
'Customerl033@somewhere.com');
INSERT INTO CUSTOMER VALUES
```

```
(CustlD.NextVal, 'Mary Beth Frederickson', '25 South Lafayette',
'Denver', 'CO', '80210', 'USA', '303', '555-1000',
'Customerl034@somewhere.com');
INSERT INTO CUSTOMER VALUES
(CustlD.NextVal, 'Selma Warning', '205 Burnaby',
'Vancouver', 'BC, 'V0N1B', 'Canada', '253', '555-1234',
'Customerl036@somewhere.com');
INSERT INTO CUSTOMER VALUES
(CustlD.NextVal, 'Susan Wu', '105 Locust Ave', 'Atlanta',
'GA', '23224', 'USA', '721', '555-1234',
'Customerl037@somewhere.com');
INSERT INTO CUSTOMER VALUES
(CustlD.NextVal, 'Donald G. Gray', '55 Bodega Ave',
'Bodega Bay', 'CA', '92114', 'USA', '705', '555-1345',
'Customerl040@somewhere.com');
INSERT INTO CUSTOMER VALUES
(CustlD.NextVal, 'Lynda Johnson', '117 C Street',
'Washington', 'DC, '11345', 'USA', '703', '555-1000', '');
INSERT INTO CUSTOMER VALUES
(CustlD.NextVal, 'Chris Wilkens', '87 Highlda Drive',
'Olympia', 'WA', '98008', 'USA', '206', '555-1234', '');
```

ოპერატორები DROP და ALTER

DROP-ის მეშვეობით შესაძლებელია ბაზიდან სხვადასხვა სტრუქტურების (ცხრილის, მიმდევრობის, სვეტის და ა.შ.) წაშლა. მაგალითად:

```
DROP ცხრილი MyTaЬIe;
DROP SEQUENCE MySequence;
```

შემდეგ მაგალითში DROP ოპერატორის საშუალებით იშლება სვეტი, რაც ცვლილებების შეტანასაც ახორციელებს:

```
ALTER ცხრილი MyTaЬIe DROP COLUMN MyColumn;
```

DATE ტიპის მონაცემების შეტანა. DATE ტიპის მონაცემების შეტანისათვის საჭიროა გარკვეული ფორმატის დაცვა, რაც შესაძლებელია ფუნქციის TO_DATE მეშვეობით. მაგალითად:

```
T0_DATE('11/12/2002',   'MM/DD/YYYY')
```

მონაცემთა შეტანის ბრძანებაში იქნება:

```
INSERT INTO Tl VALUES (100, T0_DATE('05/01/2007',
'MM/DD/YYYY'));
```

ინდექსების შექმნა. სვეტების უნიკალურობის უზრუნველყოფის, სორტირების გამარტივების და სწრაფი ძიებისათვის გამოიყენება სვეტების ინდექსირება. მაგალითად, CUSTOMER ცხრილის Name სვეტისათვის ინდექსის შესაქმნელად ვიყენებთ:

```
CREATE INDEX CustNameldx ON CUSTOMER(Name);
```

უნიკალური ინდექსის შექმნისათვის INDEX გასაღებური სიტყვის წინ უნდა ჩაისვას გასაღებური სიტყვა UNIQUE. მაგალითად:

```
CREATE UNIQUE INDEX WorkUniqueIndex ON W0RK(Title, Copy,
ArtistID);
```

ცხრილის სტრუქტურის შეცვლა. ამისათვის გამოიყენება ოპერატორი ALTER ცხრილი. მაგალითად განვიხილოთ ორი შემთხვევა:

```
ALTER ცხრილი Myცხრილი
ADD Cl NUMBER(4);
ALTER ცხრილი Myცხრილი
DROP COLUMN C1;
```

ცვლილებების შესამოწმებლად ვიყენებთ ოპერატორს DESCRIBE.

(NOT NULL) აუცილებელი სვეტის დასამატებლად, იგი ჯერ ცხრილში უნდა შევქმნათ როგორც არააუცილებელი, შევიტანოთ მასში მონაცემები და მხოლოდ ამის შემდეგ შეგვიძლია ის გამოვაცხადოთ როგორც აუცილებელი MODIFYკონსტრუქციის საშუალებით. მაგალითად:

```
ALTER ცხრილი Tl MODIFY Cl NOT NULL;
```

სააღრიცხვო ჩანაწერები და როლები

ცნებები „სააღრიცხვო ჩანაწერი“, „სქემა“ და „მომხმარებელი“ Oracle-ში ერთიდაიგივე შინაარსის არის. განირჩევა ორი სქემა SYS და SYSTEM. სქემა SYS შეიცავს ყველა სისტემურ ობიექტს, როგორიცაა - ბაზის შინაგანი ცხრილები, პაკეტები, პროცედურები.

SYS მომხმარებელი წარმოადგენს მონაცემთა ლექსიკონის მფლობელს. Oracle-ის მონაცემთა ლექსიკონი არის - ცხრილების, წარმოდგენების ერთობლიობა, რომელიც საშუალებას იძლევა მივიღოთ

ინფორმაცია მონაცემთა ბაზის სტრუქტურის, მდგომარეობის შესახებ სტანდარტული SQL-მოთხოვნების მეშვეობით. SYS სააღრიცხვო ჩანაწერი წარმოადგენს მბ ადმინისტრატორის სააღრიცხვო ჩანაწერს შეუზღუდავი უფლებებით.

SYSTEM სააღრიცხვო ჩანაწერი იძლევა ბაზის ყველა ობიექტთან წვდომის საშუალებას და გააჩნია DBA როლი. SYS და SYSTEM წაშლა დაუშვებელია.

როლები Oracle-ში წარმოადგენს პრივილეგიათა სახელდებულ ჯგუფებს. მონაცემთა ბაზის შექმნასთან ერთად იქმნება რამდენიმე სტანდარტული როლი:

- CONNECT — შეიცავს მხოლოდ ერთადერთ პრივილეგიას CREATE SESSION, რომელიც ბაზასთან შეერთების შექმნას ახორციელებს;
- DBA — პრივილეგიათა სრული კრებული;
- RESOURCE — პრივილეგიათა საბაზო კრებული, რომელიც აუცილებელია შემმუშავებლისათვის;
- DELETE_CATALOG_ROLE — აუდიტის ცხრილიდან ინფორმაციის წაშლის პრივილეგია;
- SELECT_CATALOG_ROLE — აუდიტის ცხრილიდან ინფორმაციის წაკითხვის პრივილეგია;
- EXP_FULL_DATABASE — ბაზის სრული ექსპორტისათვის აუცილებელი პრივილეგია;
- IMP_FULL_DATABASE — ბაზის სრული იმპორტისათვის აუცილებელი პრივილეგია.

ლაბორატორიული სამუშაო 3

მონაცემთა განსაზღვრების ენის (Data Manipulation Language -DML) შესწავლა

სამუშაოს მიზანი:

1. მონაცემთა მანიპულირების ენის (DML) ბრძანებების შესწავლა;
2. მონაცემთა ბაზის ცხრილებიდან ინფორმაციის ამორჩევის შესწავლა.

ინფორმაციის ამორჩევა მონაცემთა ბაზის ცხრილებიდან

ბრძანების SELECT შესრულება ერთი ცხრილის შემთხვევაში

მაგალითი:

```
SELECT customer_id, first_name, last_name, dob, phone
FROM customers;
```

```
CUSTOMER_ID FIRST_NAME LAST_NAME  DOB       PHONE
----------- ---------- ---------- --------- ------------
          1 John       Brown      01-JAN-65 800-555-1211
          2 Cynthia    Green      05-FEB-68 800-555-1212
          3 Steve      White      16-MAR-71 800-555-1213
          4 Gail       Black                800-555-1214
          5 Doreen     Blue       20-MAY-70
```

თუ გვინდა ცხრილის ყველა სვეტის ამორჩევა, მაშინ სვეტების ჩამონათვალის ნაცვლად ვიყენებთ სიმბოლოს (*).

მაგალითი:

```
SELECT *
FROM customers;
```

```
CUSTOMER_ID FIRST_NAME LAST_NAME  DOB       PHONE
----------- ---------- ---------- --------- ------------
          1 John       Brown      01-JAN-65 800-555-1211
          2 Cynthia    Green      05-FEB-68 800-555-1212
          3 Steve      White      16-MAR-71 800-555-1213
          4 Gail       Black                800-555-1214
          5 Doreen     Blue       20-MAY-70
```

Oracle-ის მონაცემთა ბაზის თითოეულ სტრიქონს გააჩნია უნიკალური იდენტიფიკატორი ანუ *rowid*, რომელიც გამოიყენება სტრიქონის წვდომისათვის. rowid წარმოადგენს 18-სიმბოლოიან რიცხვს და ის შეიცავს სტრიქონის ფიზიკურ მისამართს Oracle-ის მონაცემთა ბაზაში. შემდეგ მაგალითში მოთხოვნის შესაბამისად ხდება ROWID და customer_id სვეტების გამოტანა.

მაგალითი:

```
SELECT ROWID, customer_id
FROM customers;

ROWID                  CUSTOMER_ID
------------------      -----------
AAAF4yAABAAAHeKAAA        1
AAAF4yAABAAAHeKAAB        2
AAAF4yAABAAAHeKAAC        3
AAAF4yAABAAAHeKAAD        4
AAAF4yAABAAAHeKAAE        5
```

როგორც წესი, DESCRIBE ბრძანების გამოყენების შემთხვევაში ROWID არ გამოდის, ამიტომ ის ფსევდო (*pseudo)* სვეტის სახელწოდებით არის ცნობილი.

არითმეტიკული ოპერატორები

Operator	**Description**
+	მიმატება
-	გამოკლება
*	გამრავლება
/	გაყოფა

შემდეგ მაგალითში გამოსახულება (*expression)* შეიძლება შეიცავდეს სვეტების, ლიტერალთა მნიშვნელობების და ოპერატორების კომბინაციას.

მაგალითი:

```
SELECT 2*6
FROM dual;

      2*6
----------
       12
```

ორმაგი ცხრილები

ორმაგი არის ცხრილი, რომელიც შეიცავს ერთადერთ სტრიქონს.

მაგალითი:

DESCRIBE dual;

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 DUMMY                                              VARCHAR2(1)

SELECT *
FROM dual;

D
-
X
```

ამ შემთხვევაში ცხრილის VARCHAR2 ტიპის სვეტი dummy შეიცავს სტრიქონს X მნიშვნელობით.

თარიღების არითმეტიკის გამოყენება

მაგალითი:

```
SELECT TO_DATE('31-JUL-2003') + 2
FROM dual;
TO_DATE(
---------
02-AUG-03
```

მაგალითი:

```
SELECT TO_DATE('02-AUG-2003') - 2
FROM dual;
TO_DATE('
---------
31-JUL-03
```

მაგალითი:

```
SELECT TO_DATE('02-AUG-2003') - TO_DATE('31-JUL-2003')
FROM dual;

TO_DATE('02-AUG-2003')-TO_DATE('31-JUL-2003')
---------------------------------------------
                                            2
```

სვეტების გამოყენება არითმეტიკაში

მაგალითი:

```
SELECT name, price + 2
FROM products;
```

NAME	PRICE+2
Modern Science	21.95
Chemistry	32
Supernova	27.99
Tank War	15.95
Z Files	51.99
2412: The Return	16.95
Space Force 9	15.49
From Another Planet	14.99
Classical Music	12.99
Pop 3	17.99
Creative Yell	16.99
My Front Line	15.49

მაგალითი:

```
SELECT name, price * 3 + 1
FROM products;
```

NAME	PRICE*3+1
Modern Science	60.85
Chemistry	91
Supernova	78.97
Tank War	42.85
Z Files	150.97
2412: The Return	45.85
Space Force 9	41.47
From Another Planet	39.97
Classical Music	33.97
Pop 3	48.97
Creative Yell	45.97
My Front Line	41.47

მაგალითი:

```
SELECT 10 * 12 / 3 – 1
FROM dual;

 10*12/3-1
----------
        39
```

მაგალითი:

```
SELECT 10 * (12 / 3 – 1)
FROM dual;

10*(12/3-1)
-----------
         30
```

სვეტების შერწყმა კონკატენაციის გამოყენებით

სვეტების შეერთება-შერწყმისათვის გამოიყენება კონკატენაციის ოპერატორი (||).

მაგალითი:

```
SELECT first_name || ' ' || last_name AS "Customer Name"
FROM customers;

        Customer Name
        --------------------
        John Brown
        Cynthia Green
        Steve White
        Gail Black
        Doreen Blue
```

Null მნიშვნელობის გაგება

მაგალითი:

```
SELECT customer_id, first_name, last_name, dob
FROM customers
WHERE dob IS NULL;
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB
4	Gail	Black	

მაგალითი:

```
SELECT customer_id, first_name, last_name, phone
FROM customers
WHERE phone IS NULL;
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	PHONE
5	Doreen	Blue	

მაგალითი:

```
SELECT customer_id, first_name, last_name,
NVL(phone, 'Unknown phone number') AS PHONE_NUMBER
FROM customers;
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	PHONE_NUMBER
1	John	Brown	800-555-1211
2	Cynthia	Green	800-555-1212
3	Steve	White	800-555-1213
4	Gail	Black	800-555-1214
5	Doreen	Blue	Unknown phone number

მაგალითი:

```
SELECT customer_id, first_name, last_name,
NVL(dob, '01-JAN-2000') AS DOB
FROM customers;
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB
1	John	Brown	01-JAN-65
2	Cynthia	Green	05-FEB-68
3	Steve	White	16-MAR-71
4	Gail	Black	01-JAN-00
5	Doreen	Blue	20-MAY-70

განსხვავებული სტრიქონების გამოტანა

მაგალითი:

```
SELECT purchased_by
FROM purchases;

PURCHASED_BY
------------
         1
         1
         4
         2
         3
         2
         3
         4
         3
```

მაგალითი:

```
SELECT DISTINCT purchased_by
FROM purchases;

PURCHASED_BY
------------
         1
         2
         3
         4
```

სტრიქონების გაფილტვრა **WHERE** პირობის გამოყენებით

მაგალითი:

```
SELECT *
FROM customers
WHERE customer_id = 2;

CUSTOMER_ID    FIRST_NAME       LAST_NAME   DOB           PHONE
-----------    ----------       ----------  ---------     ------------
       2       Cynthia          Green       05-FEB-68     800-555-1212
```

შედარების ოპერატორები

ხშირად WHERE პირობასთან ერთად გამოიყენება შედარების ოპერატორები.

Operator	Description
=	ტოლია
<> or !=	არ არის ტოლი
<	ნაკლებია ვიდრე
>	მეტია ვიდრე
<=	ნაკლებია ან ტოლი
>=	მეტია ან ტოლი
ANY	ადარებს სიიდან ნებისმიერ მნიშვნელობას
ALL	ადარებს სიიდან ყველა მნიშვნელობას

მაგალითი:

```
SELECT *
FROM customers
WHERE customer_id <> 2;
```

```
CUSTOMER_ID    FIRST_NAME    LAST_NAME     DOB          PHONE
-----------    ----------    ----------    ---------    ------------
1              John          Brown         01-JAN-65    800-555-1211
3              Steve         White         16-MAR-71    800-555-1213
4              Gail          Black                      800-555-1214
5              Doreen        Blue          20-MAY-70
```

მაგალითი:

```
SELECT customer_id, name
FROM products
WHERE customer_id > 8;
```

```
CUSTOMER_ID    NAME
-----------    ----------------
9              Classical Music
10             Pop 3
11             Creative Yell
```

12 My Front Line

მაგალითი:

```
SELECT *
FROM customers
WHERE customer_id > ANY (2, 3, 4);
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB	PHONE
5	Doreen	Blue	20-MAY-70	
4	Gail	Black		800-555-1214
3	Steve	White	16-MAR-71	800-555-1213

მაგალითი:

```
SELECT *
FROM customers
WHERE customer_id > ALL (2, 3, 4);
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB	PHONE
5	Doreen	Blue	20-MAY-70	

SQL ოპერატორების გამოყენება

Operator	Description
LIKE	Matches patterns in strings
IN	Matches lists of values
BETWEEN	Matches a range of values
IS NULL	Matches null values
IS NAN	Matches the NaN special value, which means “not a number”
IS INFINITE	Matches infinite BINARY_FLOAT და BINARY_ values

LIKE ოპერატორის გამოყენება

მაგალითი:

```
SELECT *
FROM customers
```

WHERE first_name LIKE '_o%';

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB	PHONE
1	John	Brown	01-JAN-65	800-555-1211
5	Doreen	Blue	20-MAY-70	

მაგალითი:

```
SELECT *
FROM customers
WHERE first_name NOT LIKE '_o%';
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB	PHONE
2	Cynthia	Green	05-FEB-68	800-555-1212
3	Steve	White	16-MAR-71	800-555-1213
4	Gail	Black		800-555-1214

მაგალითი:

```
SELECT first_name
FROM customers
WHERE first_name LIKE '%a\_product%' ESCAPE '\';
```

IN ოპერატორის გამოყენება

მაგალითი:

```
SELECT *
FROM customers
WHERE customer_id IN (2, 3, 5);
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB	PHONE
2	Cynthia	Green	05-FEB-68	800-555-1212
3	Steve	White	16-MAR-71	800-555-1213
5	Doreen	Blue	20-MAY-70	

მაგალითი:

```
SELECT *
FROM customers
WHERE customer_id NOT IN (2, 3, 5, NULL);
```

no rows selected

BETWEEN ოპერატორის გამოყენება

მაგალითი:

```
SELECT *
FROM customers
WHERE customer_id BETWEEN 1 და 3;
```

```
CUSTOMER_ID FIRST_NAME LAST_NAME  DOB       PHONE
----------- ---------- ---------- --------- ------------
          1 John       Brown      01-JAN-65 800-555-1211
          2 Cynthia    Green      05-FEB-68 800-555-1212
          3 Steve      White      16-MAR-71 800-555-1213
```

ლოგიკური ოპერატორების გამოყენება

Operator	Description
x და *y*	Returns true when both x და y are true
x OR *y*	Returns true when either x or y is true
NOT *x*	Returns true if x is false, და returns false if x is true

მაგალითი:

```
SELECT *
FROM customers
WHERE dob > '01-JAN-1970'
და customer_id > 3;
```

```
CUSTOMER_ID  FIRST_NAME  LAST_NAME  DOB        PHONE
-----------  ----------  ----------  ---------  ------------
          5  Doreen      Blue       20-MAY-70
```

მაგალითი:

```
SELECT *
FROM customers
WHERE dob > '01-JAN-1970'
OR customer_id > 3;
```

```
CUSTOMER_ID  FIRST_NAME  LAST_NAME  DOB        PHONE
-----------  ----------  ----------  ---------  ------------
          3  Steve       White      16-MAR-71  800-555-1213
          4  Gail        Black                 800-555-1214
```

5	Doreen	Blue	20-MAY-70

ოპერატორების პრიორიტეტების გაგება

ერთ გამოსახულებაში „და“ და OR ოპერატორების კომბინირების დროს „და“ ოპერატორს გააჩნია პრიორიტეტი OR ოპერატორზე.

მაგალითი:

```
SELECT *
FROM customers
WHERE dob > '01-JAN-1970'
OR customer_id < 2
და phone LIKE '%1211';
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB	PHONE
1	John	Brown	01-JAN-65	800-555-1211
3	Steve	White	16-MAR-71	800-555-1213
5	Doreen	Blue	20-MAY-70	

როგორც ზემოთ იყო აღნიშნული, ოპერატორი და სრულდება პრიორიტეტულად:

```
dob > '01-JAN-1970' OR (customer_id < 2 და phone LIKE '%1211')
```

სტრიქონების სორტირება. ORDER BY ოპერატორის გამოყენება

მაგალითი:

```
SELECT *
FROM customers
ORDER BY last_name;
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB	PHONE
4	Gail	Black		800-555-1214
5	Doreen	Blue	20-MAY-70	
1	John	Brown	01-JAN-65	800-555-1211
2	Cynthia	Green	05-FEB-68	800-555-1212
3	Steve	White	16-MAR-71	800-555-1213

მაგალითი:

```
SELECT *
FROM customers
ORDER BY first_name ASC, last_name DESC;
```

```
CUSTOMER_ID FIRST_NAME LAST_NAME  DOB       PHONE
----------- ---------- ---------- --------- ------------
          2 Cynthia    Green      05-FEB-68 800-555-1212
          5 Doreen     Blue       20-MAY-70
          4 Gail       Black                800-555-1214
          1 John       Brown      01-JAN-65 800-555-1211
          3 Steve      White      16-MAR-71 800-555-1213
```

მაგალითი:

```
SELECT customer_id, first_name, last_name
FROM customers
ORDER BY 1;
```

```
CUSTOMER_ID FIRST_NAME LAST_NAME
----------- ---------- ----------
          1 John       Brown
          2 Cynthia    Green
          3 Steve      White
          4 Gail       Black
          5 Doreen     Blue
```

SELECT ოპერატორის შესრულება ორი ცხრილის გამოყენებით

მაგალითი:

```
SELECT name, product_type_id
FROM products
WHERE product_id = 1;
```

```
NAME                           PRODUCT_TYPE_ID
------------------------------ ---------------
Modern Science                               1
```

მაგალითი:

```
SELECT name
FROM product_types
WHERE product_type_id = 1;
```

```
NAME
----------
Book
```

მაგალითი:

```
SELECT products.name, product_types.name
FROM products, product_types
```

WHERE products.product_type_id =
product_types.product_type_id;

```
NAME                           NAME
------------------------------ ----------
Modern Science                 Book
Chemistry                      Book
Supernova                      Video
Tank War                       Video
Z Files                        Video
2412: The Return               Video
Space Force 9                  DVD
From Another Planet            DVD
Classical Music                CD
Pop 3                          CD
Creative Yell                  CD
```

დეკარტული ნამრავლი

მაგალითი: ცხრილები product_types და products შეიცავენ შესაბამისად 5 და 12 სტრიქონს.

```
SELECT pt.product_type_id, p.product_id
FROM product_types pt, products p;
```

```
PRODUCT_TYPE_ID PRODUCT_ID
--------------- ----------
              1          1
              2          1
              3          1
              4          1
              5          1
              1          2
              2          2
              3          2
              4          2
              5          2
              1          3
...
              5         11
              1         12
              2         12
              3         12
              4         12
              5         12
```

ამორჩეულია 60 სტრიქონი (5 * 12 = 60).

SELECT ოპერატორის შესრულება ორზე მეტი ცხრილის გამოყენებით

მაგალითი:

```
SELECT c.first_name, c.last_name, p.name AS PRODUCT,
pt.name AS TYPE
FROM customers c, purchases pr, products p, product_types pt
WHERE c.customer_id = pr.customer_id
და p.product_id = pr.product_id
და p.product_type_id = pt.product_type_id;
```

```
FIRST_NAME LAST_NAME  PRODUCT                        TYPE
---------- ---------- ------------------------------ ----------
John       Brown      Modern Science                 Book
Cynthia    Green      Modern Science                 Book
Steve      White      Modern Science                 Book
Gail       Black      Modern Science                 Book
John       Brown      Chemistry                      Book
Cynthia    Green      Chemistry                      Book
Steve      White      Chemistry                      Book
Gail       Black      Chemistry                      Book
Steve      White      Supernova                      Video
```

Join პირობები და Join ტიპები

განირჩევა *join conditions*-ის ორი ტიპი:

- Equijoins, სადაც გამოიყენება ტოლობის ოპერატორი (=).
- Non-equijoins, სადაც გამოიყენება უტოლობის ოპერატორები <, >, BETWEEN და ა.შ.

განირჩევა აგრეთვე join ოპერატორის სამი ტიპი:

- Inner joins.
- Outer joins.
- Self joins.

Non-equijoins

მაგალითი:

```
SELECT e.first_name, e.last_name, e.title, e.salary,
  sg.salary_grade_id
FROM employees e, salary_grades sg
```

WHERE e.salary BETWEEN sg.low_salary და sg.high_salary;

```
FIRST_NAME LAST_NAME  TITLE                SALARY SALARY_GRADE_ID
---------- ---------- ------------------ ---------- ---------------
Fred       Hobbs      Salesperson            150000               1
Susan      Jones      Salesperson            500000               2
Ron        Johnson    Sales Manager          600000               3
James      Smith      CEO                    800000               4
```

Outer Joins

მაგალითი:

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id = pt.product_type_id (+);
```

```
NAME                           NAME
------------------------------ ----------
Modern Science                 Book
Chemistry                      Book
Supernova                      Video
Tank War                       Video
Z Files                        Video
2412: The Return               Video
Space Force 9                  DVD
From Another Planet            DVD
Classical Music                CD
Pop 3                          CD
Creative Yell                  CD
My Front Line
```

Left და Right Outer Joins

Left Outer Join

მაგალითი:

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id = pt.product_type_id (+);
```

```
NAME                           NAME
------------------------------ ----------
Modern Science                 Book
Chemistry                      Book
Supernova                      Video
Tank War                       Video
Z Files                        Video
2412: The Return               Video
Space Force 9                  DVD
From Another Planet            DVD
Classical Music                CD
Pop 3                          CD
Creative Yell                  CD
My Front Line
```

Right Outer Join

მაგალითი:

```
SELECT *
FROM product_types;
```

```
PRODUCT_TYPE_ID NAME
--------------- ----------
              1 Book
              2 Video
              3 DVD
              4 CD
              5 Magazine
```

მაგალითი:

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id (+) = pt.product_type_id;
```

```
NAME                           NAME
------------------------------ ----------
Modern Science                 Book
Chemistry                      Book
Supernova                      Video
Tank War                       Video
Z Files                        Video
2412: The Return               Video
Space Force 9                  DVD
From Another Planet            DVD
Classical Music                CD
Pop 3                          CD
Creative Yell                  CD
                               Magazine
```

შეზღუდვები Outer Joins -ზე

მაგალითი:

```
SQL> SELECT p.name, pt.name
  2  FROM products p, product_types pt
  3  WHERE p.product_type_id (+) = pt.product_type_id (+);
WHERE p.product_type_id (+) = pt.product_type_id (+)
            *
ERROR at line 3:
```

მაგალითი:

```
SQL> SELECT p.name, pt.name
  2  FROM products p, product_types pt
  3  WHERE p.product_type_id (+) IN (1, 2, 3, 4);
WHERE p.product_type_id (+) IN (1, 2, 3, 4)
            *
ERROR at line 3:
```

მაგალითი:

```
SQL> SELECT p.name, pt.name
  2  FROM products p, product_types pt
  3  WHERE p.product_type_id (+) = pt.product_type_id
  4  OR p.product_type_id = 1;
WHERE p.product_type_id (+) = pt.product_type_id
            *
ERROR at line 3:
```

Joins შესრულება SQL/92 Syntax გამოყენებით:

Inner Joins შესრულება ორ ცხრილზე SQL/92 გამოყენებით

SQL/86 სტანდარტის მიხედვით inner join შესრულების მაგალითისათვის გვექნება:

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id = pt.product_type_id;
```

ხოლო SQL/92 -ში შეტანილია INNER JOIN და ON პირობებით:

```
SELECT p.name, pt.name
FROM products p INNER JOIN product_types pt
ON p.product_type_id = pt.product_type_id;
```

მაგალითი SQL/86 სტანდარტის მიხედვით

```
SELECT e.first_name, e.last_name, e.title, e.salary,
 sg.salary_grade_id
FROM employees e, salary_grades sg
WHERE e.salary BETWEEN sg.low_salary და sg.high_salary;
```

შემდეგი მაგალითი იყენებს SQL/92 სტანდარტს:

```
SELECT e.first_name, e.last_name, e.title, e.salary,
 sg.salary_grade_id
FROM employees e INNER JOIN salary_grades sg
ON e.salary BETWEEN sg.low_salary და sg.high_salary;
```

Joins გამარტივება USING გამოყენებით

მაგალითი:

```
SQL> SELECT p.name, pt.name, p.product_type_id
  2  FROM products p INNER JOIN product_types pt
  3  USING (product_type_id);
SELECT p.name, pt.name, p.product_type_id
               *
ERROR at line 1:
```

მაგალითი:

```
SQL> SELECT p.name, pt.name, p.product_type_id
  2  FROM products p INNER JOIN product_types pt
  3  USING (p.product_type_id);
USING (p.product_type_id)
       *
ERROR at line 3:
```

Inner Joins შესრულება ორზე მეტი ცხრილისათვის SQL/92 გამოყენებით

მაგალითი:

```
        SELECT c.first_name, c.last_name, p.name AS PRODUCT,
pt.name AS TYPE
        FROM customers c, purchases pr, products p, product_types pt
        WHERE c.customer_id = pr.customer_id
        და p.product_id = pr.product_id
        და p.product_type_id = pt.product_type_id;
```

მაგალითი (SQL/92 გამოყენებით):

```
        SELECT c.first_name, c.last_name, p.name AS PRODUCT,
pt.name AS TYPE
        FROM customers c INNER JOIN purchases pr
        USING (customer_id)
        INNER JOIN products p
        USING (product_id)
        INNER JOIN product_types pt
        USING (product_type_id);
```

Inner Joins შესრულება მრავალი სვეტისათვის SQL/92 გამოყენებით

```
SELECT ...
FROM ცხრილი1 INNER JOIN ცხრილი2
ON ცხრილი1.column1 = ცხრილი2.column1
და ცხრილი1.column2 = ცხრილი2.column2;
```

მოთხოვნა გამარტივდება USING პირობის გამოყენებით:

```
SELECT ...
FROM ცხრილი1 INNER JOIN ცხრილი2
```

USING (column1, column2);

Outer Joins შესრულება SQL/92 გამოყენებით

SQL/92 იყენებს სხვადასხვა სინტაქსისს.

Left Outer Joins შესრულება SQL/92 გამოყენებით

მაგალითი ადრეული ვერსიით:

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id = pt.product_type_id (+);
```

SQL/92 -ის გამოყენებით LEFT OUTER JOIN:

```
SELECT p.name, pt.name
FROM products p LEFT OUTER JOIN product_types pt
USING (product_type_id);
```

Right Outer Joins შესრულება SQL/92 გამოყენებით

მაგალითი ადრეული ვერსიით:

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id (+) = pt.product_type_id;
```

SQL/92 -ის გამოყენებით SQL/92 RIGHT OUTER JOIN:

```
SELECT p.name, pt.name
FROM products p RIGHT OUTER JOIN product_types pt
USING (product_type_id);
```

Full Outer Joins შესრულება SQL/92 გამოყენებით

მაგალითი:

```
SELECT p.name, pt.name
FROM products p FULL OUTER JOIN product_types pt
USING (product_type_id);
```

```
NAME                           NAME
------------------------------ ----------
Chemistry                      Book
Modern Science                 Book
2412: The Return               Video
Z Files                        Video
Tank War                       Video
Supernova                      Video
From Another Planet            DVD
Space Force 9                  DVD
Creative Yell                  CD
Pop 3                          CD
Classical Music                CD
My Front Line
                               Magazine
```

Self Joins შესრულება SQL/92 გამოყენებით

მაგალითი SQL/86 სტანდარტით:

```
SELECT w.last_name || ' works for ' || m.last_name
FROM employees w, employees m
WHERE w.manager_id = m.employee_id;
```

შემდეგი მაგალითი SQL/92 სტანდარტით INNER JOIN და ON :

```
SELECT w.last_name || ' works for ' || m.last_name
FROM employees w INNER JOIN employees m
ON w.manager_id = m.employee_id;
```

Cross Joins შესრულება SQL/92 გამოყენებით

მაგალითში დეკარტული ნამრავლის შესრულებისათვის SQL/92 სტანდარტით გამოიყენება CROSS JOIN:

```
SELECT *
FROM product_types CROSS JOIN products;
```

ქვემოთხოვნები

ა) ერთსტრიქონიანი ქვემოთხოვნები:

ქვემოთხოვნები პირობით **WHERE**

მაგალითი:

```
SELECT first_name, last_name
FROM customers
WHERE customer_id =
 (SELECT customer_id
  FROM customers
  WHERE last_name = 'Brown');
```

```
FIRST_NAME LAST_NAME
---------- ----------
John       Brown
```

მაგალითი:

```
SELECT product_id, name, price
FROM products
WHERE price >
 (SELECT AVG(price)
  FROM products);
```

PRODUCT_ID	NAME	PRICE
1	Modern Science	19.95
2	Chemistry	30
3	Supernova	25.99
5	Z Files	49.99

ქვემოთხოვნა პირობაში **HAVING**

მაგალითი:

```
SELECT product_type_id, AVG(price)
FROM products
GROUP BY product_type_id
HAVING AVG(price) <
 (SELECT MAX(AVG(price))
  FROM products
```

```
GROUP BY product_type_id);
```

```
PRODUCT_TYPE_ID    AVG(PRICE)
---------------    ----------
              1         24.975
              3          13.24
              4          13.99
                         13.49
```

ქვემოთხოვნა პირობაში **FROM**

მაგალითი:

```
SELECT product_id
FROM
 (SELECT product_id
  FROM products
  WHERE product_id < 3);
```

```
PRODUCT_ID
----------
         1
         2
```

მაგალითი:

```
SELECT prds.product_id, price, purchases_data.product_count
FROM products prds,
 (SELECT product_id, COUNT(product_id) product_count
  FROM purchases
  GROUP BY product_id) purchases_data
WHERE prds.product_id = purchases_data.product_id;
```

```
PRODUCT_ID     PRICE        PRODUCT_COUNT
----------     ----------   -------------
         1      19.95              4
         2      30                 4
         3      25.99              1
```

ბ) მრავალსტრიქონიანი მოთხოვნები:

IN გამოყენება მრავალსტრიქონიან ქვემოთხოვნაში

მაგალითი:

```
SELECT product_id, name
FROM products
WHERE product_id IN
```

```
 (SELECT product_id
  FROM products
  WHERE name LIKE '%e%');

PRODUCT_ID       NAME
---------- -     ------------------
         1       Modern Science
         2       Chemistry
         3       Supernova
         5       Z Files
         6       2412: The Return
         7       Space Force 9
         8         From Another Planet
        11         Creative Yell
        12         My Front Line
```

მაგალითი:

```
SELECT product_id, name
FROM products
WHERE product_id NOT IN
 (SELECT product_id
  FROM purchases);

PRODUCT_ID          NAME
----------          ------------------
         4          Tank War
         5          Z Files
         6          2412: The Return
         7          Space Force 9
         8          From Another Planet
         9          Classical Music
        10          Pop 3
        11          Creative Yell
        12          My Front Line
```

ANY გამოყენება მრავალსტრიქონიან ქვემოთხოვნაში

მაგალითი:

```
SELECT employee_id, last_name
FROM employees
WHERE salary < ANY
 (SELECT low_salary
  FROM salary_grades);
```

```
EMPLOYEE_ID  LAST_NAME
-----------  ----------
          2  Johnson
          3  Hobbs
          4  Jones
```

ALL გამოყენება მრავალსტრიქონიან ქვემოთხოვნაში

მაგალითი:

```
SELECT employee_id, last_name
FROM employees
WHERE salary > ALL
  (SELECT high_salary
   FROM salary_grades);
```

სტრიქონები არ არის შერჩეული.

გ) მრავალსვეტიანი ქვემოთხოვნები

მაგალითი:

```
SELECT product_id, product_type_id, name, price
FROM products
WHERE (product_type_id, price) IN
 (SELECT product_type_id, MIN(price)
  FROM products
  GROUP BY product_type_id);
```

```
PRODUCT_ID  PRODUCT_TYPE_ID  NAME                 PRICE
----------  ---------------  -------------------- ----------
         1                1  Modern Science        19.95
         4                2  Tank War              13.95
         8                3  From Another Planet   12.99
         9                4  Classical Music       10.99
```

დ) კორელირებული ქვემოთხოვნები

კორელირებული ქვემოთხოვნა მიმართავს სხვა მოთხოვნის ერთ ან მეტ სვეტს. მაგალითი:

```
SELECT product_id, product_type_id, name, price
FROM products outer
WHERE price >
  (SELECT AVG(price)
```

```
FROM products inner
WHERE inner.product_type_id = outer.product_type_id);
```

PRODUCT_ID	PRODUCT_TYPE_ID	NAME	PRICE
2	1	Chemistry	30
5	2	Z Files	49.99
7	3	Space Force 9	13.49
10	4	Pop 3	15.99
11	4	Creative Yell	14.99

ე) ჩასმული ქვემოთხოვნები

ქვემოთხოვნები შესაძლოა ჩავსვათ სხვა ქვემოთხოვნათა შიგნით. მაგალითი:

```
SELECT product_type_id, AVG(price)
FROM products
GROUP BY product_type_id
HAVING AVG(price) <
 (SELECT MAX(AVG(price))
  FROM products
  WHERE product_type_id IN
   (SELECT product_id
    FROM purchases
    WHERE quantity > 1)
  GROUP BY product_type_id);
```

PRODUCT_TYPE_ID	AVG(PRICE)
1	24.975
3	13.24
4	13.99
	13.49

UPDATE და DELETE ბრძანებები ქვემოთხოვნების შემადგენლობაში

ა) UPDATE ბრძანება ქვემოთხოვნების შემადგენლობაში

მაგალითი:

```
UPDATE employees
```

```
SET salary =
  (SELECT AVG(high_salary)
   FROM salary_grades)
WHERE employee_id = 4;
```

1 ერთი სტრიქონი შეიცვალა.

UPDATE შესრულების შემდეგ, ROLLBACK შესრულება აღადგენს ცვლილებას

ბ) DELETE ბრძანება ქვემოთხოვნების შემადგენლობაში

მაგალითი:

```
DELETE FROM employees
WHERE salary >
  (SELECT AVG(high_salary)
   FROM salary_grades);
```

1 სტრიქონი წაიშალა.

გაფართოებული მოთხოვნები:

Set ოპერატორების გამოყენება

Set ოპერატორები ორი ან მეტი მოთხოვნის სტრიქონების კომბინირების საშუალებას იძლევა.

ოპერატორები

Operator	Description
UNION ALL	აბრუნებს ყველა ჩანაწერს, განმეორებული ჩანაწერების ჩათვლით.
UNION	აბრუნებს ყველა არაგანმეორებულ ჩანაწერს.
INTERSECT	აბრუნებს ჩანაწერებს ორივე მოთხოვნის თანაკვეთის შესაბამისად.
MINUS	აბრუნებს ჩანაწერებს ორივე მოთხოვნის სხვაობის შესაბამისად.

მაგალითები:

შევქმნათ შემდეგი ცხრილები

```
CREATE ცხრილი products (
  product_id INTEGER
    CONSTRAINT products_pk PRIMARY KEY,
```

```
  product_type_id INTEGER
    CONSTRAINT products_fk_product_types
    REFERENCES product_types(product_type_id),
  name VARCHAR2(30) NOT NULL,
  description VARCHAR2(50),
  price NUMBER(5, 2)
);

CREATE ცხრილი more_products (
  prd_id INTEGER
    CONSTRAINT more_products_pk PRIMARY KEY,
  prd_type_id INTEGER
    CONSTRAINT more_products_fk_product_types
    REFERENCES product_types(product_type_id),
  name VARCHAR2(30) NOT NULL,
  available CHAR(1)
);
```

შევიტანოთ მონაცემები შესაბამის ცხრილებში:

PRODUCT_ID	PRODUCT_TYPE_ID	NAME
1	1	Modern Science
2	1	Chemistry
3	2	Supernova
4	2	Tank War
5	2	Z Files
6	2	2412: The Return
7	3	Space Force 9
8	3	From Another Planet
9	4	Classical Music
10	4	Pop 3
11	4	Creative Yell
12		My Front Line

PRD_ID	PRD_TYPE_ID	NAME
1	1	Modern Science
2	1	Chemistry
3		Supernova
4	2	Lunar Lდsing
5	2	Submarine

UNION ALL ოპერატორის გამოყენება

```
SELECT product_id, product_type_id, name
FROM products
UNION ALL
SELECT prd_id, prd_type_id, name
FROM more_products;
```

PRODUCT_ID	PRODUCT_TYPE_ID	NAME
1	1	Modern Science
2	1	Chemistry
3	2	Supernova
4	2	Tank War
5	2	Z Files
6	2	2412: The Return
7	3	Space Force 9
8	3	From Another Planet
9	4	Classical Music
10	4	Pop 3
11	4	Creative Yell
12		My Front Line
1	1	Modern Science
2	1	Chemistry
3		Supernova
4	2	Lunar Landing
5	2	Submarine

17 rows selected.

ORDER BY პირობის შემთხვევაში ხდება დალაგება:

```
SELECT product_id, product_type_id, name
FROM products
UNION ALL
SELECT prd_id, prd_type_id, name
FROM more_products
ORDER BY 1;
```

```
PRODUCT_ID PRODUCT_TYPE_ID NAME
---------- --------------- --------------------
         1               1 Modern Science
         1               1 Modern Science
         2               1 Chemistry
         2               1 Chemistry
         3               2 Supernova
         3                 Supernova
         4               2 Tank War
         4               2 Lunar Landing
         5               2 Z Files
         5               2 Submarine
         6               2 2412: The Return
         7               3 Space Force 9
         8               3 From Another Planet
         9               4 Classical Music
        10               4 Pop 3
        11               4 Creative Yell
        12                 My Front Line
```

UNION ოპერატორის გამოყენება

ეს ოპერატორი მოთხოვნის მეშვეობით აბრუნებს ყველა არადუბლირებულ სტრიქონს.

```
SELECT product_id, product_type_id, name
FROM products
UNION
SELECT prd_id, prd_type_id, name
FROM more_products;
```

```
PRODUCT_ID PRODUCT_TYPE_ID NAME
---------- --------------- --------------------
         1               1 Modern Science
         2               1 Chemistry
         3               2 Supernova
         3                 Supernova
         4               2 Lunar Landing
         4               2 Tank War
         5               2 Submarine
         5               2 Z Files
         6               2 2412: The Return
         7               3 Space Force 9
         8               3 From Another Planet
         9               4 Classical Music
        10               4 Pop 3
        11               4 Creative Yell
        12                 My Front Line
```

INTERSECT ოპერატორის გამოყენება

INTERSECT ოპერატორი აბრუნებს სტრიქონებს, რომლებიც არიან ორივე მოთხოვნაში.

```
SELECT product_id, product_type_id, name
FROM products
INTERSECT
SELECT prd_id, prd_type_id, name
FROM more_products;
```

```
PRODUCT_ID PRODUCT_TYPE_ID NAME
---------- --------------- --------------
         1               1 Modern Science
         2               1 Chemistry
```

MINUS ოპერატორის გამოყენება

MINUS ოპერატორი ასრულებს სხვაობის ოპერაციას:

```
SELECT product_id, product_type_id, name
FROM products
MINUS
SELECT prd_id, prd_type_id, name
FROM more_products;
```

```
PRODUCT_ID PRODUCT_TYPE_ID NAME
---------- --------------- -------------------
         3               2 Supernova
         4               2 Tank War
         5               2 Z Files
         6               2 2412: The Return
         7               3 Space Force 9
         8               3 From Another Planet
         9               4 Classical Music
        10               4 Pop 3
        11               4 Creative Yell
        12                 My Front Line
```

Set ოპერატორების კომბინირება

მაგალითი:

ჩვენ გამოვიყენებთ product_changes ცხრილს, რომელიც შეიქმნება store_schema.sql სკრიპტით:

```
CREATE ცხრილი product_changes (
  product_id INTEGER
    CONSTRAINT prod_changes_pk PRIMARY KEY,
  product_type_id INTEGER
    CONSTRAINT prod_changes_fk_product_types
    REFERENCES product_types(product_type_id),
  name VARCHAR2(30) NOT NULL,
  description VARCHAR2(50),
  price NUMBER(5, 2)
);
```

```
PRODUCT_ID PRODUCT_TYPE_ID NAME
---------- --------------- --------------
         1               1 Modern Science
         2               1 New Chemistry
         3               1 Supernova
        13               2 Lunar Landing
        14               2 Submarine
        15               2 Airplane
```

მაგალითი:

```
(SELECT product_id, product_type_id, name
FROM products
UNION
SELECT prd_id, prd_type_id, name
FROM more_products)
INTERSECT
SELECT product_id, product_type_id, name
FROM product_changes;
```

```
PRODUCT_ID PRODUCT_TYPE_ID NAME
---------- --------------- --------------
         1               1 Modern Science
```

მაგალითი:

```
SELECT product_id, product_type_id, name
FROM products
UNION
(SELECT prd_id, prd_type_id, name
FROM more_products
INTERSECT
SELECT product_id, product_type_id, name
FROM product_changes);
```

PRODUCT_ID	PRODUCT_TYPE_ID	NAME
1	1	Modern Science
2	1	Chemistry
3	2	Supernova
4	2	Tank War
5	2	Z Files
6	2	2412: The Return
7	3	Space Force 9
8	3	From Another Planet
9	4	Classical Music
10	4	Pop 3
11	4	Creative Yell
12		My Front Line

CASE გამოსახულების გამოყენება

CASE გამოსახულება ასრულებს if-then-else ლოგიკას SQL -ში PL/SQL-ის გამოყენების გარეშე. მისი სინტაქსი შემდეგია:

```
CASE search_expression
  WHEN expression1 THEN result1
  WHEN expression2 THEN result2
  ...
  WHEN expressionN THEN resultN
  ELSE default_result
END
```

მაგალითი:

```
SELECT product_id, product_type_id,
 CASE product_type_id
  WHEN 1 THEN 'Book'
```

```
  WHEN 2 THEN 'Video'
  WHEN 3 THEN 'DVD'
  WHEN 4 THEN 'CD'
  ELSE 'Magazine'
 END
FROM products;
```

PRODUCT_ID	PRODUCT_TYPE_ID	CASEPROD
1	1	Book
2	1	Book
3	2	Video
4	2	Video
5	2	Video
6	2	Video
7	3	DVD
8	3	DVD
9	4	CD
10	4	CD
11	4	CD
12		Magazine

ძებნა CASE გამოსახულებაში

მისი სინტაქსი შემდეგია:

```
CASE
  WHEN condition1 THEN result1
  WHEN condition2 THEN result2
  ...
  WHEN conditionN THEN resultN
  ELSE default_result
END
```

მაგალითი:

```
SELECT product_id, product_type_id,
 CASE
  WHEN product_type_id = 1 THEN 'Book'
  WHEN product_type_id = 2 THEN 'Video'
  WHEN product_type_id = 3 THEN 'DVD'
  WHEN product_type_id = 4 THEN 'CD'
  ELSE 'Magazine'
 END
FROM products;
```

PRODUCT_ID	PRODUCT_TYPE_ID	CASEPROD
1	1	Book
2	1	Book
3	2	Video
4	2	Video
5	2	Video
6	2	Video
7	3	DVD
8	3	DVD
9	4	CD
10	4	CD
11	4	CD
12		Magazine

მაგალითი:

```
SELECT product_id, price,
 CASE
  WHEN price > 15 THEN 'Expensive'
  ELSE 'Cheap'
 END
FROM products;
```

PRODUCT_ID	PRICE	CASEWHENP
1	19.95	Expensive
2	30	Expensive
3	25.99	Expensive
4	13.95	Cheap
5	49.99	Expensive
6	14.95	Cheap
7	13.49	Cheap
8	12.99	Cheap
9	10.99	Cheap
10	15.99	Expensive
11	14.99	Cheap
12	13.49	Cheap

იერარქიული მოთხოვნები

მოთხოვნაში მონაცემთა შორის იერარქიული დამოკიდებულების ასახვისათვის გამოიყენება ე.წ. იერარქიული მოთხოვნები.

მაგალითი:

ცხრილი სახელწოდებით more_employees არის შექმნილი store_schema.sql სკრიპტის მიხედვით:

```
CREATE ცხრილი more_employees (
  employee_id INTEGER
    CONSTRAINT more_employees_pk PRIMARY KEY,
  manager_id INTEGER
    CONSTRAINT more_empl_fk_fk_more_empl
    REFERENCES more_employees(employee_id),
  first_name VARCHAR2(10) NOT NULL,
  last_name VARCHAR2(10) NOT NULL,
  title VARCHAR2(20),
  salary NUMBER(6, 0)
);
```

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME	LAST_NAME	TITLE	SALARY
1		James	Smith	CEO	800000
2	1	Ron	Johnson	Sales Manager	600000
3	2	Fred	Hobbs	Sales Person	200000
4	1	Susan	Jones	Support Manager	500000
5	2	Rob	Green	Sales Person	40000
6	4	Jane	Brown	Support Person	45000
7	4	John	Grey	Support Manager	30000
8	7	Jean	Blue	Support Person	29000
9	6	Henry	Heyson	Support Person	30000
10	1	Kevin	Black	Ops Manager	100000
11	10	Keith	Long	Ops Person	50000
12	10	Frank	Howard	Ops Person	45000
13	10	Doreen	Penn	Ops Person	47000

Employee-ს მონაცემთა რელაციებს გააჩნიათ ხისებრი სტუქტურა.

CONNECT BY და START WITH პირობები

იერარქიული მოთხოვნების ფორმირებისათვის გამოიყენება CONNECT BY და START WITH პირობები შემდეგი სინტაქსისით:

SELECT [LEVEL], *column*, *expression*, ...
FROM *ცხრილი*
[WHERE *where_clause*]
[[START WITH *start_condition*] [CONNECT BY PRIOR *prior_condition*]];

მაგალითი:

```
SELECT employee_id, manager_id, first_name, last_name
FROM more_employees
START WITH employee_id = 1
CONNECT BY PRIOR employee_id = manager_id;
```

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME	LAST_NAME
1		James	Smith
2	1	Ron	Johnson
3	2	Fred	Hobbs
5	2	Rob	Green
4	1	Susan	Jones
6	4	Jane	Brown
9	6	Henry	Heyson
7	4	John	Grey
8	7	Jean	Blue
10	1	Kevin	Black
11	10	Keith	Long
12	10	Frank	Howard
13	10	Doreen	Penn

LEVEL ფსევდო-სვეტის გამოყენება

მაგალითი:

```
SELECT LEVEL, employee_id, manager_id, first_name, last_name
FROM more_employees
START WITH employee_id = 1
CONNECT BY PRIOR employee_id = manager_id
ORDER BY LEVEL;
```

LEVEL	EMPLOYEE_ID	MANAGER_ID	FIRST_NAME	LAST_NAME
1	1		James	Smith
2	2	1	Ron	Johnson
2	4	1	Susan	Jones
2	10	1	Kevin	Black
3	3	2	Fred	Hobbs
3	7	4	John	Grey
3	12	10	Frank	Howard
3	13	10	Doreen	Penn
3	11	10	Keith	Long
3	5	2	Rob	Green
3	6	4	Jane	Brown
4	9	6	Henry	Heyson
4	8	7	Jean	Blue

შემდეგ მაგალითში მოთხოვნა იყენებს COUNT() ფუნქციას და LEVEL დონის რიცხვს ხეში:

```
SELECT COUNT(DISTINCT LEVEL)
FROM more_employees
START WITH employee_id = 1
CONNECT BY PRIOR employee_id = manager_id;
```

```
COUNT(DISTINCTLEVEL)
--------------------
                   4
```

შედეგების ფორმატირება იერარქიული მოთხოვნიდან

მაგალითი:

```
SET PAGESIZE 999
COLUMN employee FORMAT A25
SELECT LEVEL,
 LPAD(' ', 2 * LEVEL - 1) || first_name || ' ' ||
 last_name AS employee
FROM more_employees
START WITH employee_id = 1
CONNECT BY PRIOR employee_id = manager_id;
```

```
     LEVEL EMPLOYEE
---------- ------------------
         1 James Smith
         2   Ron Johnson
         3     Fred Hobbs
         3     Rob Green
         2   Susan Jones
         3     Jane Brown
         4       Henry Heyson
         3     John Grey
         4       Jean Blue
         2   Kevin Black
         3     Keith Long
         3     Frank Howard
         3     Doreen Penn
```

Node-დან Root-ის დაწყება

მაგალითი:

```
SELECT LEVEL,
 LPAD(' ', 2 * LEVEL - 1) || first_name || ' ' ||
 last_name AS employee
FROM more_employees
START WITH last_name = 'Jones'
CONNECT BY PRIOR employee_id = manager_id;
```

```
     LEVEL EMPLOYEE
---------- -----------------
         1  Susan Jones
         2    Jane Brown
         3      Henry Heyson
         2    John Grey
         3      Jean Blue
```

ქვემოთხოვნის გამოყენება START WITH პირობაში

მაგალითი:

```
SELECT LEVEL,
 LPAD(' ', 2 * LEVEL - 1) || first_name || ' ' ||
 last_name AS employee
FROM more_employees
START WITH employee_id = (
```

```
  SELECT employee_id
  FROM more_employees
  WHERE first_name = 'Kevin'
  და last_name = 'Black'
)
CONNECT BY PRIOR employee_id = manager_id;
```

```
     LEVEL EMPLOYEE
---------- ---------------
         1  Kevin Black
         2    Keith Long
         2    Frank Howard
         2    Doreen Penn
```

სხვა პირობების ჩართვა იერარქიულ მოთხოვნაში

მაგალითი:

```
SELECT LEVEL,
 LPAD(' ', 2 * LEVEL - 1) || first_name || ' ' ||
 last_name AS employee, salary
FROM more_employees
WHERE salary <= 50000
START WITH employee_id = 1
CONNECT BY PRIOR employee_id = manager_id;
```

```
     LEVEL EMPLOYEE                       SALARY
---------- ------------------------- ----------
         3      Rob Green                  40000
         3      Jane Brown                 45000
         4        Henry Heyson             30000
         3      John Grey                  30000
         4        Jean Blue                29000
         3      Keith Long                 50000
         3      Frank Howard               45000
         3      Doreen Penn                47000
```

გაფართოებული GROUP BY პირობის გამოყენება

პრაქტიკული თვალსაჩინოებისათვის შევქმნათ შემდეგი ცხრილები.

ცხრილი divisions:

```
CREATE ცხრილი divisions (
  division_id CHAR(3)
    CONSTRAINT divisions_pk PRIMARY KEY,
  name VARCHAR2(15) NOT NULL
);
```

შემდეგი სტრიქონებით:

```
DIV NAME
--- ----------
SAL Sales
OPE Operations
SUP Support
BUS Business
```

ცხრილი jobs:

```
CREATE ცხრილი jobs (
  job_id CHAR(3)
    CONSTRAINT jobs_pk PRIMARY KEY,
  name VARCHAR2(20) NOT NULL
);
```

შემდეგი სტრიქონებით:

```
JOB NAME
--- ------------
WOR Worker
MGR Manager
ENG Engineer
TEC Technologist
PRE President
```

ცხრილი employees2:

```
CREATE ცხრილი employees2 (
  employee_id INTEGER
    CONSTRAINT employees2_pk PRIMARY KEY,
  division_id CHAR(3)
    CONSTRAINT employees2_fk_divisions
    REFERENCES divisions(division_id),
```

```
  job_id CHAR(3) REFERENCES jobs(job_id),
  first_name VARCHAR2(10) NOT NULL,
  last_name VARCHAR2(10) NOT NULL,
  salary NUMBER(6, 0)
);
```

შემდეგი სტრიქონებით:

```
EMPLOYEE_ID DIV JOB FIRST_NAME LAST_NAME      SALARY
----------- --- --- ---------- ---------- ----------
          1 BUS PRE James      Smith          800000
          2 SAL MGR Ron        Johnson        350000
          3 SAL WOR Fred       Hobbs          140000
          4 SUP MGR Susan      Jones          200000
          5 SAL WOR Rob        Green          350000
```

ROLLUP პირობის გამოყენება

ROLLUP წესი აფართოებს GROUP BY, რომელიც აჯგუფებს საერთო მნიშვნელობის მქონე ჩანაწერებს ბლოკებში.

მაგალითი:

```
SELECT division_id, SUM(salary)
FROM employees2
GROUP BY division_id;
```

```
DIV    SUM(SALARY)
---     -----------
BUS     1610000
OPE     1320000
SAL     4936000
SUP     1015000
```

ერთი სვეტის შემთხვევაში ROLLUP-ზე გადასვლით:

```
SELECT division_id, SUM(salary)
FROM employees2
GROUP BY ROLLUP(division_id);
```

```
DIV       SUM(SALARY)
---       -----------
BUS       1610000
OPE       1320000
SAL       4936000
SUP       1015000
          8881000
```

მრავალი სვეტის შემთხვევაში **ROLLUP**-ზე გადასვლით:

```
SELECT division_id, job_id, SUM(salary)
FROM employees2
GROUP BY ROLLUP(division_id, job_id);
```

```
DIV JOB SUM(SALARY)
--- --- -----------
BUS MGR      530000
BUS PRE      800000
BUS WOR      280000
BUS         1610000
OPE ENG      245000
OPE MGR      805000
OPE WOR      270000
OPE         1320000
SAL MGR     4446000
SAL WOR      490000
SAL         4936000
SUP MGR      465000
SUP TEC      115000
SUP WOR      435000
SUP         1015000
            8881000
```

სვეტების პოზიციათა ცვლილების შემთხვევაში:

```
SELECT job_id, division_id, SUM(salary)
FROM employees2
GROUP BY ROLLUP(job_id, division_id);
```

```
JOB DIV SUM(SALARY)
--- --- -----------
ENG OPE      245000
ENG          245000
MGR BUS      530000
MGR OPE      805000
MGR SAL     4446000
MGR SUP      465000
MGR         6246000
PRE BUS      800000
PRE          800000
TEC SUP      115000
TEC          115000
WOR BUS      280000
WOR OPE      270000
WOR SAL      490000
WOR SUP      435000
WOR         1475000
            8881000
```

ლაბორატორიული სამუშაო 4

მუშაობა Object Browser - ში

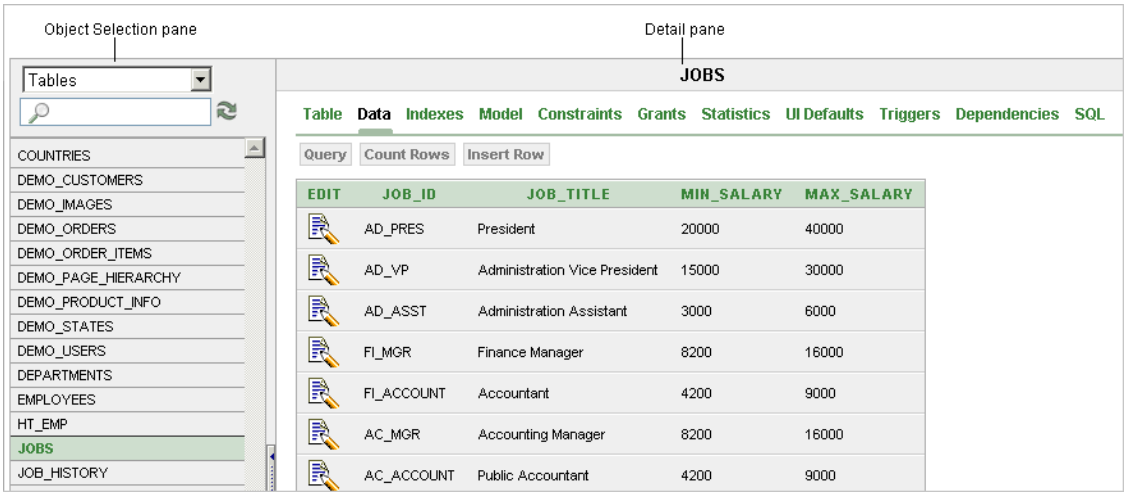
სამუშაოს მიზანი:

1. Object Browser -ში ახალი ობიექტის შექმნის შესწავლა.
2. Object Browser -ში ცხრილების მართვის შესწავლა.

Object Browser -ში ახალი ობიექტის შექმნის შესწავლა

Object Browser -ის გვერდი შედგება ორი სექციისაგან:

- **Object Selection pane** .
- **Detail pane**.



ნახ.4.1

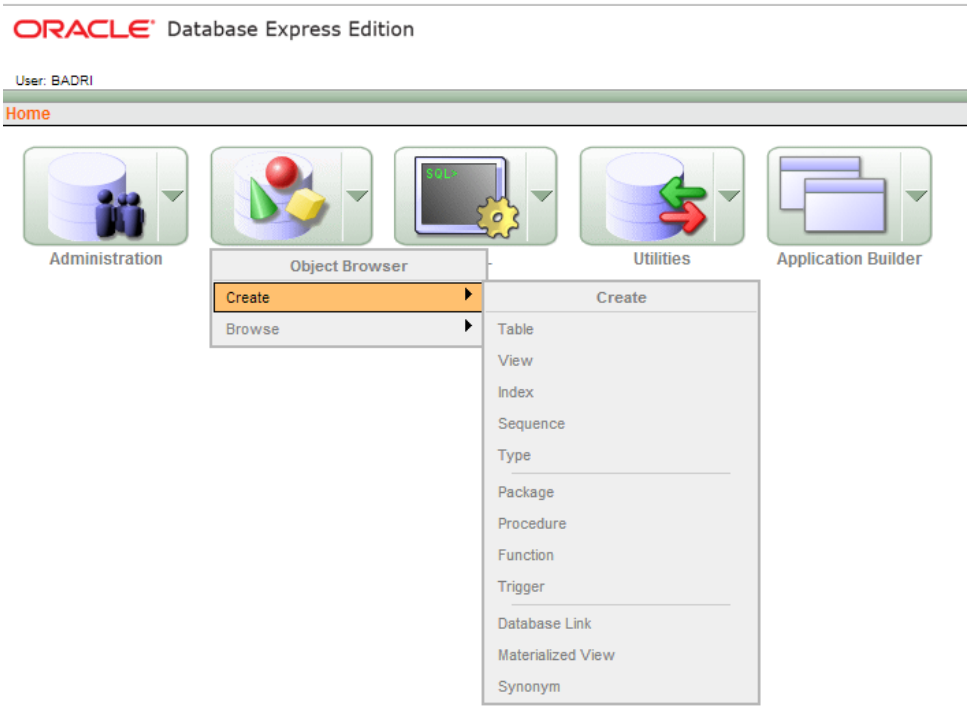
ახალი ობიექტის შექმნისათვის:

1. **Database Home Page** -ზე ვაწკაპუნებთ იკონაზე **Object Browser**. Object Browser გაიხსნება.
2. ვაწკაპუნებთ **Create**, რომელიც განთავსებულია Detail pane -ის მარჯვენა ზედა კუთხეში.
3. ობიექტების ტიპების სიიდან ვირჩევთ სასურველ ტიპს, რომლის შექმნაც გვინდა.
4. მივყვებით ეკრანზე მითითებულ ინსტრუქციებს.

Object Browser -ის წვდომა:

1. Database Home Page -ის მთავარ გვერდზე Log in.
2. Object Browser -ის ნახვისათვის ჩვენ შეგვიძლია:

- ვაწკაპუნებთ იკონაზე **Object Browser**.
- ვაწკაპუნებთ იკონის მარჯვენა მხარეს. ჩნდება ჩამოშლადი მენიუ.



ნახ.4.2

ახალი ობიექტის შექმნა

ახალი ობიექტის შექმნისათვის:

1. Database Home Page-ზე ვაწკაპუნებთ იკონაზე**Object Browser**. ჩნდება Object Browser.

2. ვაწკაპუნებთ **Create**, რომელიც განთავსებულია Detail pane -ის მარჯვენა ზედა კუთხეში.

3. ობიექტების ტიპების სიიდან ვირჩევთ სასურველ ტიპს, რომლის შექმნაც გვინდა.

4. მივყვებით ეკრანზე მითითებულ ინსტრუქციებს.

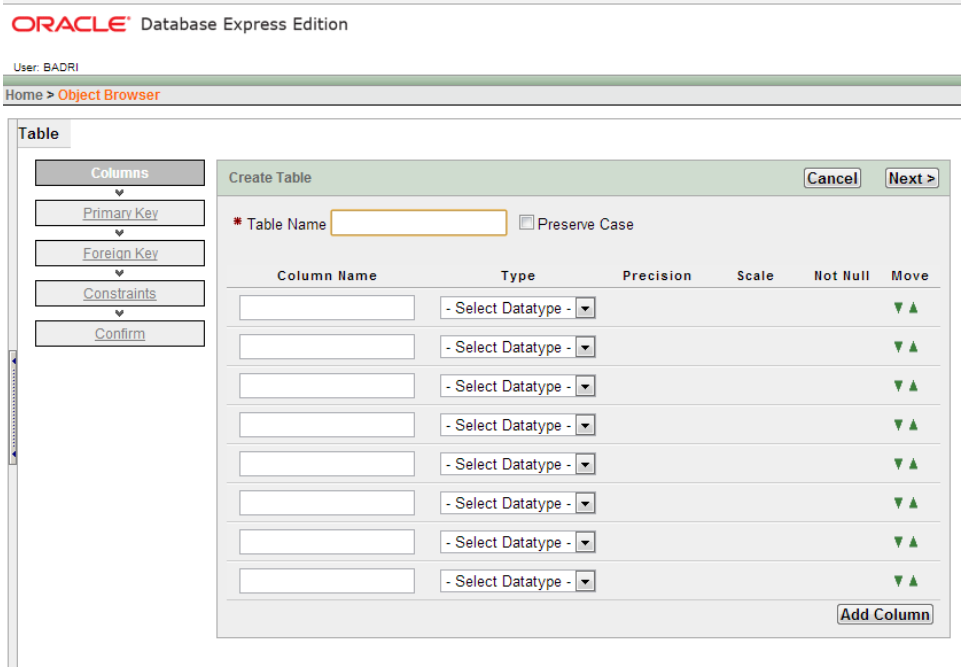
ცხრილების მართვა

ახალი ცხრილის შექმნა

ახალი ცხრილის შექმნისათვის:

1. Database Home Page -ზე ვაწკაპუნებთ იკონაზე **Object Browser**. ჩნდება Object Browser.
2. ვაწკაპუნებთ **Create**.
3. ობიექტების ტიპების სიიდან ავირჩიოთ **ცხრილი**.
4. შეგვაქვს ცხრილის სახელი.

აქ უნდა დავიცვათ შემდეგი წესები: ცხრილის სახელი არ უნდა შეიცავდეს სიცარიელეს ან არ უნდა იწყებოდეს ციფრით და არც ხაზგასმის სიმბოლოთი.



ნახ.4.3

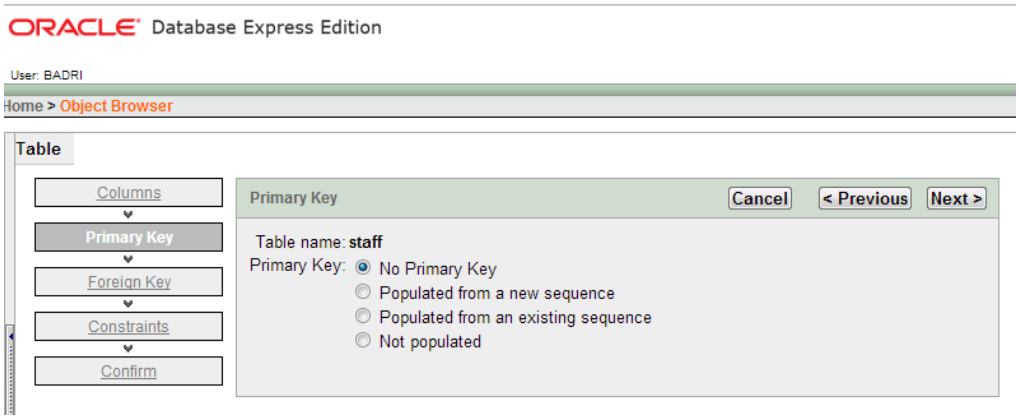
5. ცხრილი Name ველში სახელის შეტანის შემდეგ ვაწკაპუნებთ **Preserve Case**.
6. ყოველი სვეტისათვის შეგვაქვს შემდეგი ინფორმაცია:
   - სვეტის სახელი.
   - სვეტის ტიპი, მათ შორის NUMBER, VARCHAR2, DATE, TIMESTAMP, CHAR, CLOB, BLOB, NVARCHAR2, BINARY_FLOAT, და BINARY_ორჯერ

- შესაძლო დამატებითი ინფორმაცია:
  - Precision
  - Scale
- საჭიროების შემთხვევაში ვირჩევთ **Not Null**.

სვეტების თანამიმდევრობის შეცვლისათვის ვაწკაპუნებთ **Up** და **Down** სვეტში Move. სვეტის დამატებისათვის ვაწკაპუნებთ **Add Column**.

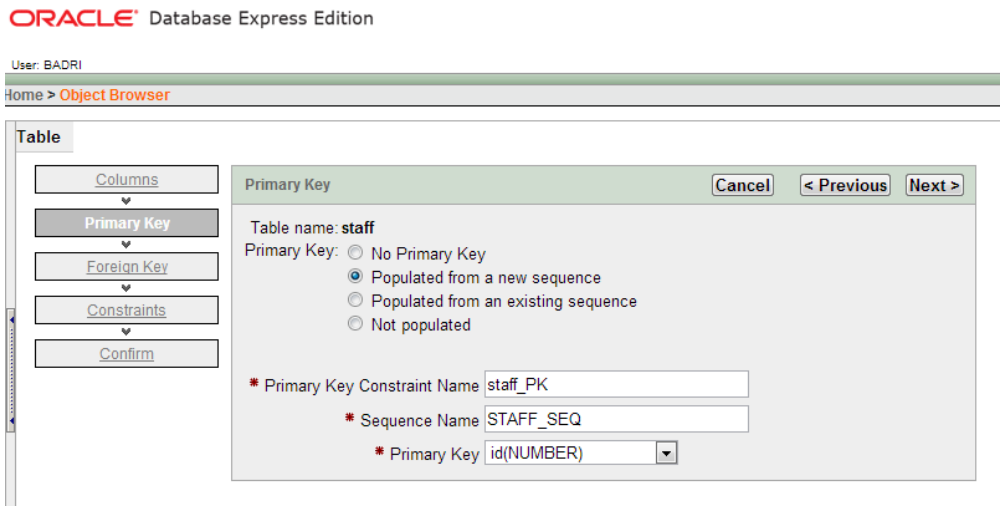
- ვაწკაპუნებთ **Next**.

7. პირველადი გასაღებისათვის ვირჩევთ ღილაკს Primary Key. ვაწკაპუნებთ **Next**:



ნახ.4.4

- **No Primary Key** - იქმნება არაგასაღებური ველი.



ნახ.4.5

– **Populated from a new sequence** - ქმნის პირველად გასაღებს, ახალ ტრიგერს და ახალ მიმდევრობას (sequence). პირველადი გასაღები შეიძლება მხოლოდ ერთი სვეტი იყოს.
– **Populated from an existing sequence** - ქმნის პირველად გასაღებს და ახალ ტრიგერს.
– **Not populated** - განსაზღვრავს პირველად გასაღებს ავტომატური მიმდევრობის მნიშვნელობის გარეშე. (ამ შემთხვევაში შესაძლებელია ერთზე მეტი სვეტი განისაზღვროს როგორც პირველადი გასაღები).

Next -ით ვამატებთ გარე გასაღებს. ამისათვის:

Name - ში შეგვაქვს გარე გასაღების სახელი.

Select Key Column(s) - ში ვირჩევთ სვეტებს, რომლებიც არის გარე გასაღების ნაწილი. ერთხელ ვირჩევთ, ვაწკაპუნებთ **Add** იკონაზე Key Column(s) -ში გადასატანად.

References Tables - ში ვირჩევთ ცხრილს, რომელიც უნდა იყოს დაკავშირებული გარე გასაღებთან. ერთხელ ვირჩევთ, ვაწკაპუნებთ **Add** იკონაზე შერჩეული სვეტების Referenced Column(s) -ში გადასატანად.

ვირჩევთ შემდეგიდან ერთ-ერთს:

– **Disallow Delete** - მოცემულ ცხრილში ბლოკავს ჩანაწერების წაშლას.

– **Cascade Delete** - კასკადურად წაშლის დამოკიდებულ ჩანაწერებს მოცემული ცხრილიდან, როცა „მშობელ“ ცხრილში შესაბამისი ჩანაწერი იქნება წაშლილი.

– **Set to Null on Delete** - მოცემულ ცხრილში ანულებს გარე გასაღების მნიშვნელობებს, როცა „მშობელ“ ცხრილში შესაბამისი ჩანაწერი იქნება წაშლილი.

ვაწკაპუნებთ **Add**.

ვაწკაპუნებთ **Next**.

– **Next** -ით ვამატებთ შეზღუდვებს. ამისათვის:

ა. განვსაზღვრავთ შეზღუდვის ტიპს (Check ან Unique).

ბ. შეგვაქვს შეზღუდვა საჭირო ველში.

გ. ვაწკაპუნებთ **Add**.

დ. ვაწკაპუნებთ **Finish**.

ჩნდება დადასტურების გვერდი. SQL ნახვისათვის ვაწკაპუნებთ **SQL Syntax.**

– ვაწკაპუნებთ **Create**.

ცხრილის რედაქტირება (Browsing)

ცხრილის ნახვა:

1. Database Home Page -ზე ვაწკაპუნებთ იკონაზე **Object Browser**.

ჩნდება Object Browser.

2. ობიექტების ტიპების სიიდან ვირჩევთ **ცხრილი**.
3. Object Selection pane -დან ვირჩევთ ცხრილს.

ჩნდება ცხრილის აღწერა. ცხრილის რედაქტირებისათვის:

1. Database Home Page -ზე ვაწკაპუნებთ იკონაზე **Object Browser**.

ჩნდება Object Browser.

2. ობიექტების ტიპების სიიდან ვირჩევთ **ცხრილი**.
3. Object Selection pane -დან ვირჩევთ ცხრილს.

ჩნდება ცხრილის აღწერა.

4. რედაქტირების მიზნიდან გამომდინარე ვაწკაპუნებთ შესაბამის ღილაკზე: Add Column, Modify Column, Rename Column, Drop Column, Rename, Copy, Drop, Truncate, Create Lookup ცხრილი.

ცხრილის წაშლა

ცხრილის წაშლისათვის:

1. Database Home Page -ზე ვაწკაპუნებთ იკონაზე **Object Browser**.

ჩნდება Object Browser.

2. ობიექტების ტიპების სიიდან ვირჩევთ **ცხრილს**.
3. Object Selection pane -დან ვირჩევთ ცხრილს.

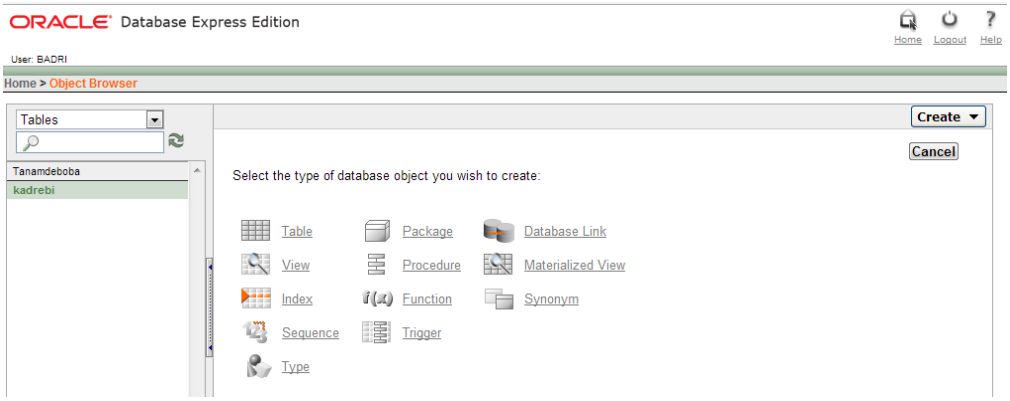
ჩნდება ცხრილის აღწერა.

4. ვაწკაპუნებთ **Drop**.

მაგალითი

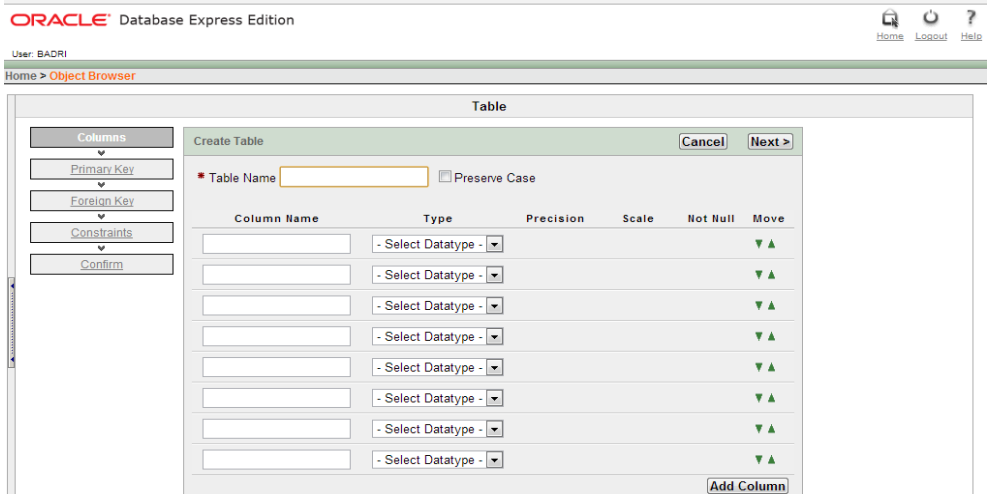
მაგალითის სახით განვიხილოთ Object Browser -ში ცხრილისა და მოთხოვნის შექმნა:

თანამიმდევრობით ვაწკაპუნებთ Object Browser – Create – ცხრილი.



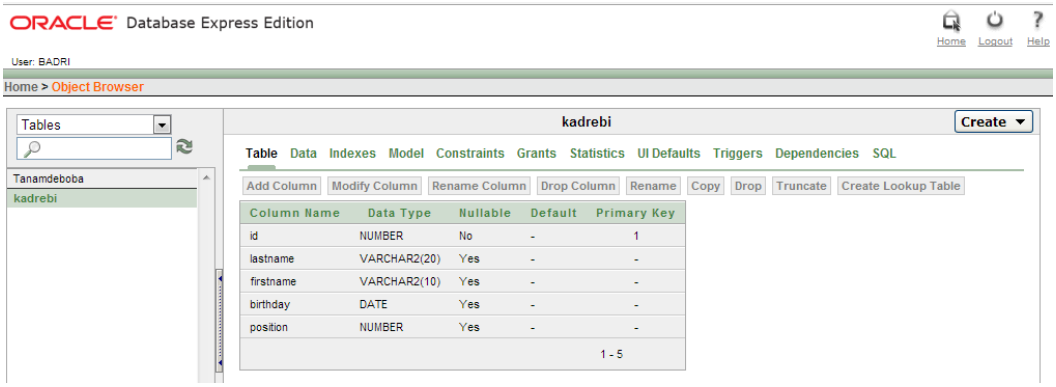
ნახ.4.6

იხსნება ცხრილის სტრუქტურის შექმნის ფანჯარა.

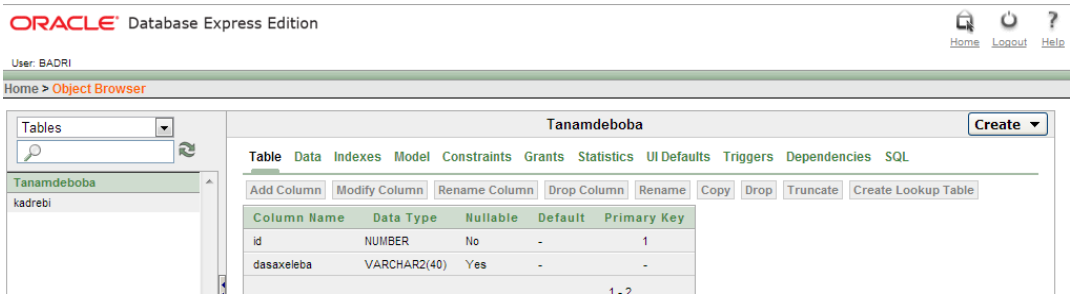


ნახ.4.7

ზემოთ აღწერილი ინსტრუქციის თანახმად ვქმნით ჯერ ერთ და შემდეგ მეორე ცხრილებს.

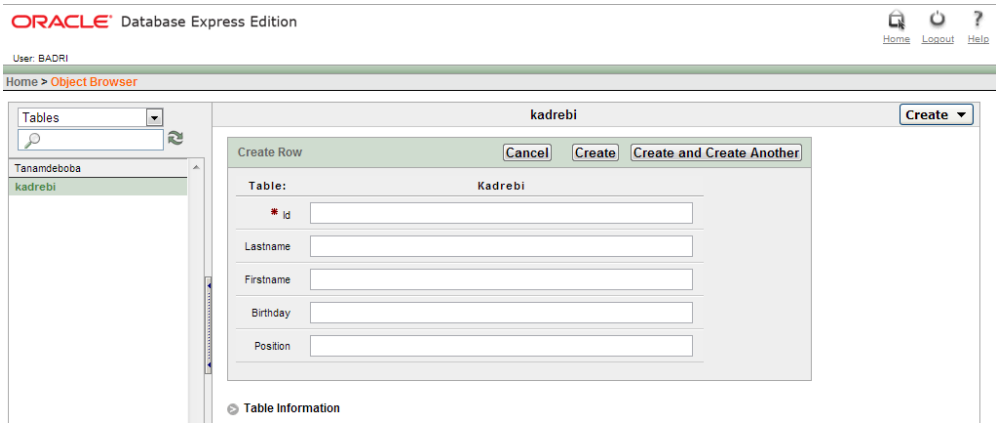


ნახ.4.8



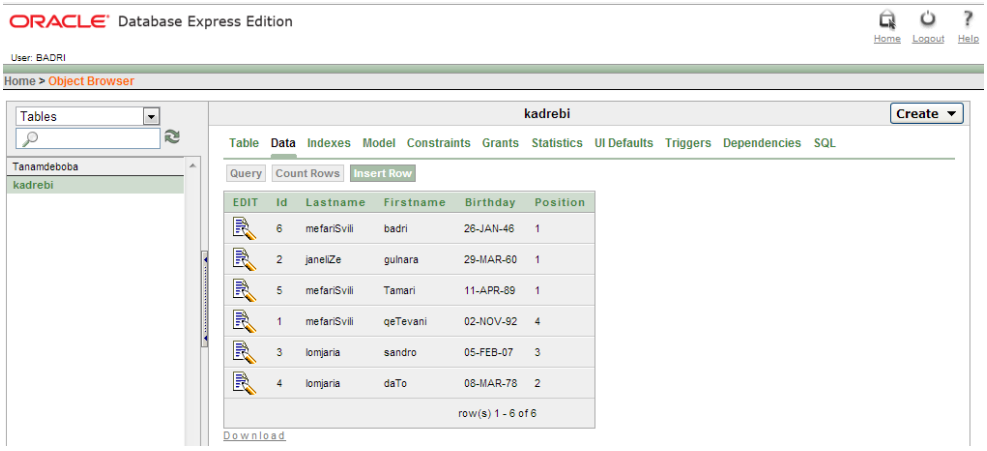
ნახ.4.9

Insert Row ღილაკზე დაწკაპუნებით შეგვაქვს მონაცემები. შემდეგ სტრიქონზე გადასასვლელად ვაწკაპუნებთ Create და Create Another ღილაკზე, ხოლო მონაცემთა შეტანის დამთავრებისათვის კი შესამაბისად Create ღილაკზე.



ნახ.4.10

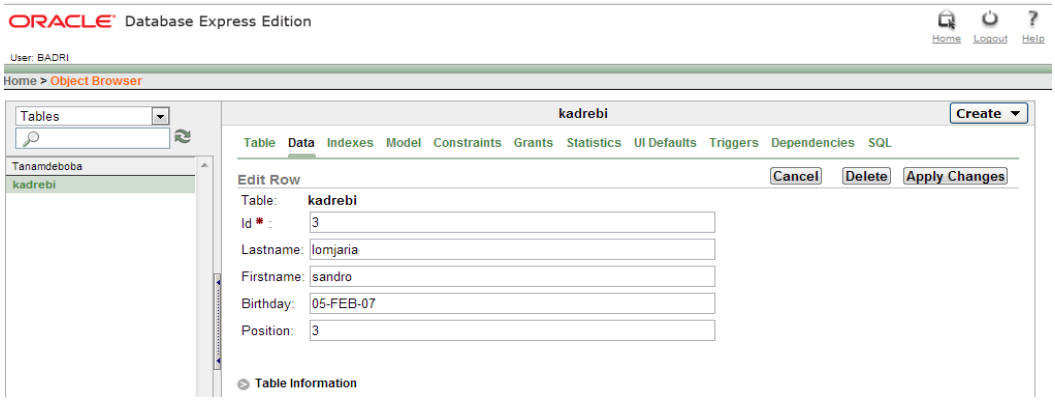
რამდენიმე ჩანაწერის შეტანის შემდეგ ვნახულობთ მათ.



ნახ.4.11

ნახ.4.12

ჩანაწერთა რედაქტირებისათვის ვაწკაპუნებთ სასურველი ჩანაწერის Edit იკონაზე, რაც შესწორებების გაკეთების საშუალებას იძლევა.



ნახ.4.13

შესწორების დაფიქსირებისათვის ვაწკაპუნებთ Apply Changes ღილაკზე.

ლაბორატორიული სამუშაო 5
მუშაობა Query Builder -ში

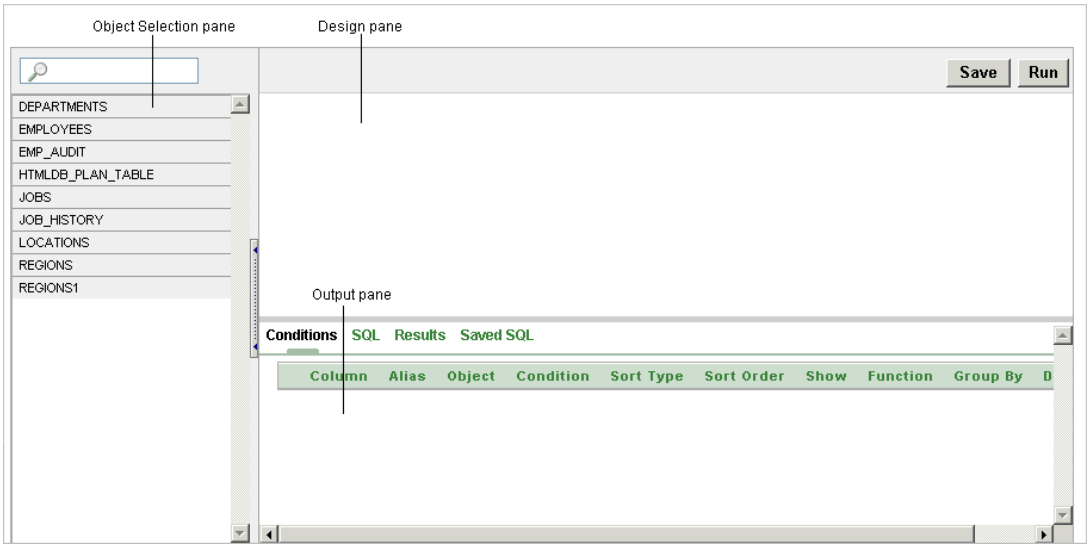
სამუშაოს მიზანი:

1. Query Builder -ში მოთხოვნებთან მუშაობის შესწავლა;
2. Query Builder -ში წარმოდგენებთან მუშაობის შესწავლა;
3. Query Builder -ში ინდექსებთან მუშაობის შესწავლა;
4. Query Builder -ში თანამიმდევრობებთან მუშაობის შესწავლა.

Query Builder -ში მოთხოვნებთან მუშაობა

Query Builder -ის გვერდი იყოფა სამ სექციად:

- Object Selection pane;
- Design pane;
- Output pane.

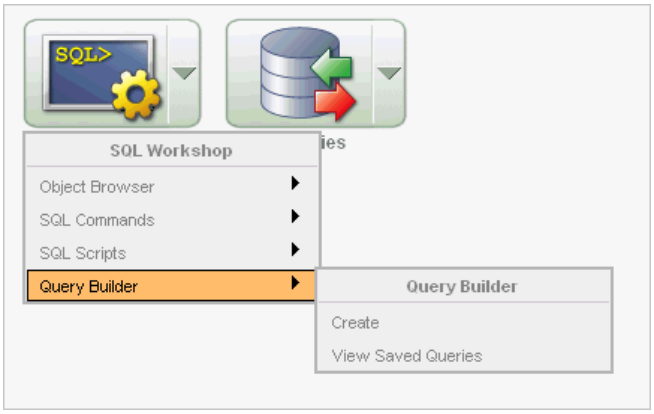


ნახ.5.1

Query Builder -სადმი წვდომისათვის:

1. Database Home Page -ზე ვაწკაპუნებთ **SQL** იკონაზე.
2. Query Builder -სათვის:

- ვაწკაპუნებთ **Query Builder** იკონაზე;
- ვაწკაპუნებთ იკონის მარჯვენა მხარეს. შედეგად იხსნება ჩამოშლადი მენიუ.



ნახ.5.2

მოთხოვნის აგების პროცესი

მოთხოვნის აგებისათვის უნდა შესრულდეს შემდეგი ბიჯები:

1. Object Selection pane -დან ვირჩევთ ობიექტს (ანუ ცხრილს).
2. ვამატებთ ობიექტებს Design pane -ში და ვირჩევთ სვეტებს.
3. (არ არის აუცილებელი): ვამყარებთ დამოკიდებულებას ობიექტებს შორის.
4. (არ არის აუცილებელი): ვქმნით მოთხოვნის პირობებს.
5. ვუშვებთ მოთხოვნას შესრულებაზე და ვნახულობთ შედეგებს.

Design pane-ში ასახულია სვეტების ყველა ხელმისაწვდომი ტიპი. ამასთან არსებობს შემდეგი შეზღუდვები:

- ჩვენ შეგვიძლია მაქსიმუმ 60 სვეტის ამორჩევა თითოეულ მოთხოვნაში.
- სვეტების შემდეგი ტიპები არ არიან ამორჩევითი და არ შეიძლება ჩართული იქნას მოთხოვნის აგების პროცესში:
  - BLOB
  - NCLOB
  - RAW
  - LONG
  - LONG RAW
  - XMLType
  - ნებისმიერი სხვა არასკალარული ტიპის სვეტი

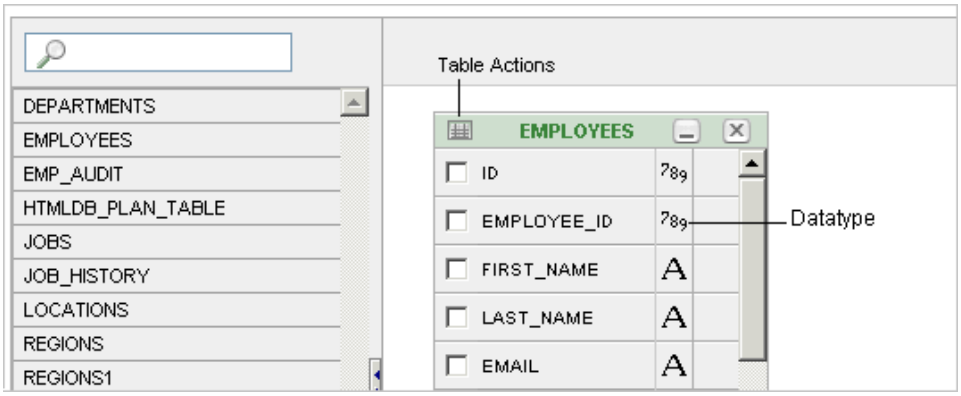
ობიექტის დამატება Design Pane-ში

Design pane-ში ობიექტის დამატებისათვის:

1. Workspace home page -ზე ვაწკაპუნებთ **SQL Workshop** და შემდეგ **Query Builder**.

   ჩნდება Query Builder.

2. Object Selection pane -დან ვირჩევთ ობიექტს, რომელიც ჩნდება Design Pane-ში. მონაცემთა ტიპების გრაფიკული წარმოდგენა მოყვანილია სვეტის მარჯვნივ.



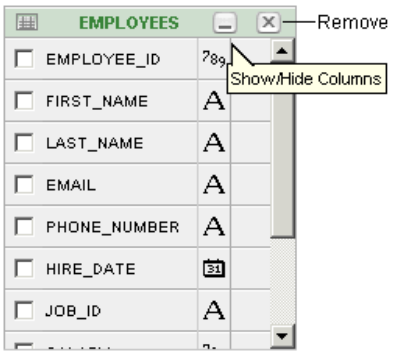
ნახ.5.3

მოთხოვნაში ველის ჩართვისათვის ვაწკაპუნებთ check box-ზე სვეტის სახელის მარცხნივ. ოცდამეერთე სვეტის არჩევისათვის ვაწკაპუნებთ იკონაზე **ცხრილი Actions** ობიექტის მარცხენა ზედა კუთხეში.

მოქმედებათა ფანჯარაში ვირჩევთ **Check All**.

3. მოთხოვნის შესრულებაზე გაშვებისა და შედეგების ნახვისათვის ვაწკაპუნებთ **Run** ან ვაჭერთ **CTRL + ENTER**.

Design Pane -ზე ობიექტის ამოგდებისათვის მარჯვენა ზედა კუთხეში ვირჩევთ იკონას **Remove**, ხოლო ობიექტის დროებითი დამალვისათვის ვაწკაპუნებთ იკონაზე **Show/Hide Columns**.



ნახ.5.4

მოთხოვნის პირობების დადგენა

მოთხოვნის პირობების დადგენისათვის:

1. Workspace home page -ზე ვაწკაპუნებთ **SQL Workshop** და შემდეგ **Query Builder**. ჩნდება Query Builder.
2. Object Selection pane -დან ვირჩევთ ობიექტს, რომელიც ჩნდება Design Pane -ში.
3. მოთხოვნაში სვეტების ჩართვისათვის ვაწკაპუნებთ check box-ზე სვეტის სახელის მარცხნივ.
4. თითოეული სვეტის შერჩევის შემდეგ ჩნდება პირობების (Conditions) შემცველი სტრიქონი, საიდანაც განვსაზღვრავთ საჭირო პირობებს. writes the SQL for you.
5. Query Builder -ის მიერ შედგენილი SQL კოდის ნახვისათვის, ვაწკაპუნებთ **SQL** tab -ზე.

ობიექტებს შორის დამოკიდებულებების შექმნა

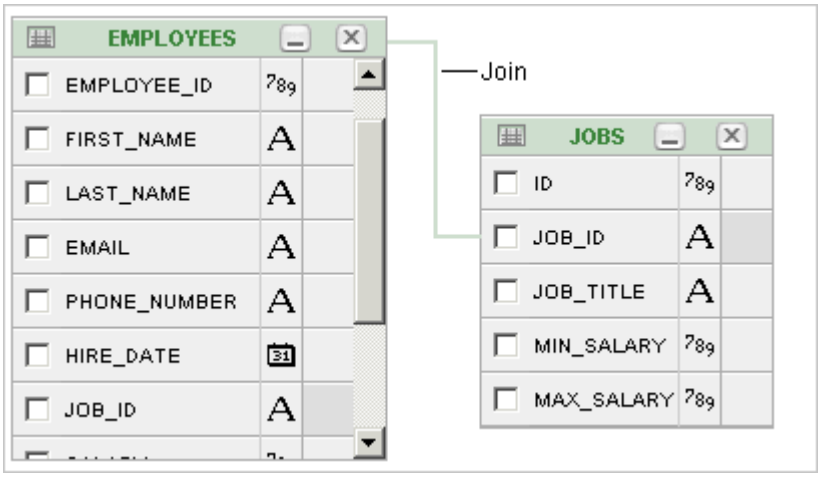
ორი ან მეტი ცხრილის ან წარმოდგენის სვეტების Design pane -ში ხელით დაკავშირებისათვის, რისთვისაც გამოიყენება ბრძანება **join**, სრულდება შემდეგი ქმედებები:

1. Workspace home page -ზე ვაწკაპუნებთ **SQL Workshop** და შემდეგ **Query Builder**. ჩნდება Query Builder.
2. Object Selection pane - დან ვირჩევთ ობიექტებს, რომელთა დაკავში-რებაც გვინდა.

ეს ობიექტები ჩნდება Design pane -ში.

3. ვირჩევთ სვეტებს, რომელთა მიხედვითაც მოხდება ობიექტების დაკავშირება, რისთვისაც ვაწკაპუნებთ ჯერ პირველი ობიექტის შესაბამისი სვეტის მარჯვნივ. შედეგად ეს უჯრა გამუქდება. მონიშვნის გასაუქმებლად ხელმეორედ მოვნიშნავთ ან ვაჭერთ **ESC**. შემდეგ ვირჩევთ მეორე ობიექტის შესაბამის სვეტს და მოვნიშნავთ მას.

ჩვენ შეგვიძლია ამ ობიექტების დაკავშირება აგრეთვე პირველი ობიექტის საჭირო სვეტიდან მეორე ობიექტის შესაბამის სვეტამდე მაუსის დაჭერილ მდგომარეობაში უწყვეტი მოძრაობით და აშვებით ე.წ. „გადათრევით“ (dragging და dropping).



ნახ.5.5

შედეგად ჩნდება შემაერთებელი ხაზი.

4. შერჩეული სვეტები ჩართული იქნებიან ჩვენს მოთხოვნაში.
5. მოთხოვნის შესრულებაზე გაშვებისათვის ვაწკაპუნებთ **Run**.
6. შედეგი აისახება Results pane -ში.

Query Builder განიხილავს არსებულ სვეტებს შორის მშობელ და შვილობილ კავშირებს. ობიექტების ავტომატური დაკავშირებისათვის:

1. Workspace home page -ზე ვაწკაპუნებთ **SQL Workshop** და შემდეგ **Query Builder**.
   ჩნდება Query Builder.
2. Object Selection pane -დან ვირჩევთ ობიექტს, რომელიც აისახება Design pane -ში.

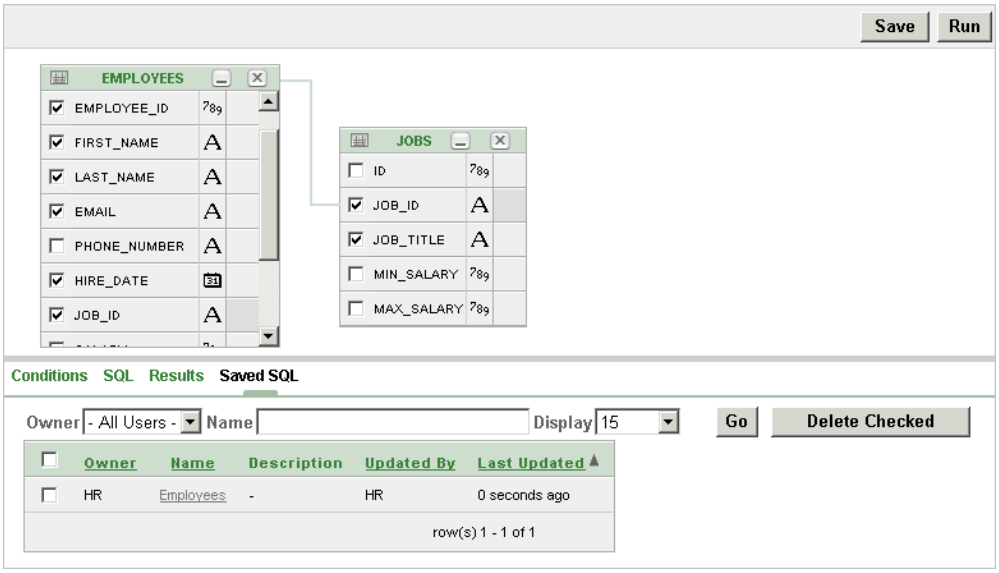
3. ვაწკაპუნებთ მცირე იკონაზე ობიექტის მარცხენა ზედა კუთხეში. შერჩეული ობიექტის გვარობიდან გამომდინარე წარწერა იკონაზე იქნება ან **ცხრილი Actions** ან **წარმოდგენა Actions**.

4. Actions ფანჯარაში ვირჩევთ შესაბამის ოპციებს:
   - **Check All** - პირველი ოცი სვეტის შერჩევის შემთხვევაში.
   - **Add Parent**.
   - **Add Child**.

   ჩნდება მონიშნული ობიექტები კავშირის ხაზით.
5. შერჩეული სვეტები ჩართული იქნებიან ჩვენს მოთხოვნაში.
6. მოთხოვნის შესრულებაზე გაშვებისათვის ვაწკაპუნებთ **Run**.

შედეგი აისახება პანელის Results ბრძანებაში.

7. მოთხოვნის შენახვისათვის ვაწკაპუნებთ **Save**.
8. შეგვაქვს მოთხოვნის სახელი და აღწერილობა. ვაწკაპუნებთ **Save**.
9. შენახული მოთხოვნა აისახება პანელის Saved SQL ბრძანებაში.



ნახ.5.6

შენახული მოთხოვნის რედაქტირება

მოთხოვნის რედაქტირება ხდება Saved SQL-ში, რისთვისაც:

1. Workspace home page-ში ვაწკაპუნებთ **SQL Workshop** და შემდეგ **Query Builder**. ჩნდება Query Builder.
2. ვირჩევთ **Saved SQL** tab-ს.
3. გაფილტვრისათვის:
   - Owner სიიდან ვირჩევთ და ვაწკაპუნებთ **Go**.
   - საძიებელი მოთხოვნა შეგვაქვს ველში Name და ვაწკაპუნებთ **Go**.
4. ვირჩევთ საჭირო მოთხოვნის სახელს. ჩნდება შერჩეული მოთხოვნა.
5. Design pane -ში ჩნდება შენახული ობიექტების სახელები და Conditions წარმოდგენა.

შენახული მოთხოვნის წაშლა

Saved SQL -ში მოთხოვნის წაშლისათვის:

1. Workspace home page-ში ვაწკაპუნებთ **SQL Workshop** და

შემდეგ **Query Builder**. ჩნდება Query Builder.

2. ვირჩევთ **Saved SQL** tab.
3. ვირჩევთ წასაშლელ მოთხოვნას და ვაწკაპუნებთ **Delete Checked**.

SQL კოდის ნახვა

SQL კოდი ნახვისათვის:

1. Workspace home page -ში ვაწკაპუნებთ **SQL Workshop** და შემდეგ **Query Builder**. ჩნდება Query Builder.
2. Object Selection pane -და ვირჩევთ ობიექტს, რომელიც ჩნდება Design Pane -ში.
3. ვირჩევთ სვეტებს ჩვენს მოთხოვნაში.
4. ვაწკაპუნებთ **SQL** tab.
   ჩნდება Query Builder-ის მიერ გენერირებული SQL კოდი.

მოთხოვნის შედეგების ნახვა

მოთხოვნის შესრულებაზე გაშვებისათვის:

- ვაწკაპუნებთ ღილაკზე **Run** (ან ვაჭერთ **CTRL + ENTER**)
- ვირჩევთ **Results** tab

ჩნდება მოთხოვნის შესრულების შედეგები. (.csv) ფაილის სახით ანგარიშის ექსპორტისათვის გვერდის ქვემოთ ვაწკაპუნებთ Download link.

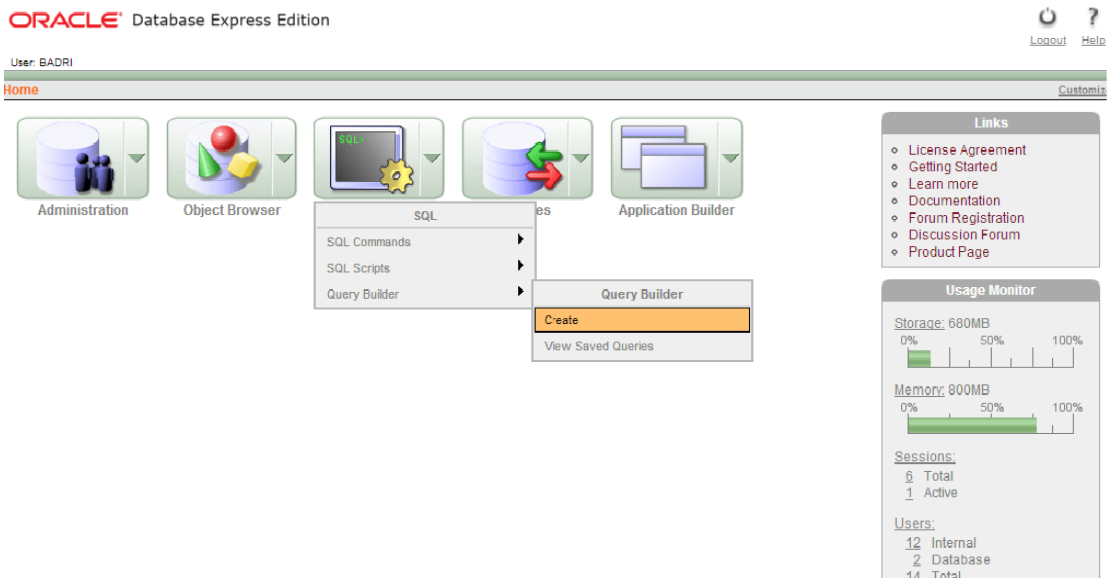
Conditions SQL **Results** Saved SQL

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	JOB_ID	JOB_TITLE
100	Steven	King	SKING	17-JUN-87	AD_PRES	AD_PRES	President
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	AD_VP	Administration Vice President
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	AD_VP	Administration Vice President
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	IT_PROG	Programmer
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	IT_PROG	Programmer
105	David	Austin	DAUSTIN	25-JUN-97	IT_PROG	IT_PROG	Programmer
106	Valli	Pataballa	VPATABAL	05-FEB-98	IT_PROG	IT_PROG	Programmer

ნახ.5.7

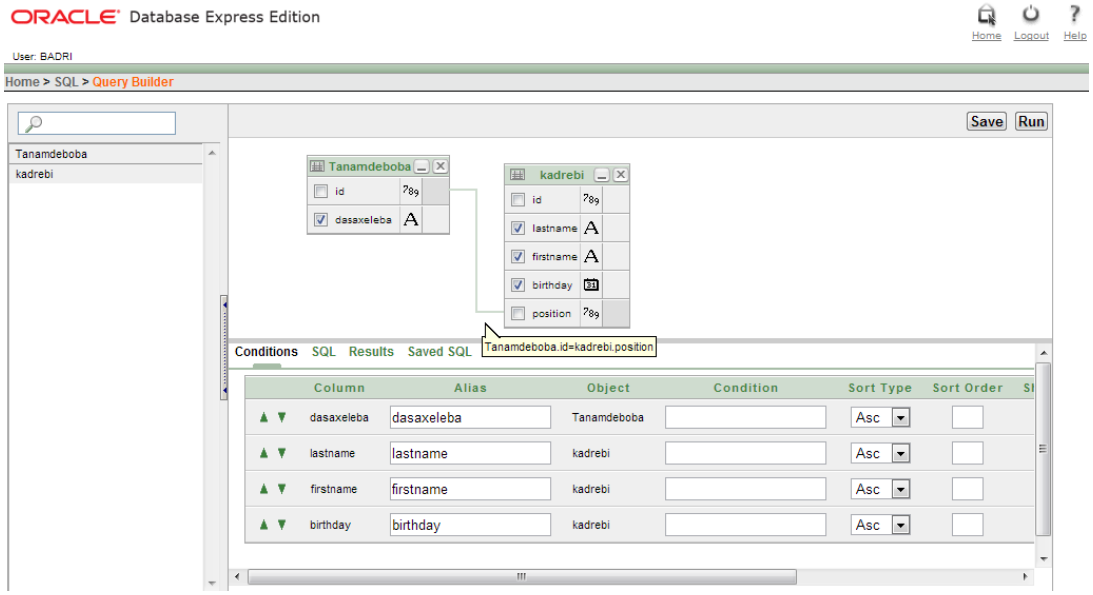
მაგალითი

გადავდივართ კონკრეტული მოთხოვნის შექმნაზე. ამისათვის თანამიმდევრობით ვაწკაპუნებთ SQL – Query Builder – Create.



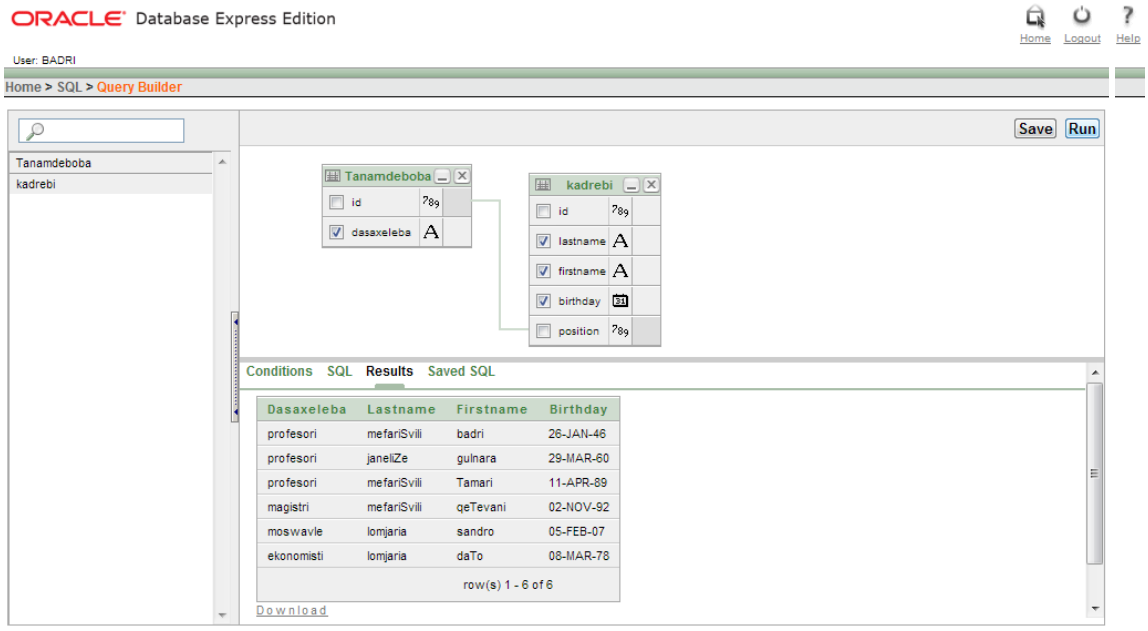
ნახ.5.8

დაწკაპუნებით Object Selection pane -დან ცხრილები რიგრიგობით გადაგვაქვს Detail pane-ში. შემდეგ ვაწკაპუნებთ ცხრილების პირველად და გარე გასაღებური ველების მარჯვენა უჯრაზე. იკვრება ცხრილებს შორის კავშირი.



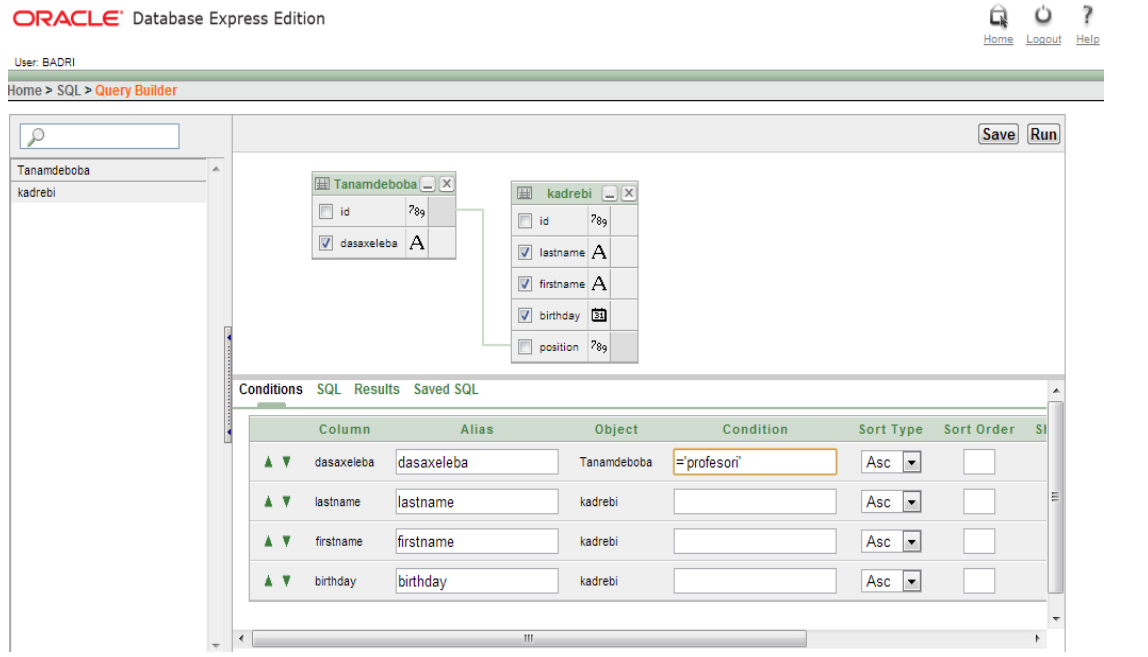
ნახ.5.9

ვაწკაპუნებთ ღილაკზე Run. სრულდება მოთხოვნა.



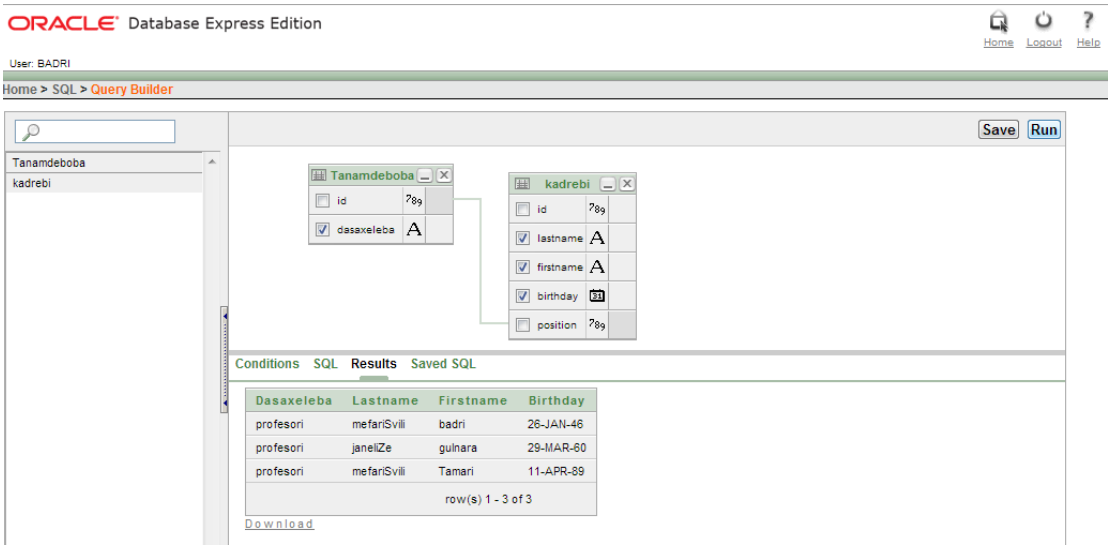
ნახ.5.10

შემდეგ საჭირო ველისათვის სვეტში Condition შეგვაქვს ამორჩევის კრიტერიუმი.



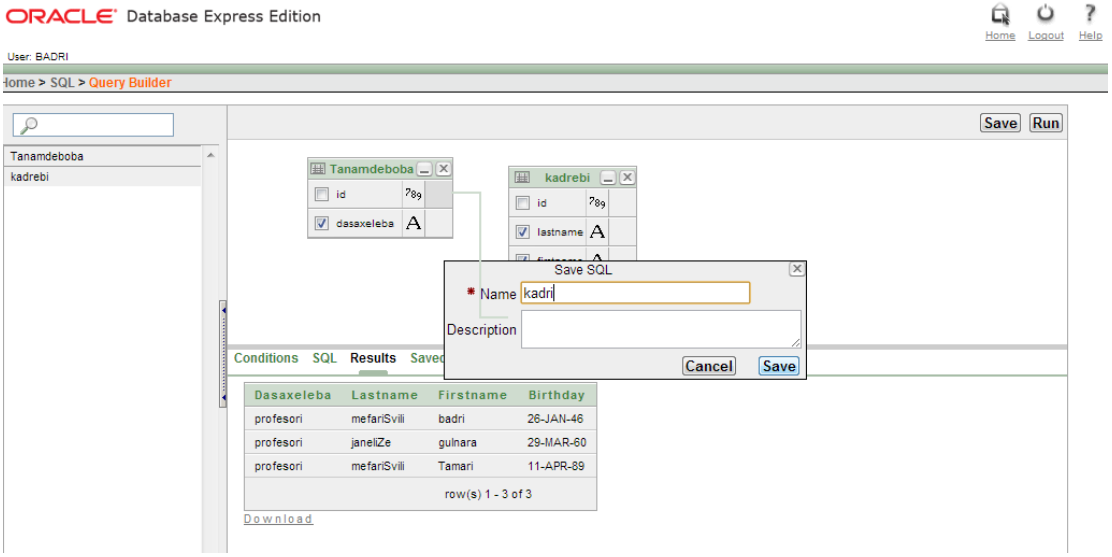
ნახ.5.11

კვლავ ვაწკაპუნებთ ღილაკზე Run. სრულდება ახალი მოთხოვნა,



ნახ.5.12

რომელსაც ვარქმევთ სახელს და ვინახავთ.

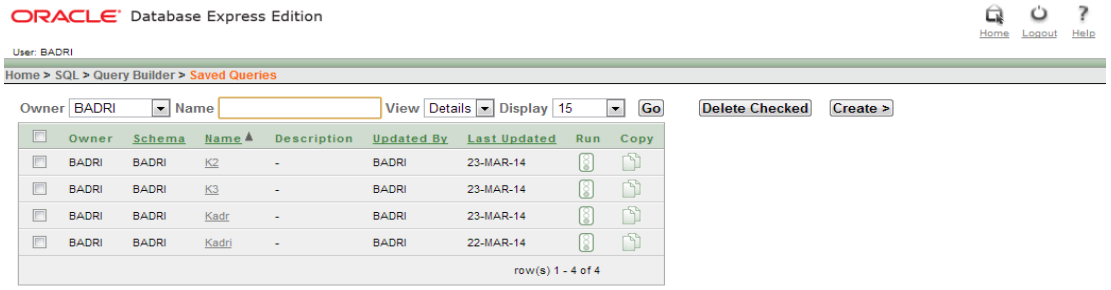


ნახ.5.13

მოთხოვნის რედაქტირებისათვის ვაწკაპუნებთ: Query Builder – წარმოდგენა Saved Queies.

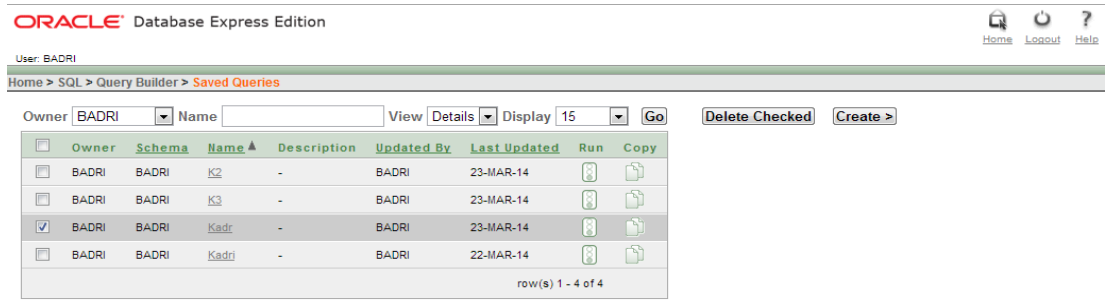
ნახ.5.14

გამოდის შექმნილი მოთხოვნების სია.



ნახ.5.15

ვირჩევთ საჭირო მოთხოვნას. მოვნიშნავთ და ვაწკაპუნებთ იკონაზე მის გასწვრივ Run სვეტში.

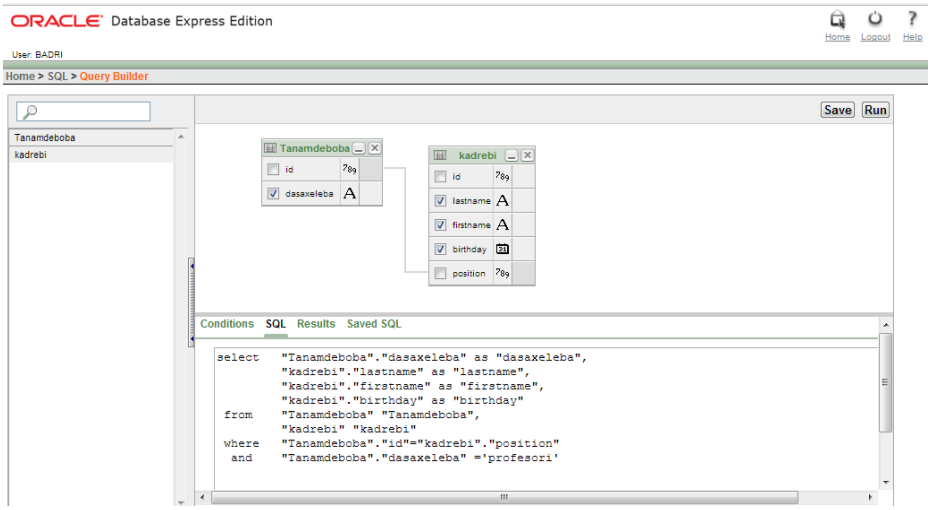


ნახ.5.16

გამოდის მოთხოვნა. ვაწკაპუნებთ ღილაკზე Edit Query.

ნახ.5.17

მოცემული მოთხოვნის SQL კოდის სანახავად Detail pane-ში გადავდივართ SQL -ზე.



ნახ.5.18

წარმოდგენები

წარმოდგენის შექმნა

წარმოდგენის შექმნისათვის:

1. Database Home Page -ზე ვაწკაპუნებთ იკონაზე **Object Browser**.

ჩნდება Object Browser.

2. ვაწკაპუნებთ **Create**.
3. ობიექტების ტიპების სიიდან ვირჩევთ **წარმოდგენას**.
4. წარმოდგენის განსაზღვრისათვის:
   - **წარმოდგენა Name** - შეგვაქვს წარმოდგენის სახელი.
   - **Query** - მივუთითებთ მოთხოვნაზე.

Query Builder-ის ან SQL Command Processor -ის წვდომისათვის ვაწკაპუნებთ შესაბამის link -ზე გვერდის ქვევით. pop-up ფანჯარაში ჩნდება შერჩეული ინსტრუმენტი. ერთხელ ვქმნით შესაბამის SQL -ს, popup ფანჯრის ავტომატურად დახურვისათვის და SQL ოსტატში დაბრუნებისათვის ვაწკაპუნებთ **Return**.

5. ვაწკაპუნებთ **Next**.

ჩნდება შესაბამისი გვერდი. წარმოდგენის შექმნისათვის გამოყენებული SQL -ის ნახვისათვის ვაწკაპუნებთ **SQL**.

6. ვაწკაპუნებთ **Create**.

წარმოდგენის რედაქტირება

წარმოდგენის რედაქტირებისათვის:

1. Database Home Page-ზე ვაწკაპუნებთ იკონაზე **Object Browser**.
2. ჩნდება Object Browser. .
3. ობიექტების ტიპების სიიდან ვირჩევთ **წარმოდგენას**.
4. Object Selection pane -დან ვირჩევთ წარმოდგენას.

წარმოდგენის განსაზღვრისათვის გამოჩნდება შესაბამისი სვეტები: წარმოდგენა, Data, Grants, UI Defaults (User interface defaults), Dependencies, SQL.

წარმოდგენის ხელით რედაქტირება

წარმოდგენის ხელით რედაქტირებისათვის:

1. Database Home Page-ზე ვაწკაპუნებთ იკონაზე **Object Browser**.
2. ჩნდება Object Browser. .
3. ობიექტების ტიპების სიიდან ვირჩევთ **წარმოდგენას**.
4. ვირჩევთ Code tab.
5. ვაწკაპუნებთ **Edit.**

Find და Replace ბრძანებების გამოყენება

ძებნისა და შეცვლის შესრულებიათვის ვაწკაპუნებთ **Find**.

წარმოდგენის ჩატვირთვა-შენახვა

მიმდინარე წარმოდგენის ფაილის სახით შენახვისათვის ვაწკაპუნებთ **Download**.

წარმოდგენის კომპილაცია

წარმოდგენის ცვლილების შემთხვევაში შენახვის მიზნით საჭირო არის კომპილაცია. მიმდინარე წარმოდგენის ხელმეორედ შექმნის ანუ კომპილაციის მიზნით ვაწკაპუნებთ **Compile**.

წარმოდგენის წაშლა

წარმოდგენის წაშლისათვის:

1. Database Home Page-ზე ვაწკაპუნებთ იკონაზე **Object Browser**.
2. ჩნდება Object Browser.
3. ობიექტების ტიპების სიიდან ვირჩევთ **წარმოდგენას**.
4. ვირჩევთ **წარმოდგენა** tab ან **Code** tab.
5. მიმდინარე წარმოდგენის წაშლისათვის ვაწკაპუნებთ **Drop**.

ინდექსები

ინდექსის შექმნა

ინდექსის შექმნისათვის:

1. Database Home Page-ზე ვაწკაპუნებთ იკონაზე **Object Browser**.

ჩნდება Object Browser.

2. ვაწკაპუნებთ **Create**.
3. ობიექტების ტიპების სიიდან ვირჩევთ **Index**.
4. ვირჩევთ ცხრილს და ცხრილის ტიპს, რომლის ინდექსის შექმნაც გვინდა. განირჩევა ინდექსის ტიპები:
   - **Normal** - ცხრილის ატრიბუტების ერთი ან რამდენიმე სკალარული ობიექტის ინდექსები.
   - **Text** - ქმნის ტექსტურ ინდექსს (Oracle Text).
5. ვაწკაპუნებთ **Next**.
6. განვსაზღვრავთ ინდექსის სახელს. ვირჩევთ ერთ ან რამდენიმე სვეტს ინდექსირებისათვის და ვაწკაპუნებთ **Next**.
7. შექმნილი ინდექსის SQL ნახვისათვის ვაწკაპუნებთ **SQL.**
8. ვაწკაპუნებთ **Finish**.

ინდექსის რედაქტირება

ინდექსის რედაქტირებისათვის:

1. Database Home Page-ზე ვაწკაპუნებთ იკონაზე **Object Browser**.

ჩნდება Object Browser.

2. ობიექტების ტიპების სიიდან ვირჩევთ **Indexes**.
3. Object Selection pane -დან ვირჩევთ ინდექსს. ჩნდება ინდექსის სახელი, ტიპი, ცხრილის მფლობელი (owner) და ინდექსირებული სვეტების სია.

ინდექსის წაშლა

ინდექსის წაშლისათვის:

1. Database Home Page-ზე ვაწკაპუნებთ იკონაზე **Object Browser**. ჩნდება Object Browser.
2. ობიექტების ტიპების სიიდან ვირჩევთ **Indexes**.
3. Object Selection pane -დან ვირჩევთ ინდექსს.
4. Object Details ქვევით ვაწკაპუნებთ **Drop** tab.

თანამიმდევრობა (Sequence)

თანამიმდევრობის (Sequence) შექმნა

თანამიმდევრობის შექმნისათვის:

1. Database Home Page-ზე ვაწკაპუნებთ იკონაზე **Object Browser**.
ჩნდება Object Browser.
2. ობიექტების ტიპების სიიდან ვირჩევთ **Sequences**.
3. თანამიმდევრობის განსაზღვრისათვის ვირჩევთ მის სახელს და ვაწკაპუნებთ **Next**.
4. თანამიმდევრობის შექმნაში გამოყენებული SQL ნახვისათვის ვაწკაპუნებთ **Show SQL.**
5. ვაწკაპუნებთ **Create**.

თანამიმდევრობის რედაქტირება

თანამიმდევრობის რედაქტირებისათვის:

1. Database Home Page-ზე ვაწკაპუნებთ იკონაზე **Object Browser**.
ჩნდება Object Browser.
2. ობიექტების ტიპების სიიდან ვირჩევთ **Sequences**.
3. Object Selection pane -დან ვირჩევთ თანამიმდევრობას.

თანამიმდევრობის წაშლა

თანამიმდევრობის წაშლისათვის:

1. Database Home Page -ზე ვაწკაპუნებთ იკონაზე **Object Browser**.
ჩნდება Object Browser.
2. Object სიიდან ვირჩევთ **Sequences**.
3. Object Selection pane-დან ვირჩევთ თანამიმდევრობას. ჩნდება Object Details.
4. ვაწკაპუნებთ **Drop**.

ლაბორატორიული სამუშაო № 6
დანართის შემუშავება Oracle-ის გარემოში

სამუშაოს მიზანი:

1. მარტივი ანგარიშიანი დანართის შემუშავების შესწავლა;
2. დეტალური ანგარიშიანი დანართის შემუშავების შესწავლა;
3 .„მთავარი - დაქვემდებარებული“ ანგარიშიანი დანართის შემუშავების შესწავლა.

Oracle დანართის ექსპრეს-შემუშავება

მონაცემთა ბაზების Web-დანართის სწრაფი შემუშავებისათვის გამოიყენება პაკეტი Oracle Application Express (APEX). იმის გამო, რომ ეს პაკეტი ფასიანია, შემუშავებულია კიდევ სხვა ინსტრუმენტი Application Builder, რომლის მომხმარებლის ინტერფეისი აწყობილია APEX გარემოში და წარმოადგენს APEX-ის ერთ-ერთ კომპონენტს.

დანართი, რომელსაც ჩვენ შევიმუშავებთ, შეიცავს ორი ტიპის ანგარიშებს: მარტივი და „მთავარი-დეტალური“ ტიპის. თუ მარტივ ანგარიშში აისახება ერთი ცხრილის მონაცემები, „მთავარი-დეტალური“ ტიპის ანგარიში აიგება არანაკლებ ურთიერთდაკავშირებული ორი ცხრილის საფუძველზე, სადაც ერთი წარმოადგენს მთავარ ცხრილს (Master ცხრილი), ხოლო მეორე – დაქვემდებარებულ (დეტალურ, Detail ცხრილი) ცხრილს.

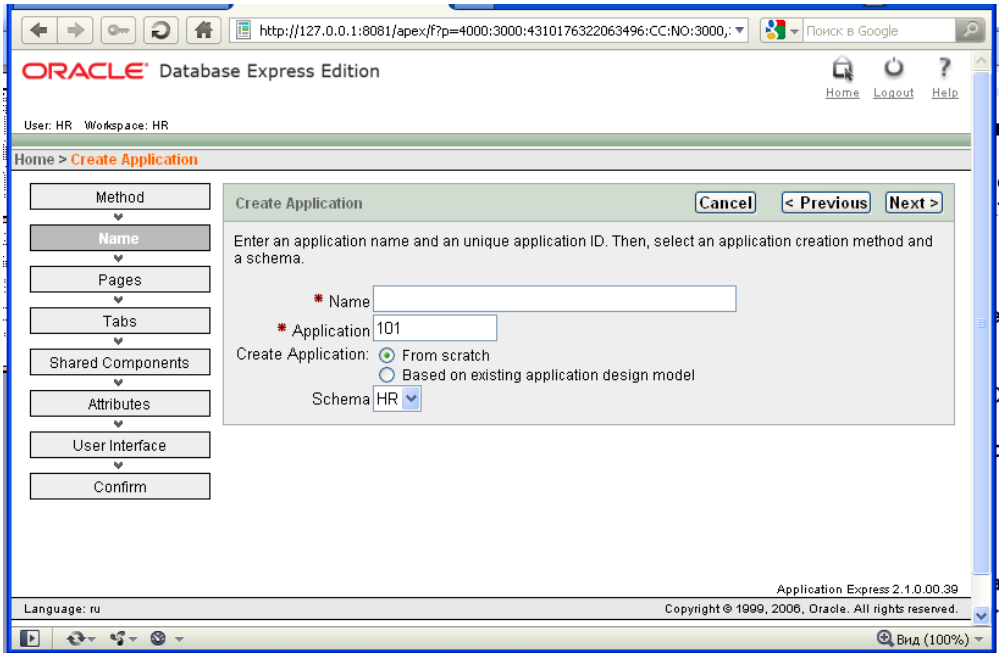
შესაბამისად, დანართში იქმნება ორი კომპონენტი (მაგალითად, სტრიქონების ბადის სახით), რომლებიც ცხრილებთან არის დაკავშირებული. მთავარი ცხრილის კომპონენტი ასახავს ცხრილის ყველა ჩანაწერს. დეტალური ცხრილის კომპონენტი კი ასახავს მხოლოდ იმ ჩანაწერებს, რომლებიც დაკავშირებული არიან მთავარი ცხრილის შერჩეულ ჩანაწერთან. ანგარიშები აიგება შენახული მოთხოვნების საფუძველზე.

მარტივი ანგარიშიანი დანართის შემუშავება

მაგალითისათვის შევქმნათ მოთხოვნა, რომელიც აერთიანებს ცხრილებს Departments, Locations და Countries, რისთვისაც გამოვიყენოთ

გრაფიკული ინსტრუმენტი – Query Builder. შექმნილი მოთხოვნა შევინახოთ შემდგომი გამოყენების მიზნით და გავუშვათ შესრულებაზე. შენახული მოთხოვნის მეშვეობით ანგარიშში უნდა აისახოს ცხრილიდან ამოღებული ყველა მონაცემი. ამისათვის:

1. ბრაუზერის მთავარ გვერდზე Home ვუშვებთ Application Builder -> ღილაკი ქვემოთ დახრილი ისრით -> Create Application -> Create Application. იხსნება გვერდი Home> Create Application, რომელზეც იწყებს მუშაობას დანართის ოსტატი (ნახ.6.1).

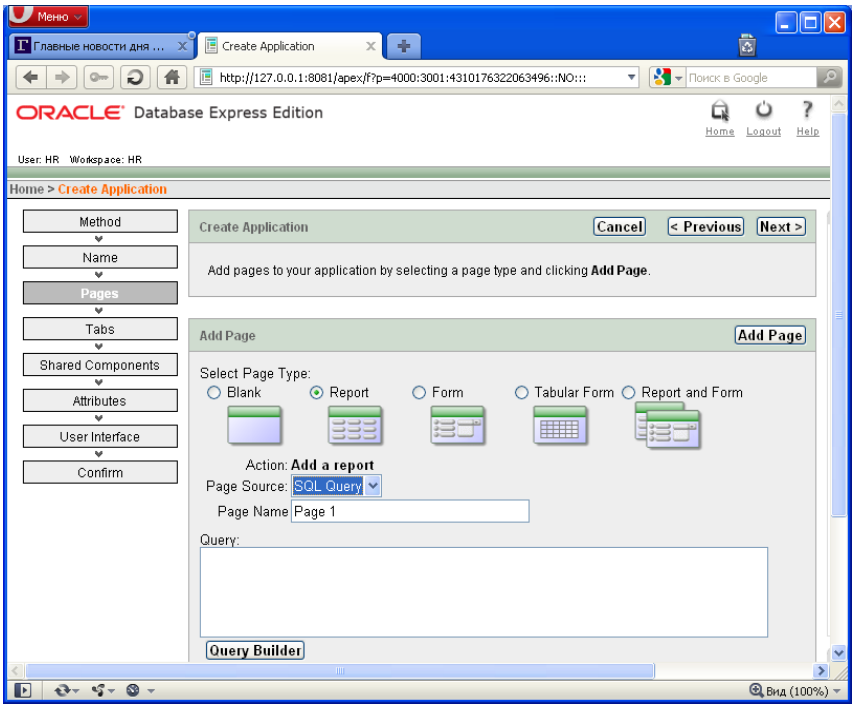


ნახ. 6.1

2. გადავდივართ ბიჯზე Name. სტრიქონში Name შეგვაქვს შესაქმნელი დანართის სახელი (მაგალითად, HR Application) -> Next.

3. გადავდივართ ბიჯზე Pages. აქ ხდება დანართის გვერდის განსაზღვრა, რომელზეც განთავსდება ანგარიშები და კომპონენტები მონაცემებთან სამუშაოდ. ჯერჯერობით ჩვენს დანართს გააჩნია ერთი გვერდი.

4. ვირჩევთ ანგარიშის ტიპს Report (მარტივი ანგარიში). ვირჩევთ მონაცემთა წყაროს ანგარიშისათვის Page Source: -> SQL Query (ნახ.).



ნახ. 6.2

4. გადავდივართ ბიჯზე Pages. შესასრულებლად ვირჩევთ შენახულ მოთხოვნას: Query Builder -> გადავდივართ მოთხოვნების კონსტრუქტორის გვერდზე -> Saved SQL -> ვირჩევთ **department locations** -> Return -> ტექსტურ ველში Query: გამოჩნდება შენახული მოთხოვნის ტექსტი-> გვერდის სახელი Page Name -> შეგვაქვს ტექსტი Departments -> Add page -> Next.

5. გადავდივართ ბიჯზე Tabs -> ვტოვებთ მნიშვნელობას One Level of Tabs -> Next.

6. გადავდივართ ბიჯზე Shared Components -> ვტოვებთ მნიშვნელობას No -> Next.

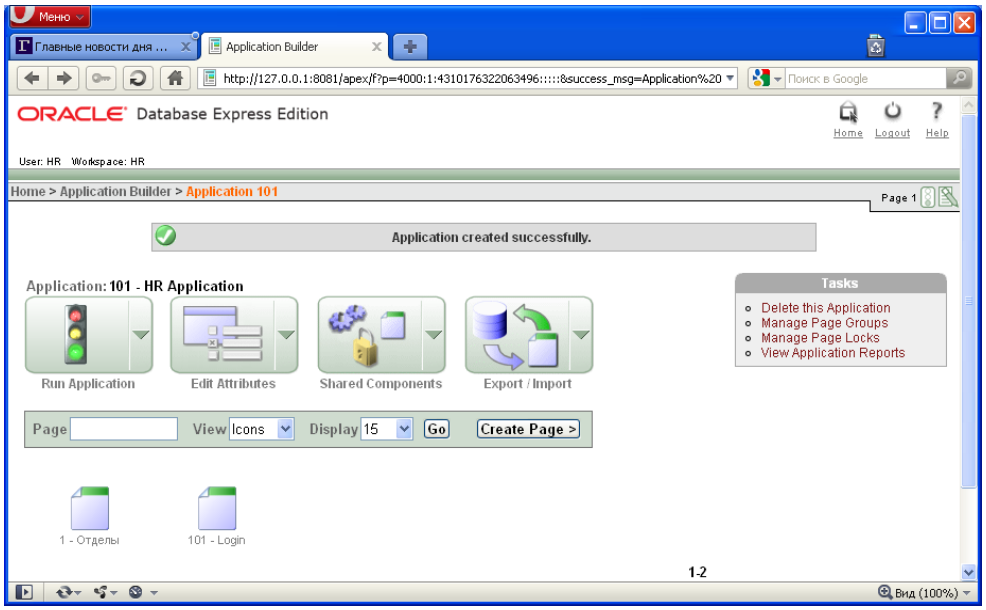
7. გადავდივართ ბიჯზე Attributes. ვაყენებთ Authentification Scheme: -> No Authentification -> ვტოვებთ ენობრივ სტანდარტებს -> Next.

8. გადავდივართ ბიჯზე User Interface. ვირჩევთ ფანჯრის სტილს-> Next.

9. გადავდივართ ბიჯზე Confirm. ვამოწმებთ დანართის დაყენებული თვისებების სისწორეს -> Create.

10. იხსნება გვერდი (ნახ. 3). ვაწკაპუნებთ იკონაზე Run Application. ჩნდება ანგარიშის ფანჯარა.

11. შევცვალოთ სორტირების თანამიმდევრობა ცხრილში (ვაწკაპუნებთ ცხრილის სახელზე -> სორტირება აღმავლობით-> მეორე ვაწკაპუნებთ -> სორტირება დაღმავლობით). ვახორციელებთ ძებნას რაიმე ნიშნით -> ჩნდება ნაპოვნი სტრიქონი (ან სტრიქონები) -> სრული ასახვისაკენ დაბრუნებისათვის Reset.



ნახ. 6.3

დეტალური ანგარიშიანი დანართის შემუშავება

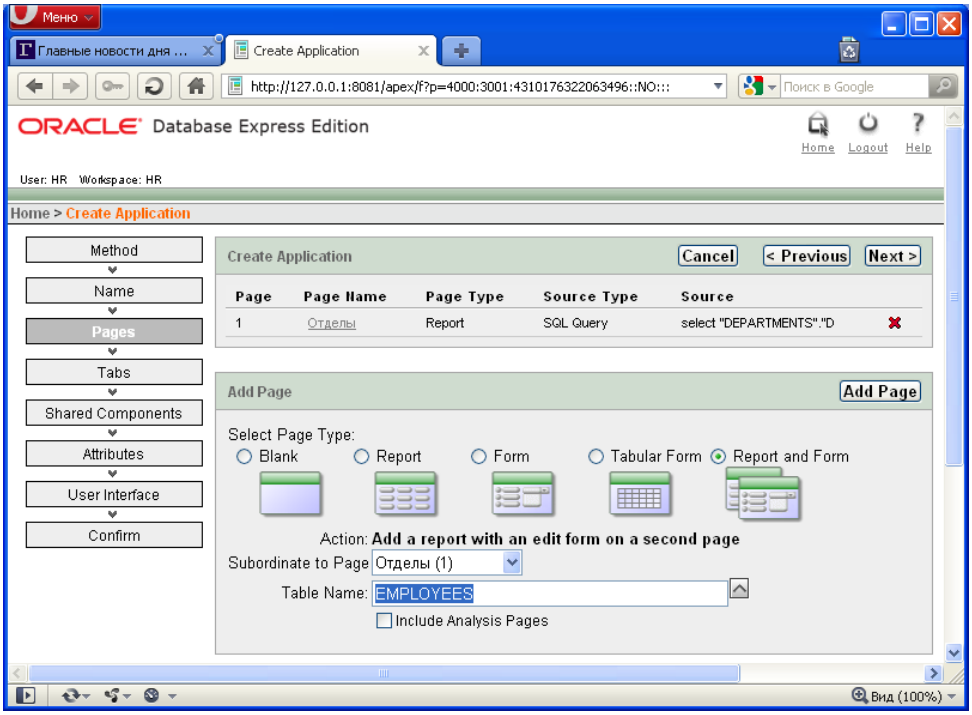
დანართს ექნება სამი გვერდი. პირველი გვერდი – შექმნილი მარტივი ანგარიშის გვერდი, რომელიც ამჯერად მთავარი ცხრილის სახით გვევლინება. მეორე გვერდზე განთავსებულია სტრიქონების ბადე, რომელიც ასახავს დაქვემდებარებულ ცხრილს. და ბოლოს, მესამე გვერდზე იქნება განთავსებული ფორმა, რომელზეც აისახება დეტალური მონაცემები მეორე გვერდზე ამორჩეული ჩანაწერისათვის. ამ ფორმის დანიშნულებას წარმოადგენს ერთი კონკრეტული ჩანაწერის ნახვა და რედაქტირება. ასეთ ფორმას ზოგჯერ უწოდებენ ჩაღრმავებულ ანგარიშს (Report Drill Down). „მთავარი - დაქვემდებარებული“ ანგარიშის შექმნისათვის:

1. სრულდება მარტივი ანგარიში შექმნის 1 – 3 პუნქტი.

2. გადავდივართ ბიჯზე Pages. შესრულებისათვის ვირჩევთ შენახულ მოთხოვნას: Query Builder -> გადავდივართ კონსტრუქტორის გვერდზე -> Saved SQL -> ვირჩევთ **department locations** -> შევცვალოთ ფსევდონიმი სვეტისათვის DEPARTMENT_ID -> Return -> ტექსტურ ველში Query: ჩნდება მოთხოვნის ტექსტი შესწორებებით -> გვერდის სახელი Page Name -> შეგვაქვს ტექსტი Departments -> Add page.

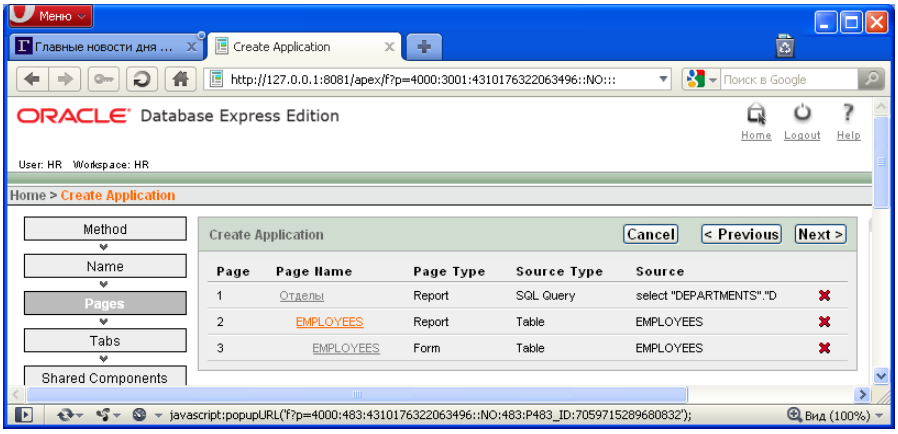
3. გადავდივართ ბიჯზე Pages. შევქმნათ ორი გვერდი დაკავშირებული ანგარიშისათვის და ჩაღრმავებული ანგარიშისათვის.

ვირჩევთ ღილაკს Report და Form -> ვირჩევთ მთავარ ცხრილს: Subordinate to Page მნიშვნელობა department -> ვირჩევთ დაქვემდებარებულ ცხრილს: ცხრილი Name მნიშვნელობა Employees -> Add page.



ნახ. 6.4

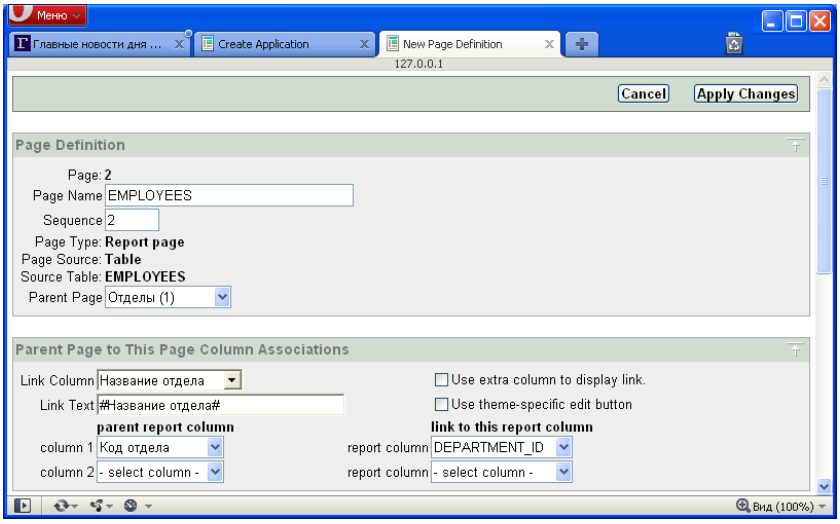
4. გადავდივართ ბიჯზე Pages. ამჯერად აუცილებელია განისაზღვროს კავშირის პირობები მთავარ ცხრილსა Departments და დაქვემდებარებულ ცხრილს Employees შორის. გვერდების სიაში ვირჩევთ მე-2 გვერდს (Employees). იხსნება ფანჯარა Page Definition.



ნახ. 6.5

5. გადავდივართ ბიჯზე Pages – Page Definition მე-2 გვერდისათვის.

6. ვირჩევთ Select Link Column მნიშვნელობას DEPARTMENT_NAME-> column1 მნიშვნელობას DEPARTMENT_ID -> report column მნიშვნელობას DEPARTMENT_ID -> ვხსნით დროშას use theme-specific edit button -> Apply Changes.



ნახ. 6.6

6. გადავდივართ ბიჯზე Pages. გადასვლა ხდება ნახ. 6.5 -ის მიხედვით -> Next.

7. გადავდივართ ბიჯზე Tabs -> ვტოვებთ მნიშვნელობას One Level of Tabs -> Next.

8. გადავდივართ ბიჯზე Shared Components -> ვტოვებთ მნიშვნელობას No -> Next.

9. გადავდივართ ბიჯზე Attributes. ვაყენებთ Authentification Scheme: -> No Authentification -> უსიტყვოდ ვტოვებთ ენობრივი სტანდარტების მნიშვნელობებს-> Next.

10. გადავდივართ ბიჯზე User Interface. ვირჩევთ დანართის ფანჯრის სტილს-> Next.

11. გადავდივართ ბიჯზე Confirm. ვამოწმებთ დანართის დაყენებული თვისებების სისწორეს -> Create.

12. იხსნება დანართთან მუშაობის გვერდი.

ვაწკაპუნებთ იკონაზე Run Application. ვაწკაპუნებთ გვერდზე Departments სახელზე, რაც გამოიწვევს გადასვლას EMPLOYEES გვერდზე, სადაც აისახება მხოლოდ შერჩეული განყოფილების თანამშრომლები.

13. ვაწკაპუნებთ იკონაზე შერჩეული თანამშრომლის მონაცემების დათვალიერებისა და რედაქტირებისათვის.

14. ვცადოთ მონაცემების რედაქტირება და ახალი თანამშრომლის შექმნა.

„მთავარი - დაქვემდებარებული“ ანგარიშიანი დანართის შემუშავება

განსხვავებით წინა შემთხვევისაგან, ამ დანართში დაემატება ცხრილური ფორმა, რომელიც დაქვემდებარებული ცხრილის რამდენიმე ჩანაწერის რედაქტირების საშუალებას იძლევა.

ასეთი ცხრილური ფორმის (Master–Detail Form) შესაქმნელად და მისი არსებულ გვერდზე დამატებისათვის აუცილებელია:

1. გავუშვათ დანართი დეტალური ანგარიშით. ვირჩევთ განყოფილებას და თანამშრომელს, გადავდივართ მე-3 გვერდზე. რედაქტირების ფორმის ქვემოთ ვაწკაპუნებთ ღილაკზე Edit Page 3. იხსნება რედაქტირების გვერდი, სადაც ვაწკაპუნებთ ღილაკზე Create.

2. გადავდივართ ბიჯზე Create. ვირჩევთ Region on this Page -> Next.

3. გადავდივართ ბიჯზე Region. ვირჩევთ Form -> Next.

4. გადავდივართ ბიჯზე Create. ვირჩევთ Tabular Form -> Next.

5. გადავდივართ ბიჯზე ცხრილი/წარმოდგენა Owner. ვტოვებთ მნიშვნელობას Allowed Options -> Next.

6. გადავდივართ ბიჯზე ცხრილი/წარმოდგენა Name. ვირჩევთ ცხრილი/წარმოდგენა Name მნიშვნელობას EMPLOYEES -> Next.

7. გადავდივართ ბიჯზე Displayed Columns. ვირჩევთ ყველა სვეტს -> Next.

8. გადავდივართ ბიჯზე Primary Key. ვირჩევთ Primary Key Column 1-სათვის მნიშვნელობას Employee_id -> Next.

9. გადავდივართ ბიჯზე Primary Key Source. ვირჩევთ Existing Sequence ->Sequence -სთვის ვაყენებთ მნიშვნელობას EMPLOYEES_SEQ -> Next.

10. გადავდივართ ბიჯზე Updatable Columns. ვირჩევთ ყველა სვეტს, რომლებიც არ წარმოადგენენ იდენტიფიკატორებს (არ მთავრდება _id-ით) -> Next.

11. გადავდივართ ბიჯზე Page და Region Attributes. ვადგენთ Page -სათვის მნიშვნელობას 3 -> Region Title -სათვის შეგვაქვს ტექსტი **დაქვემდებარებული**-> Next.

12. გადავდივართ ბიჯზე Button Labels -> Next.

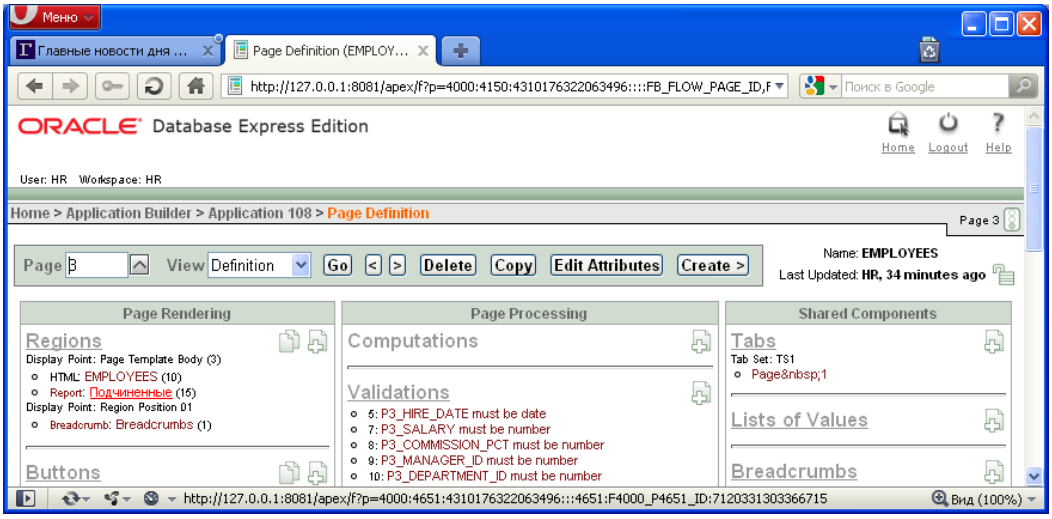
13. გადავდივართ ბიჯზე Branching. When Cancel -სათვის … ვაყენებთ მნიშვნელობას 2 -> Next.

14. გადავდივართ ბიჯზე Confirm. ვამოწმებთ დანართის დაყენებული თვისებების სისწორეს -> Finish.

15. იხსნება დანართთან მუშაობის გვერდი. ვაწკაპუნებთ იკონაზე Edit Page.

16. იხსნება მე-3 გვერდისათვის თვისებების რედაქტირების ფორმა (ნახ. 6.7.).

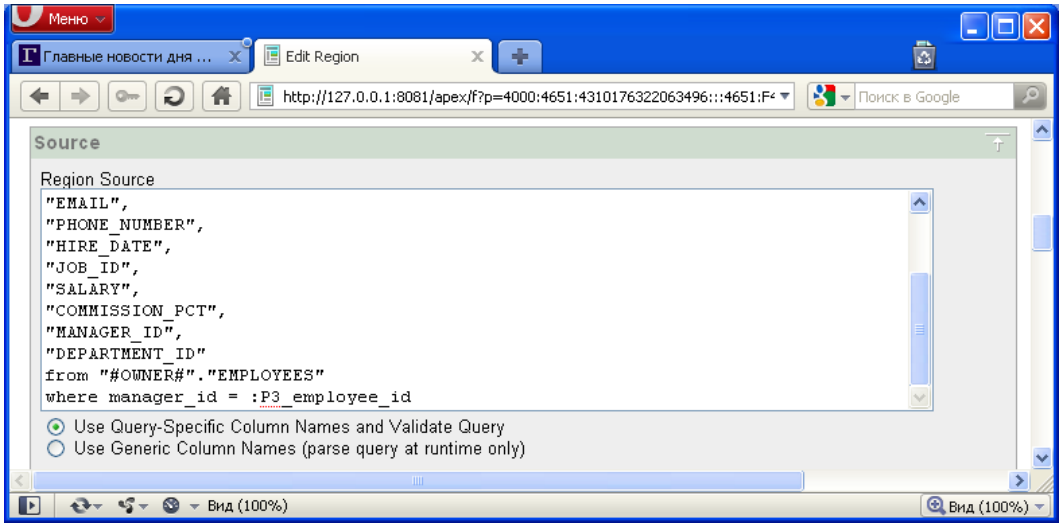
17. ვირჩევთ Page Rendering – Regions – Report: დაქვემდებარებული.



ნახ. 6.7

18. იხსნება რეგიონის რედაქტირების ფორმა.

19. გადავდივართ Source -> SQL- მოთხოვნას ვამატებთ სტრიქონს **where manager_id = :P3_employee_id**-> Apply Changes (ნახ. 8).



ნახ. 6.8

20. ამჟამად ისევ რედაქტირების ფორმის გვერდზე ვიმყოფებით (ნახ. 7). გადავიდეთ გვერდზე Application XXX და გავუშვათ დანართი შესრულებაზე (Run Application).

21. დავრწმუნდეთ, რომ დაქვემდებარებულ ცხრილში ხვდებიან მხოლოდ ის ჩანაწერები, რომლისთვისაც შერჩეული თანამშრომელი წარმოადგენს მენეჯერს. შევეცადოთ მონაცემების რედაქტირება და ახალი თანამშრომლის დამატება.

ანგარიშისადმი მოთხოვნები

ანგარიში სრულდება ტექსტურ რედაქტორში MS Word. ანგარიში უნდა შეიცავდეს:

- მცირე თეორიული აღწერა.
- ყველა გამოყენებული SQL- მოთხოვნის ტექსტები.
- მიღებული შედეგებისა და ფანჯრების *screenshot* - ები.
- დასკვნები შესრულებული სამუშაოს შესახებ.

ლაბორატორიული სამუშაო 7

PL/SQL ენის საფუძვლები

სამუშაოს მიზანი:

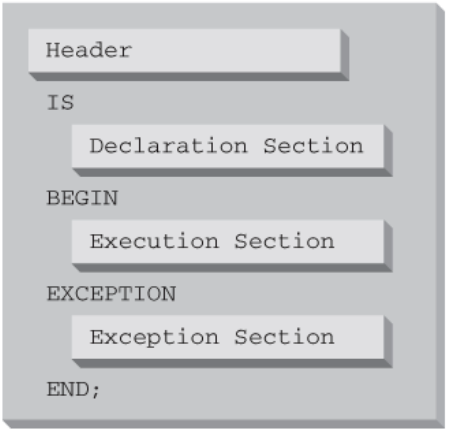
5. **PL/SQL ბლოკური სტრუქტურის შესწავლა;**
6. მარტივი პროგრამების შედგენა;
7. DML ბრძანებების გამოყენება PL/SQL-ში.

PL/SQL ბლოკური სტრუქტურა

PL/SQL – არის პროცედურული ენა SQL-ში (Procedural Language extensions to the Structured Query Language).

PL/SQL პროგრამის ძირითადი ერთეულია - ბლოკი. PL/SQL ყველა პროგრამა შედგება ბლოკებისაგან, რომლებიც შესაძლოა ერთმანეთში იყოს ჩასმული. თითოეულ ბლოკს გააჩნია შემდეგი სტრუქტურა:

Header
Declaration section
Execution section
Exception section



ნახ. 7.1

```
DECLARE
/* ცვლადების, ტიპებისა და ქვეპროგრამების გამოცხადების (აღწერის) განყოფილება */
BEGIN
/* შესასრულებელი კოდის განყოფილება: პროცედურული და SQL ენების ბრძანებები */
```

/* ეს ერთდათერთი აუცილებელი განყოფილებაა */

EXCEPTION

/* განსაკუთრებულ სიტუაციათა დამუშავების განყოფილება: შეცდომების დამუშავების ბრძანებები. */

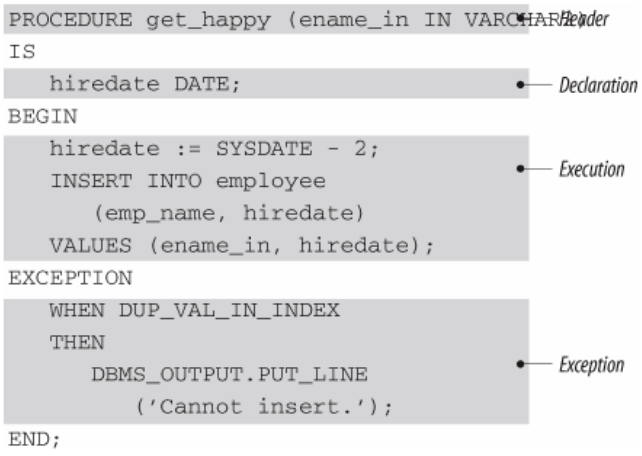
END;

განირჩევა ბლოკების შემდეგი ტიპები: ანონიმური, პროცედურა და ფუნქცია. ანონიმური ბლოკი არ ინახება მონაცემთა ბაზაში.

PL/SQL პროგრამის შესასრულებელ კოდში შეიძლება არსებობდეს SQL-ის ბრძანებები - SELECT, INSERT, UPDATE, DELETE, აგრეთვე მონაცემთა დამუშავებისა და ტრანზაკციების მართვის სხვა ბრძანებები.

პროცედურა შეიცავს ოთხივე სექციას, ოღონდ განსხვავება იმაშია, რომ შესრულების ბლოკის დაწყება შეიცავს გასაღებურ სიტყვას PROCEDURE, ხოლო დამთავრებისათვის, როგორც ყველა ბლოკი გამოიყენება გასაღებური სიტყვა END.

პროცედურის მაგალითი:



ნახ. 7.2

ანონიმური ბლოკები

PL/SQL ანონიმური ბლოკის სინტაქსისი შემდეგია:

[DECLARE

... declaration statements ...]

BEGIN

... one or more execuცხრილი statements ...

[EXCEPTION

... exception hდsler statements ...]

END;

კვადრატული ფრჩხილები მიუთითებენ სინტაქსის არააუცილებელ ნაწილზე. ინსტრუქციები BEGIN და END კი აუცილებელია. მაგალითად:

```
BEGIN
DBMS_OUTPUT.PUT_LINE(SYSDATE);
END;
```

ფუნქციონალურად მსგავსი ბლოკი, რომელიც ამატებს გამოცხადების განყოფილებას:

```
DECLARE
l_right_now VARCHAR2(9);
BEGIN
l_right_now := SYSDATE;
DBMS_OUTPUT.PUT_LINE (l_right_now);
END;
```

იგივე ბლოკი გამონაკლისი ოპერატორის (EXCEPTION) ჩართვით:

```
DECLARE
l_right_now VARCHAR2(9);
BEGIN
l_right_now := SYSDATE;
DBMS_OUTPUT.PUT_LINE (l_right_now);
EXCEPTION
    WHEN VALUE_ERROR
    THEN
      DBMS_OUTPUT.PUT_LINE('I bet l_right_now is too small '
        || 'for the default date format!')
END;
```

სახელდებული ბლოკები

პროცედურის header ასე გამოიყურება:

PROCEDURE [*schema.*]*name* [(*parameter* [, *parameter ...*])]
[AUTHID {DEFINER | CURRENT_USER}]

ფუნქციის header-ს გააჩნია მსგავსი სინტაქსი, ოღონდ შეიცავს გასაღებურ სიტყვას RETURN :

FUNCTION [*schema.*]*name* [(*parameter* [, *parameter ...*])]

```
RETURN return_datatype
[AUTHID {DEFINER | CURRENT_USER}]
[DETERMINISTIC]
[PARALLEL ENABLE ...]
[PIPELINED [USING...] | AGGREGATE USING...]
```

ჩასმული ბლოკები

PL/SQL ამ მაგალითში პროცედურა შეიცავს ერთ ანონიმურ ე.წ. ჩასმულ ბლოკს:

```
PROCEDURE calc_totals
IS
  year_total NUMBER;
BEGIN
  year_total := 0;
  /* Beginning of nested block */
  DECLARE
    month_total NUMBER;
  BEGIN
    month_total := year_total / 12;
  END set_month_total;
  /* End of nested block */
END;
```

გამყოფი /* და */ მიუთითებს კომენტარებზე. ასევე შესაძლებელია ჩასმულ ანონიმურ ბლოკებში კიდევ სხვა (შვილობილი ან ქვებლოკი) ანონიმური ბლოკების არსებობა.

სამდონიანი სიღრმის ჩასმული ანონიმური ბლოკების მაგალითი:

```
DECLARE
   CURSOR emp_cur IS ...;
BEGIN
   DECLARE
      total_sales NUMBER;
   BEGIN
      DECLARE
         hiredate DATE;
      BEGIN
         ...
      END;
   END;
END;
```

ნახ. 7.3

PL/SQL სიმბოლოების სიმრავლე

PL/SQL პროგრამა შეიცავს ინსტრუქციების თანამიმდევრობას, რომელიც შედგება ტექსტის ერთი ან მეტი სტრიქონისაგან. ყოველი გასაღებური სიტყვა, ოპერატორი თუ მონაცემი წარმოადგენს ნიშანი-სიმბოლოების სხვადასხვა კომბინაციას შემდეგ სიმბოლოთა სიმრავლეში.

Type	Characters
Letters	A-Z, a-z
Digits	0-9
Symbols	~ ! @ # $ % * ( ) _ - + = | : ; " ' < > , . ? / ^
Whitespace	Tab, space, newline, carriage return

და კიდევ, რამდენიმე PL/SQL უბრალო ნიშნები: & { } [ ]

ქვემოთ მოყვანილია ზოგიერთი სპეციალური სიმბოლოს დანიშნულება .

Symbol	Description
;	Semicolon: terminates declarations and statements
%	Percent sign: attribute indicator (cursor attributes like %ISOPEN and indirect declaration attributes like %ROWTYPE); also used as multibyte wildcard symbol with the LIKE condition
_	Single underscore: single-character wildcard symbol in LIKE condition
@	At-sign: remote location indicator
:	Colon: host variable indicator, such as :block.item in Oracle Forms
**	Double asterisk: exponentiation operator
< > or != or ^= or ~=	Ways to denote the "not equal" relational operator
||	Double vertical bar: concatenation operator
<< and >>	Label delimiters
<= and >=	Less than or equal, greater than or equal relational operators
:=	Assignment operator
=>	Association operator for positional notation
..	Double dot: range operator
--	Double dash: single-line comment indicator
/* and */	Beginning and ending multiline comment block delimiters

PL/SQL-ში ასოები შეიძლება დავაჯგუფოთ შემდეგ ლექსიკურ ერთეულებად :

- Identifier
- Literal
- Delimiter
- Comment

იდენტიფიკატორები (Identifiers)

PL/SQL ობიექტის იდენტიფიცირებისათვის გამოიყენება მისი სახელი, რომელიც შეიცავს შემდეგს:

- Constant or variable
- Exception
- Cursor
- Program name: procedure, function, package, object type, trigger, etc.
- Reserved word
- Label

PL/SQL იდენტიფიკატორები:

- უნდა იყოს სიგრძით არაუმეტეს 30 სიმბოლოსი
- უნდა იწყებოდეს ასოთი
- შეიძლება შეიცავდეს $ (dollar sign), _ (underscore) და # (pound sign)
- არ უნდა შეიცავდეს ცარიელი სივრცის სიმბოლოს ("whitespace").

მაგალითად:

```
SQL> DECLARE
  2    "pi" CONSTANT NUMBER := 3.141592654;
  3    "PI" CONSTANT NUMBER := 3.14159265358979323846;
  4    "2 pi" CONSTANT NUMBER := 2 * "pi";
  5  BEGIN
  6    DBMS_OUTPUT.PUT_LINE('pi: ' || "pi");
  7    DBMS_OUTPUT.PUT_LINE('PI: ' || pi);
  8    DBMS_OUTPUT.PUT_LINE('2 pi: ' || "2 pi");
  9* END;
 10 /
          pi: 3.141592654
          PI: 3.14159265358979323846
          2 pi: 6.283185308
```

PL/SQL -ში იდენტიფიკატორებად არ შეიძლება ე.წ. დარეზერვებული სიტყვების გამოყენება.

მაგალითი:

```
DECLARE
  dup_val_on_index EXCEPTION;  -- local re-declaration
BEGIN
  ...
  INSERT INTO ...  /* may raise the built-in exception */
  ...
  RAISE dup_val_on_index;  -- resolves to the locally declared
exception
EXCEPTION
  WHEN dup_val_on_index
  THEN
    /* hდale the locally declared exception */
  ...
  WHEN STდაARD.DUP_VAL_ON_INDEX
  THEN
    /* hდale the usual exception */
  ...
END;
```

ლიტერალები

ლიტერალი არის მნიშვნელობა, რომელიც იდენტიფიკატორის მიერ არ იქნება წარმოდგენილი, ანუ ის არის უბრალოდ მნიშვნელობა. მაგალითად:

Number

415, 21.6, 3.141592654f, 7D, NULL

String

'This is my sentence', '01-OCT-2006', q'!hello!', NULL

Time interval

INTERVAL '25-6' YEAR TO MONTH, INTERVAL '-18' MONTH, NULL

Boolean

trUE, FALSE, NULL

NULLs

მნიშვნელობის არ არსებობა წარმოდგება გასაღებური სიტყვის NULL საშუალებით.

მაგალითად:

```
DECLARE
  str VARCHAR2(1) := '';
BEGIN
  IF str IS NULL   -- will be TRUE
```

წერტილ-მძიმე გამყოფი (Semicolon Delimiter)

წერტილ-მძიმე გამყოფი (;) დაისმება არა ფიზიკური ხაზის, არამედ თითოეული ლოგიკური თუ შესრულებადი ინსტრუქციის ბოლოს. მაგალითად:

```
IF salary < min_salary (2003)
THEN
  salary := salary + salary * .25;
END IF;
```

იგივე ინსტრუქცია შეიძლება განთავსდეს ერთ ხაზზეც:

```
IF salary < min_salary (2003) THEN salary := salary + salary*.25; END IF;
```

გასაღებური სიტყვა PRAGMA

პროგრამირების ეს ცნება იწარმოება ბერძნული სიტყვიდან „pragma“, რაც ქმედება-ქცევას ნიშნავს. PL/SQL-ში PRAGMA -ს გააჩნია შემდეგი სინტაქსი:

```
PRAGMA instruction_to_compiler;
```

ეტიკეტი ანუ იარლიყი (Labels)

PL/SQL-ში label არის პროგრამის ცალკეული ნაწილისათვის სახელის მინიჭების საშუალება. სინტაქსურად label-ს აქვს ფორმატი:

```
<<identifier>>
```

label არის აგრეთვე ანონიმური ბლოკის სახელი მისი შესრულების მთელი ხნის განმავლობაში. მაგალითად:

```
<<insert_but_ignore_dups>>
BEGIN
  INSERT INTO catalog
```

```
  VALUES (...);
EXCEPTION
  WHEN DUP_VAL_ON_INDEX
  THEN
    NULL;
END insert_but_ignore_dups;
```

PL/SQL მარტივი პროგრამები

PL/SQL პროგრამის შესრულებისათვის პროგრამის ტექსტის ბოლო ორ სტრიქონში უნდა იყოს ჩართული:

- სტრიქონი სიმბოლოთი "წერტილი" (.)
- სტრიქონი პროგრამის გამშვები ბრძანებით run;

PL/SQL პროგრამები, ისევე როგორც SQL პროგრამები, შეიძლება უშუალოდ შევიტანოთ sqlplus-ში ან მოვათავსოთ ფაილში და გავუშვათ sqlplus-ში.

ყველაზე მარტივი პროგრამა შეიცავს რამდენიმე გამოცხადებას და შესასრულებელი კოდის განყოფილებას, რომელიც შედგება ერთი ან რამდენიმე SQL ბრძანებისაგან. მთავარი თავისებურება იმაში მდგომარეობს, რომ SELECT ბრძანების ფორმატი განსხვავდება ფორმატისაგან SQL-ში. SELECT-ის შემდეგ უნდა იყოს პარამეტრი INTO ცვლადების სიით SELECT ბრძანების ყოველი სვეტისათვის, რომელშიც უნდა იყოს მოთავსებული ნაპოვნი სტრიქონის (კორტეჟის) მნიშვნელობები.

PL/SQL-ში ბრძანება SELECT სრულდება მხოლოდ მაშინ, თუ მოთხოვნის შედეგი შეიცავს ერთ სტრიქონს (კორტეჟს). იმ შემთხვევაში, თუ მოთხოვნა აბრუნებს ერთზე მეტ სტრიქონს (კორტეჟს), მაშინ აუცილებელია კურსორის გამოყენება.

მაგალითად:

```
CREATE ცხრილი T1(
  e INTEGER,
  f INTEGER
);

DELETE FROM T1;
```

```
INSERT INTO T1 VALUES(1, 3);
NSERT INTO T1 VALUES(2, 4);
/* რაც ზემოთ არის დაწერილი – ეს უბრალოდ SQL; შემდეგ
იქნება - PL/SQL პროგრამა. */
DECLARE
  a NUMBER;
  b NUMBER;
BEGIN
  SELECT e,f INTO a,b FROM T1 WHERE e>1;
  INSERT INTO T1 VALUES(b,a);
END;
.
run;
```

განშტოებათა და ციკლების ორგანიზაცია PL/SQL-ში

PL/SQL განშტოებათა და ციკლების ორგანიზების საშუალებას იძლევა ცნობილი ხერხებით.

IF ბრძანებას გააჩნია შემდეგი ფორმატი:

IF <პირობა> THEN <ბრძანებათა სია> ELSE <ბრძანებათა სია> END IF;

ELSE – არააუცილებელი ნაწილია.

ჩასმული განშტოებებისათვის გამოიყენება ბრძანება:

```
IF <პირობა_1> THEN ...
ELSIF <პირობა_2> THEN ...
... ...
ELSIF <პირობა_n> THEN ...
ELSE ...
END IF;
```

მაგალითად:

```
DECLARE
  a NUMBER;
  b NUMBER;
BEGIN
  SELECT e,f INTO a,b FROM T1 WHERE e>1;
  IF b=1 THEN
    INSERT INTO T1 VALUES(b,a);
```

```
    ELSE
      INSERT INTO T1 VALUES(b+10,a+10);
    END IF;
  END;

  run;
```

ციკლების ორგანიზება ხდება შემდეგი ბრძანებით:

```
  LOOP
    <ციკლის_ტანი> /* ბრძანებათა სია. */
  END LOOP;
```

სულ მცირე ერთ-ერთი ბრძანება <ციკლის_ტანი> მაინც უნდა შეიცავდეს ბრძანებას EXIT ფორმატით:

```
  EXIT WHEN <პირობა>;
```

ციკლი შეწყდება, თუ <პირობა> - ჭეშმარიტია.

მაგალითად:

```
  DECLARE
    i NUMBER := 1;
  BEGIN
    LOOP
      INSERT INTO T1 VALUES(i,i);
      i := i+1;
      EXIT WHEN i>100;
    END LOOP;
  END;
  .
  run;
```

ციკლების ორგანიზაციის დამატებითი ბრძანებები:

- ბრძანება EXIT წყვეტს ციკლს პირობის გარეშე. ეს ბრძანება შეიძლება გამოვიყენოთ განშტოების ბრძანებაში.
- ციკლის ბრძანებაWHILE:
- WHILE <პირობა> LOOP
- <ციკლის_ტანი>

– END LOOP;
– ციკლის ბრძანება FOR:

```
FOR <ცვლადი> IN
<საწყ.მნიშვნელობა>..<საბოლ.მნიშვნელობა> LOOP
  <ციკლის_ტანი>
END LOOP;
```

DML PL/SQL-ში

PL/SQL კოდის ნებისმიერ ბლოკში ჩვენ შეგვიძლია გამოვიყენოთ DML ბრძანებები (INSERT, UPDATE და DELETE).

INSERT ბრძანება

აქ წარმოდგენილია INSERT ბრძანების სინტაქსისის ორი ძირითადი ტიპი:

ერთი სტრიქონის შეტანის შემთხვევაში:

```
INSERT INTO ცხრილი [(col1, col2, ..., coln)]
  VALUES (val1, val2, ..., valn);
```

ერთი ან მეტი სტრიქონის შეტანის შემთხვევაში ცხრილში, რომელიც განსაზღვრულია SELECT ბრძანებით ერთი ან რამდენიმე ცხრილიდან:

```
INSERT INTO ცხრილი [(col1, col2, ..., coln)]
AS
  SELECT ...;
```

მაგალითი:

```
BEGIN
  INSERT INTO book
     VALUES ('1-56592-335-9',
       'Oracle PL/SQL Programming',
       'Reference for PL/SQL developers,' ||
       'including examples და best practice ' ||
       'recommendations.',
       'Feuerstein,Steven, with Bill Pribyl',
       TO_DATE ('01-SEP-1997','DD-MON-YYYY'),
       987);
END;
```

ჩვენ შეგვიძლია ჩამოვთვალოთ სვეტების სახელები როგორც ცვლადები:

```
DECLARE
  l_isbn book.isbn%TYPE := '1-56592-335-9';
  ... other declarations of local variables
BEGIN
  INSERT INTO books (
     isbn, title, summary, author,
     date_published, page_count)
    VALUES (
      l_isbn, l_title, l_summary, l_author,
      l_date_published, l_page_count);
```

UPDATE ბრძანება

ჩვენ შეგვიძლია გამოვიყენოთ UPDATE ბრძანება ერთი ან რამდენიმე სვეტში ან სტრიქონში მონაცემების ცვლილებისათვის შემდეგი სინტაქსით:

```
UPDATE ცხრილი
  SET col1 = val1
     [, col2 = val2, ... colN = valN]
[WHERE where clause];
```

სადაც: WHERE წესი არასავალდებულოა;

თუ არ მივუთითებთ ერთ რომელიმე სტრიქონს, მაშინ ყველა სტრიქონი იქნება შეცვლილი.

მაგალითი:

```
UPDATE books SET title = UPPER (title);
```

მაგალითი:

```
CREATE OR REPLACE PROCEDURE remove_time (
  author_in IN VARCHAR2)
IS
BEGIN
  UPDATE books
    SET title = UPPER (title),
       date_published =
          TRUNC (date_published)
   WHERE author LIKE author_in;
```

```
END;
```

DELETE ბრძანება

ჩვენ შეგვიძლია ამოვაგდოთ ერთი, რამდენიმე ან ყველა სტრიქონი ცხრილში DELETE ბრძანების გამოყენებით, რომლის სინტაქსი შემდეგია:

```
DELETE FROM ცხრილი
 [WHERE where-clause];
```

სადაც: WHERE წესი არასავალდებულოა;

თუ არ მივუთითებთ ერთ რომელიმე სტრიქონს, მაშინ ყველა სტრიქონი იქნება ამოგდებული.

მაგალითი:

```
DELETE FROM books;
```

მაგალითი:

```
CREATE OR REPLACE PROCEDURE remove_books (
date_in  IN  DATE,
removal_count_out   OUT      PLS_INTEGER)
IS
BEGIN
DELETE FROM books WHERE date_published < date_in;
removal_count_out := SQL%ROWCOUNT;
END;
```

ლაბორატორიული სამუშაო 8

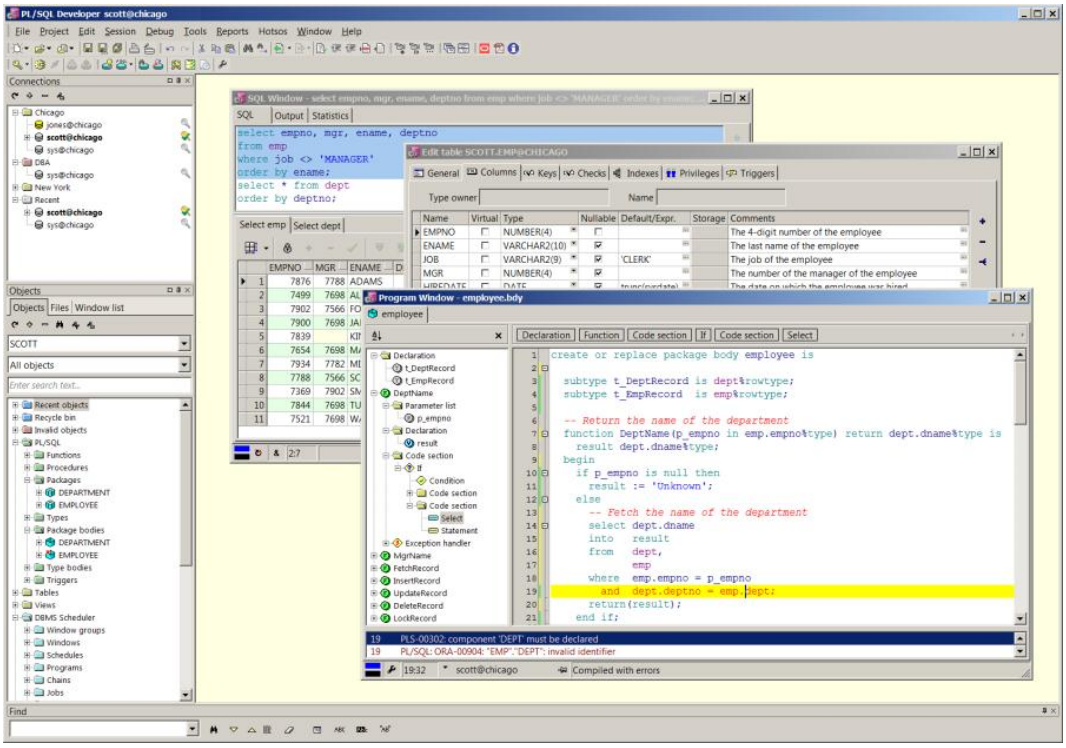
მუშაობა PL/SQL Developer-ში

სამუშაოს მიზანი:

1. PL/SQL Developer - ის მოკლე მიმოხილვა;
2. ობიექტებთან მუშაობის შესწავლა;
3. DBMS დამგეგმავის გამოყენების შესწავლა;
4. მოთხოვნისა და ანგარიშების ამგების გამოყენების შესწავლა;
5. პროექტების შექმნის შესწავლა;
6. პროგრამირება PL/SQL Developer- ში.

PL/SQL Developer – ის მოკლე მიმოხილვა

PL/SQL Developer IDE. PL/SQL Developer წარმოადგენს ინტეგრირებულ გარემოს (Integrated Development Environment - IDE) Oracle მონაცემთა ბაზის პროგრამული ერთეულების შემუშავებისათვის.



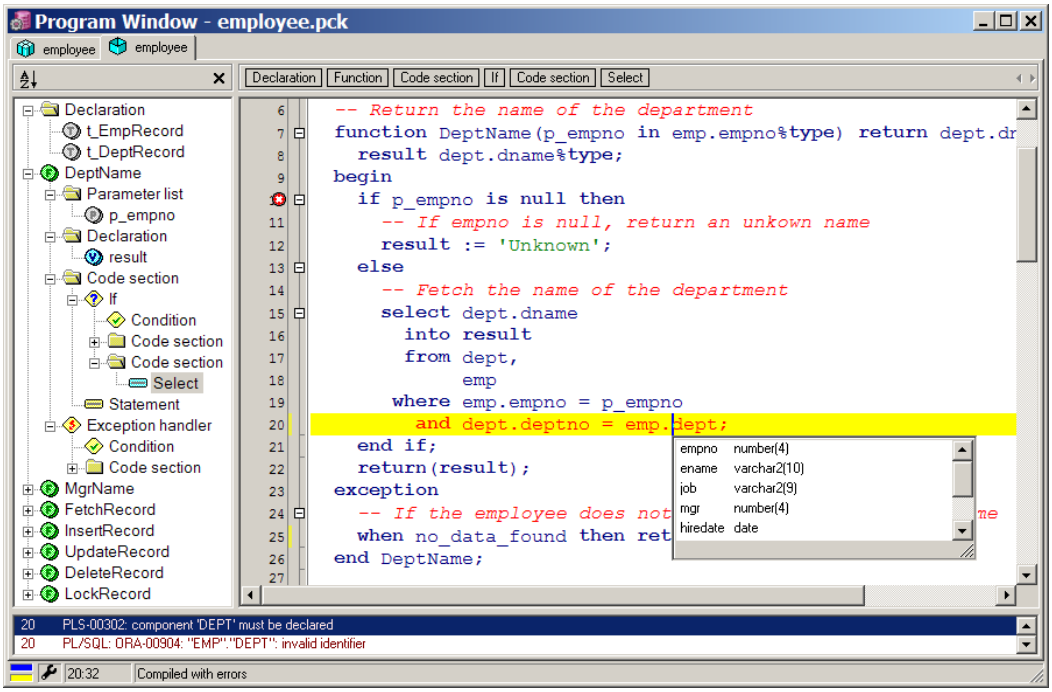
ნახ. 8.1

PL/SQL Developer-ის გამოყენება მნიშვნელოვნად ამარტივებს თქვენი კლიენტ/სერვერული აპლიკაციის სერვერული ნაწილის შექმნას.

PL/SQL Developer IDE-ის მარცხენა მხარეს ჩვენ ვხედავთ Connection List, რომელიც ჩვენი შეერთებების მართვის საშუალებას იძლევა და Object Browser, რომელიც მონაცემთა ბაზის ყველა ობიექტთან წვდომას ახორციელებს. მარჯვენა მხარეს არის Program Editor, ცხრილი Definition Editor და SQL Window სამუშაო სივრცე.

გარდა ამისა, მარცხენა მხარეს არსებობს მრავალი ინსტრუმენტის ჩართვის შესაძლებლობა, როგორიცაა Connection List, Object Browser, File Browser, შაბლონი List დაWindow List, ხოლო ეკრანის ქვედა ნაწილში - Search Bar. მოკლედ მიმოვიხილოთ ისინი.

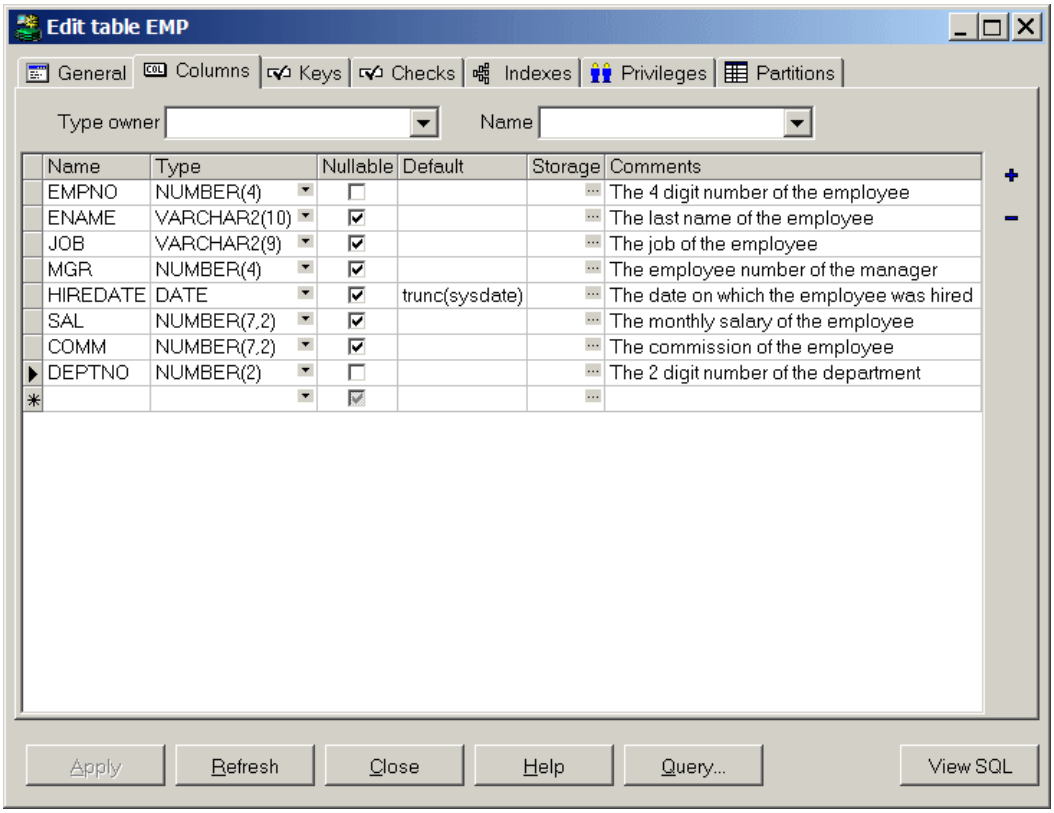
PL/SQL Editor. PL/SQL Editor გვთავაზობს შემდეგი სერვისების მხარდაჭერას: *Help, Object Popup Menu, Code Assistant, Special Paste და Copy From Host Language, Compiler Hints, Highlighting, Refactoring, PL/SQL Beautifier, Code Contents, Code Hierarchy, Code Folding, Hyperlink Navigation Compare To Web Search და სხვ.*



ნახ. 8.2

PL/SQL Editor შეიძლება აგრეთვე გამოყენებულ იქნას Oracle მონაცემთა ბაზაში Java Source კოდის რედაქტირებისათვის.

Non-PL/SQL Objects. შემუშავების პროცესში ხშირად საჭირო ხდება სხვადასხვა ობიექტების ნახვა, ცვლილებები ან შექმნა, რაც შესაძლებელია SQL გამოყენების გარეშე. კერძოდ, ობიექტზე მარჯვენა დაწკაპუნებით Object Browser - ში ან popup მენიუდან ვირჩევთ 'წარმოდგენა' ან 'Edit'.



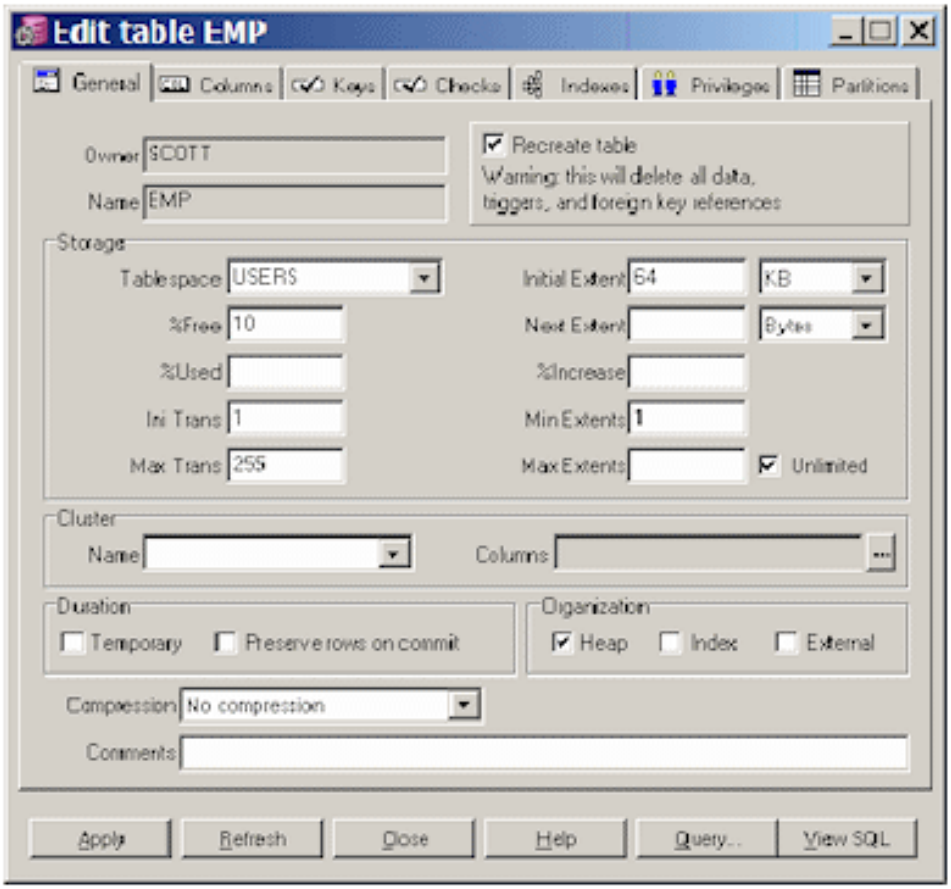
ნახ. 8.3

ახალი ობიექტის შექმნის ან არსებულ ობიექტში ცვლილებების შეტანის შემდეგ, შესაძლებელია ამ ქმედებათა შემსრულებელი SQL ბრძანებების ნახვა, რედაქტირება, კოპირება, შენახვა ან Command Window - ში ექსპორტირება. ასევე შესაძლებელია SQL კოდის შესრულებაზე გაშვება 'Apply' ღილაკზე დაჭერით.

ობიექტებთან მუშაობა

ობიექტების შექმნა და მოდიფიკაცია

ცხრილის განსაზღვრის რედაქტორი. ცხრილის განსაზღვრის რედაქტორს გააჩნია 7 გვერდი (ჩანართი):



ნახ. 8.4

რედაქტორის ქვედა ნაწილში არის 6 ღილაკი:

- **Apply** – ახორციელებს ყველა ცვლილებას მონაცემთა ბაზაში.
- **Refresh** – აღადგენს ცვლილებამდელ მნიშვნელობას.
- **Close** – ხურავს რედაქტორის ფანჯარას.
- **Help** – აჩვენებს დახმარებას რეალურ დროში.
- **Query** – ხსნის ფანჯარას მოთხოვნის ნახვის ან რედაქტირებისათვის.

- **წარმოდგენა SQL** – ეკრანზე გამოაქვს SQL ბრძანებების ტექსტი ნახვის ან ცვლილებების მიზნით.

General page

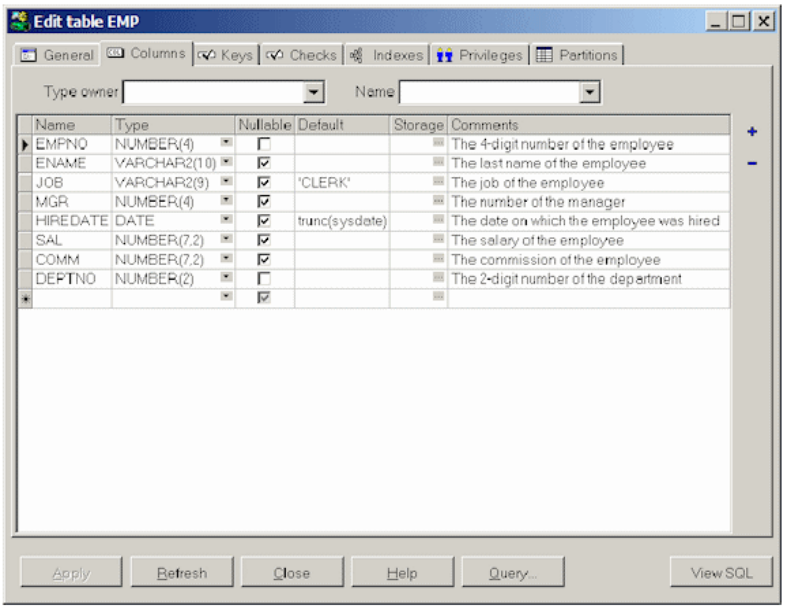
მთავარი გვერდი (*General*) შეიცავს ცხრილის მფლობელს (owner), სახელს და სხვა ინფორმაციებს:

- Owner – მიმდინარე მომხმარებელი.
- ცხრილი space – მიმდინარე მომხმარებლის ცხრილისათვის განკუთვნილი სივრცის მნიშვნელობა უსიტყვოდ.
- %Free – 10
- %Used – 40
- Initial transaction entries (შემავალი ტრანზაკციების საწყისი რაოდენობა) – 1
- Maximum transaction entries (შემავალი ტრანზაკციების მაქსიმალური რაოდენობა) – 255

თუ გვსურს რომელიმე თვისების შეცვლა საჭიროა ცხრილის ფანჯრის თავში *Recreate* ოპციის მოძებნა, წინააღმდეგ შემთხვევაში შეიძლება დაიკარგოს მონაცემები.

Columns page

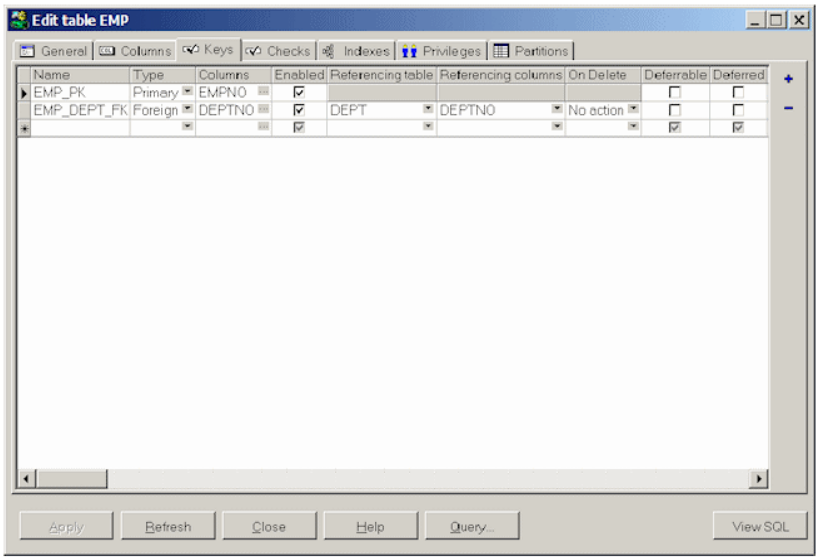
სვეტების გვერდზე (*Columns*) შეგვიძლია ცხრილის სვეტების ნახვა, დამატება, წაშლა, გადატანა და მოდიფიცირება:



ნახ. 8.5

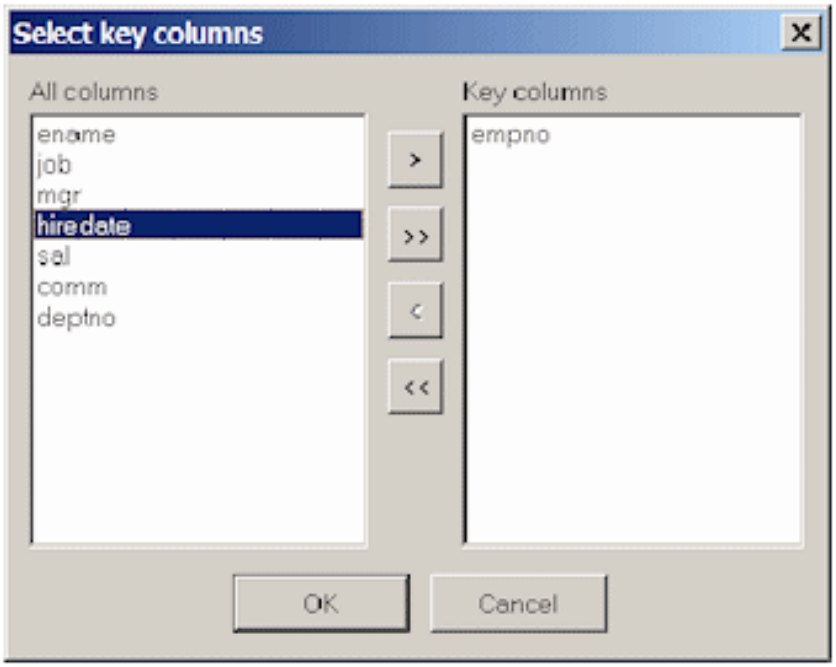
Keys page

გასაღებების (*Keys*) გვერდზე შეგვიძლია გასაღებების ნახვა, დამატება, წაშლა და მოდიფიცირება:



ნახ. 8.6

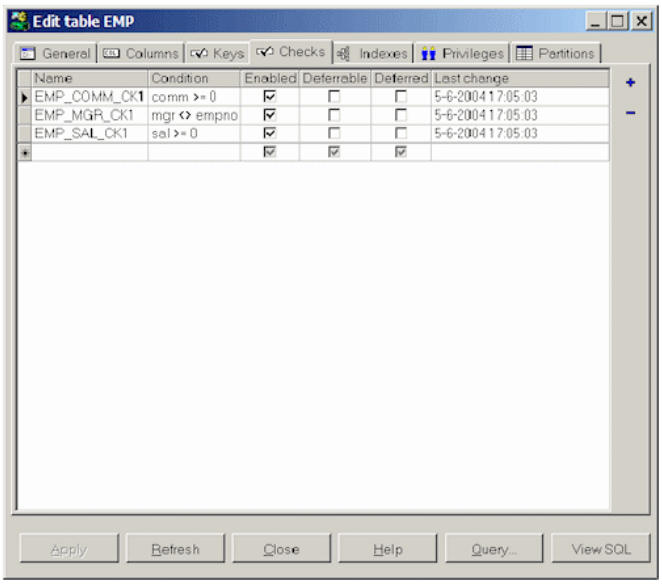
სვეტების განსაზღვრა ხდება შემდეგ ფანჯარაში:



ნახ. 8.7

Checks page

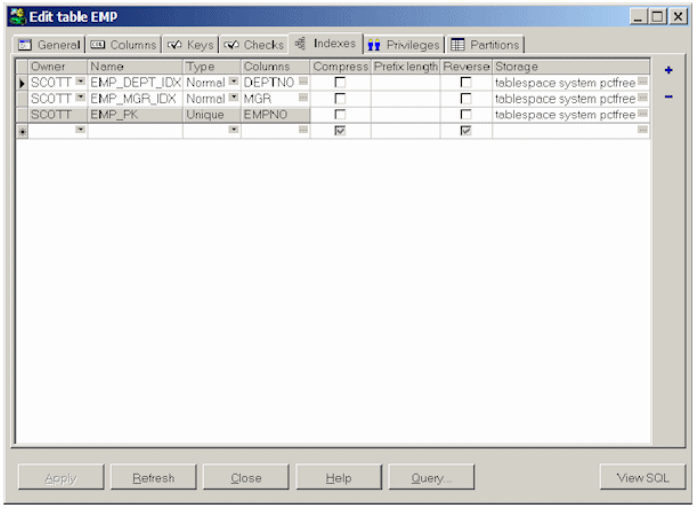
check შეზღუდვების ნახვა, დამატება, წაშლა და მოდიფიცირება შესაძლებელია *Checks* გვერდზე:



ნახ. 8.7

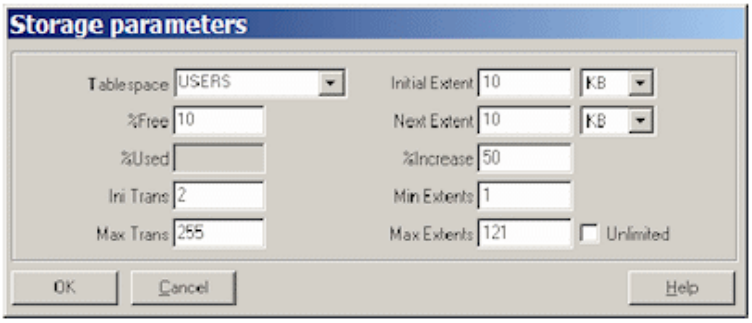
Indexes page

ინდექსების ნახვა, დამატება, წაშლა და მოდიფიცირება შესაძლებელია *Indexes* გვერდზე:



ნახ. 8.8

Storage პარამეტრების განსაზღვრა ან ნახვა შეიძლება შესაბამისი უჯრედის ღილაკზე დაჭერით. შედეგად გამოჩნდება storage რედაქტორი პარამეტრებით:

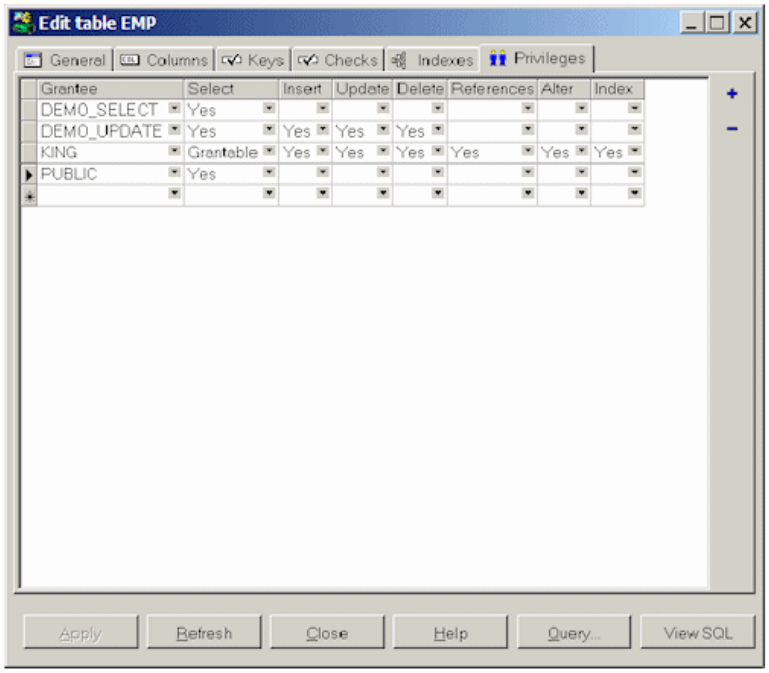


ნახ. 8.9

– ცხრილი space – მიმდინარე მომხმარებლის ცხრილისათვის განკუთვნილი სივრცის მნიშვნელობა უსიტყვოდ.
– %Free – 10
– Initial transaction entries (შემავალი ტრანზაკციების საწყისი რაოდენობა) – 2
– Maximum transaction entries (შემავალი ტრანზაკციების მაქსიმალური რაოდენობა) – 255

Privileges page

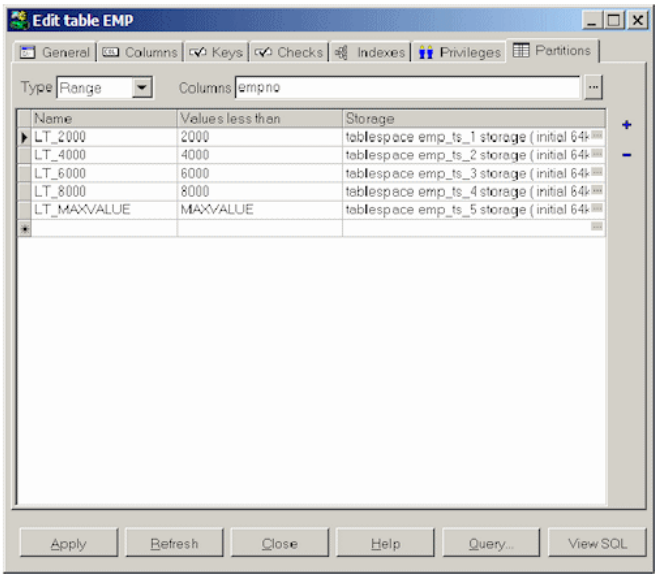
პრივილეგიების (*Privileges*) გვერდზე შესაძლებელია პრივილეგიების მინიჭება/მოხსნა (grant/revoke) მომხმარებლისათვის (ან მომხმარებლიდან) და როლებისათვის:



ნახ. 8.10

Partitions page

Partitions გვერდი არის მხოლოდ ნახვისათვის, როცა ხდება ცხრილის (re)create, ან როცა ხდება მისი რედაქტირება:

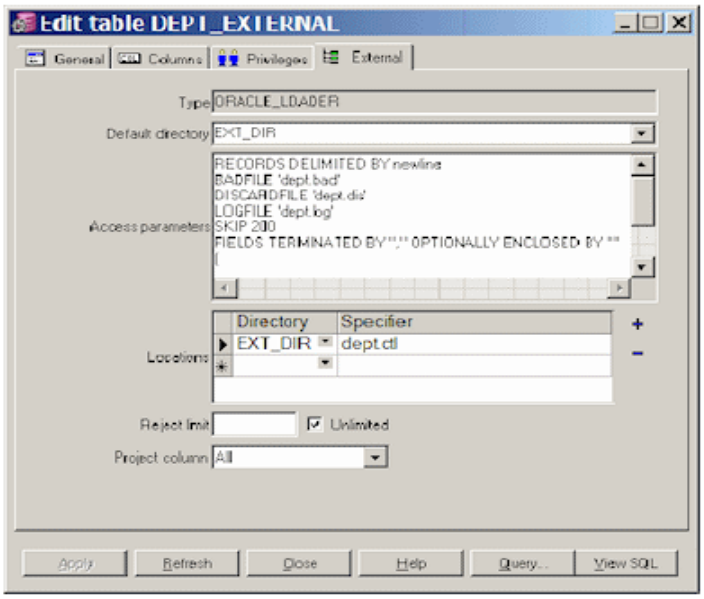


ნახ. 8.11

გვერდის თავში შეგვიძლია ავირჩიოთ *Range*, *Hash* ან *List*.

External page

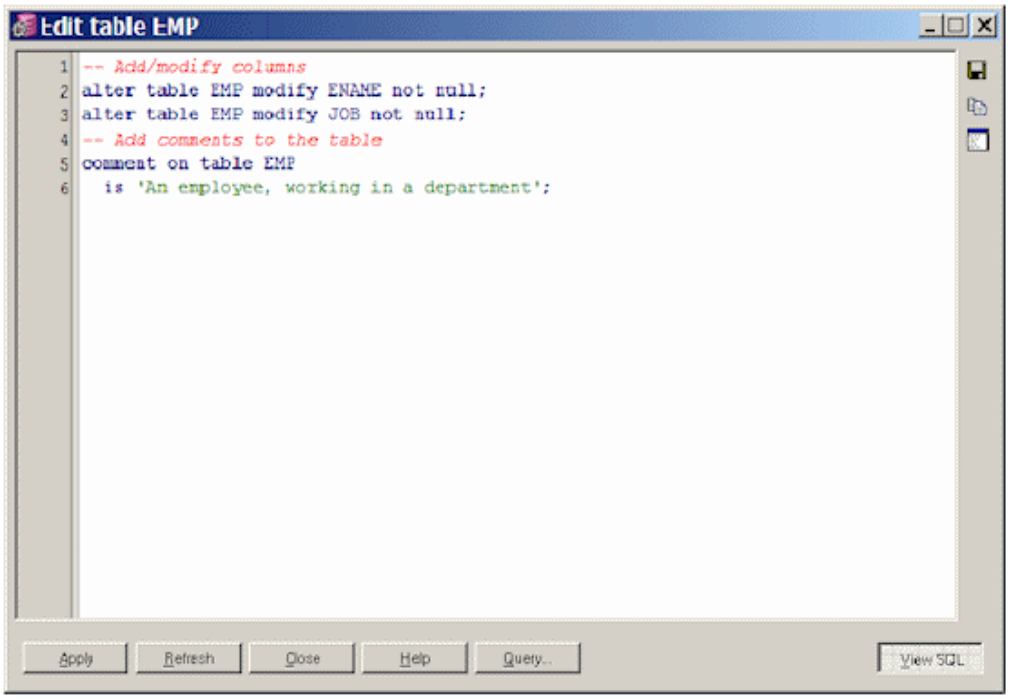
ცხრილის როგორც *External* ორგანიზება შესაძლებელია მთავარ გვერდზე, რის შემდეგაც ცხრილი ხდება ხილული:



ნახ. 8.12

SQL-ის ნახვა და მოდიფიკაცია

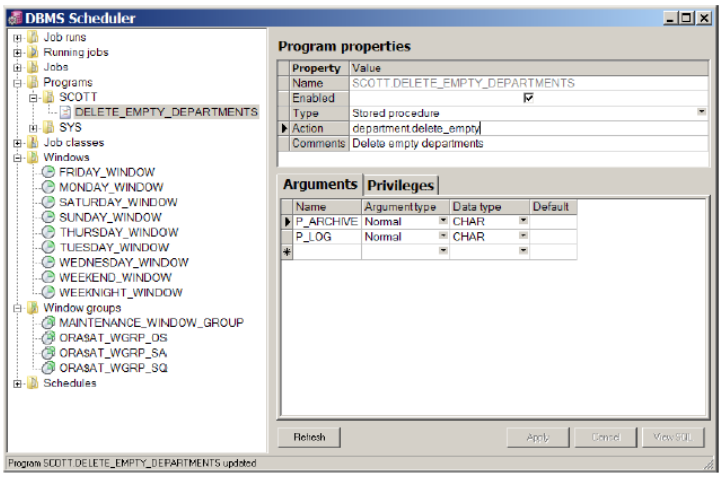
ცხრილის შექმნის ან არსებულის მოდიფიცირების შემდეგ *წარმოდგენა SQL* ღილაკზე დაჭერით შესაძლებელია SQL შედეგების ნახვა:



ნახ. 8.13

DBMS დამგეგმავი

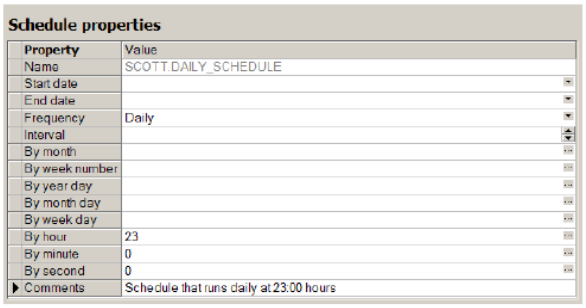
DBMS დამგეგმავის (DBMS_SCHEDULER) გამოყენება ხდება *Tools* მენიუდან ან right-ვაწკაპუნებთ ობიექტის დამგეგმავზე Object Browser-ში და ვირჩევთ *Edit* ან *view*-ს popup მენიუში. შესაბამისი Object Browser-ის კატალოგებია *Windows*, *Window groups*, *Schedules*, *Programs*, *Jobs* და *Job classes*. DBMS დამგეგმავის გაშვების შემდეგ ეკრანზე ჩნდება შემდეგი ფანჯარა:



ნახ. 8.14

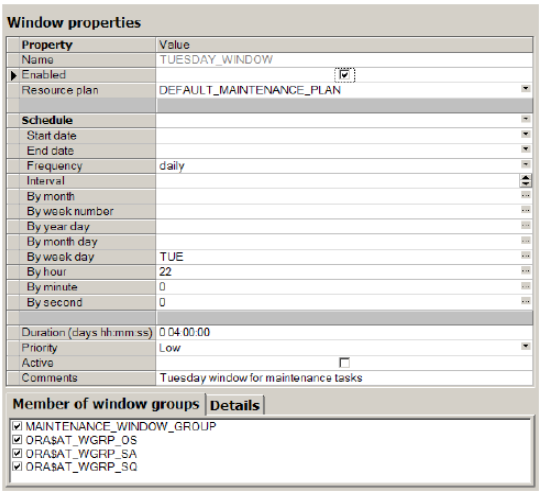
ვნახოთ Object Bowser-ის ფოლდერების *Windows*, *Window groups*, *Schedules*, *Programs*, *Jobs* და *Job classes* შესაბამისი ფანჯრები.

Schedules



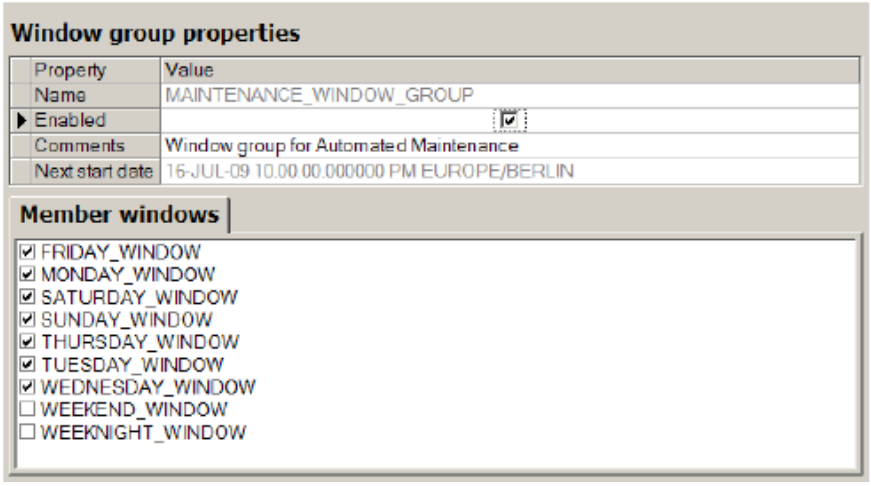
ნახ. 8.15

Windows



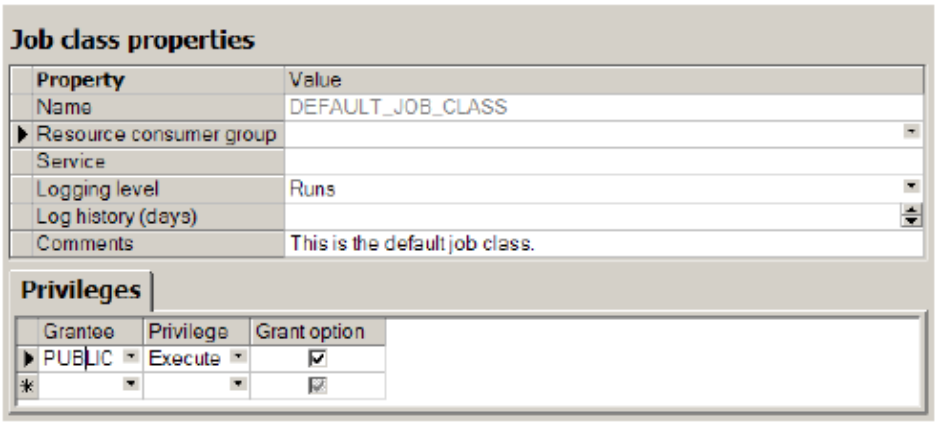
ნახ. 8.16

Window groups



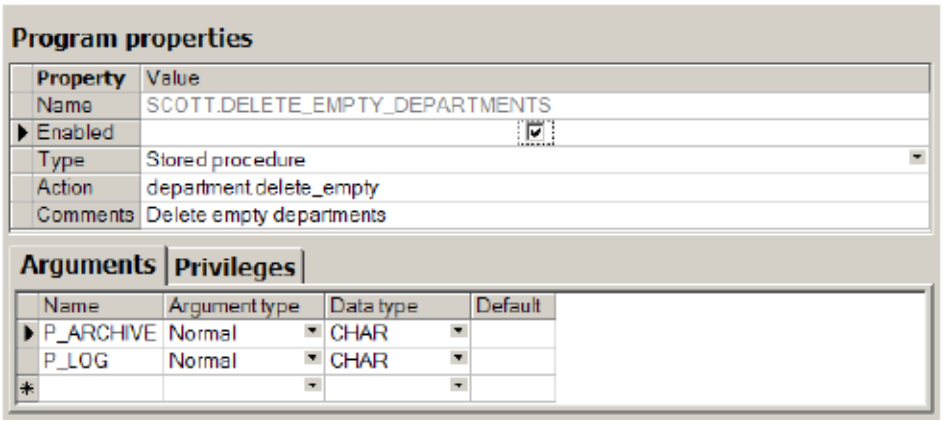
Бзб. 8.17

Job classes



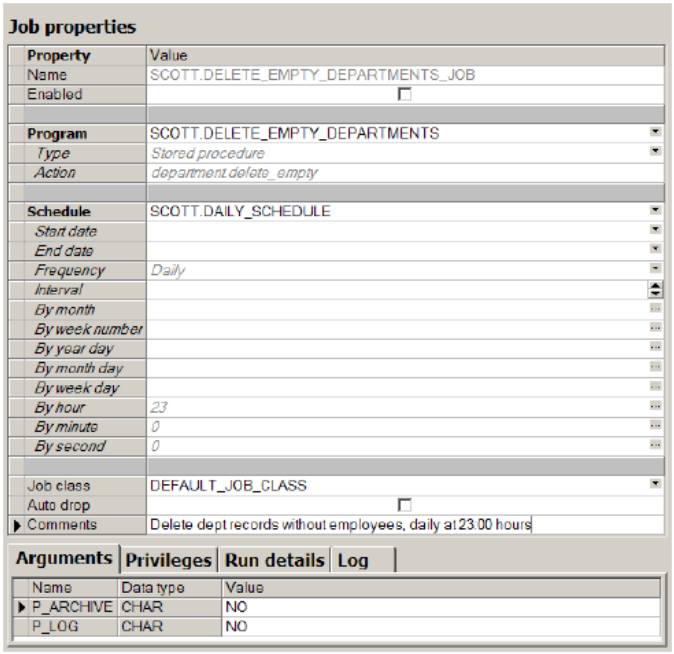
Бзб. 8.18

Programs



Бзб. 8.19

Jobs



ნახ. 8.20

ობიექტების შექმნა

ახალი ობიექტის შესაქმნელად ვაწკაპუნებთ შესაბამის ფოლდერზე და ვირჩევთ *New*-ს popup მენიუდან. შემდეგ ეკრანის მარჯვენა მხარეს შეგვაქვს თვისებები და დამატებითი დეტალური ინფორმაცია. ღილაკზე *Apply* დაჭერით სრულდება ობიექტის შექმნა მონაცემთა ბაზაში. შექმნილი ობიექტისათვის SQL ნახვა შესაძლებელია ღილაკზე *view SQL*დაჭერით, ხოლო გაუქმებისათვის ვაჭერთ ღილაკს *Cancel*.

ობიექტების რედაქტირება

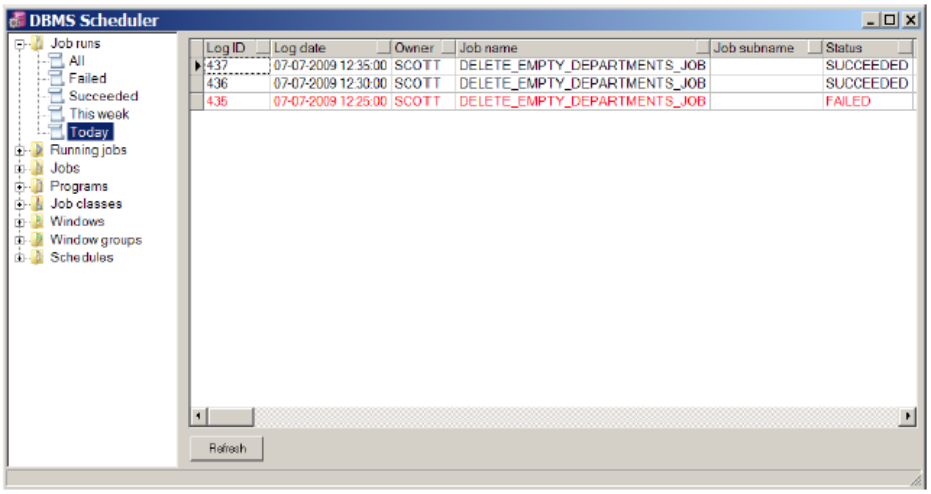
ეკრანის მარცხენა მხარეს ფოლდერში ვირჩევთ არსებულ ობიექტს, ვცვლით თვისებებს და დამატებით დეტალურ ინფორმაციას. SQL ნახვა შესაძლებელია ღილაკზე *view SQL*დაჭერით, ხოლო გაუქმებისათვის ვაჭერთ ღილაკს *Cancel*.

ობიექტების წაშლა

ეკრანის მარცხენა მხარეს ფოლდერში ვირჩევთ არსებულ ობიექტს და *Drop* -ს popup მენიუდან.

დავალების შესრულების ნახვა

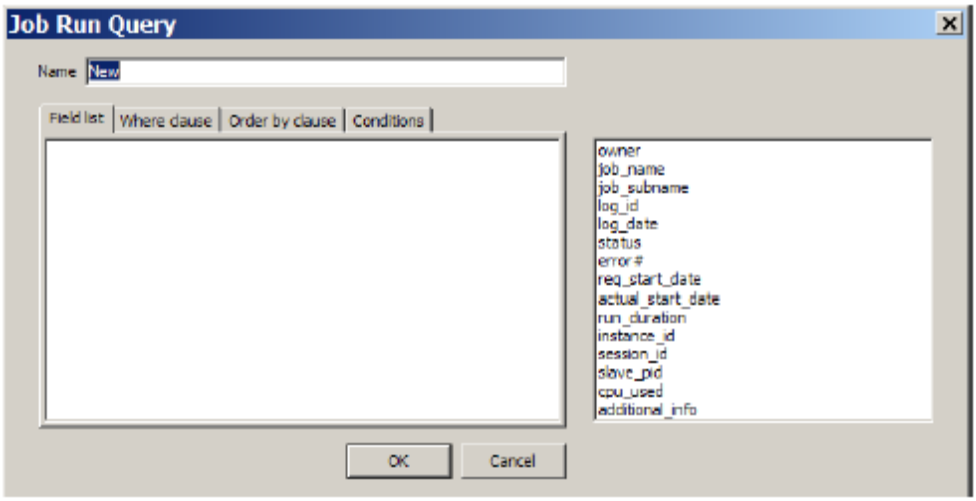
ვხსნით *Job runs* ფოლდერს. აქ დავინახავთ ანგარიშებს, როგორიც არის *All, Failed* და ა.შ. ნახვისათვის ვაწკაპუნებთ ანგარიშზე:



ნახ. 8.21

დავალების შესრულების ანგარიშის შექმნა

right-ვაწკაპუნებთ ფოლდერზე *Job run* და ვირჩევთ *New*-ს popup მენიუდან:



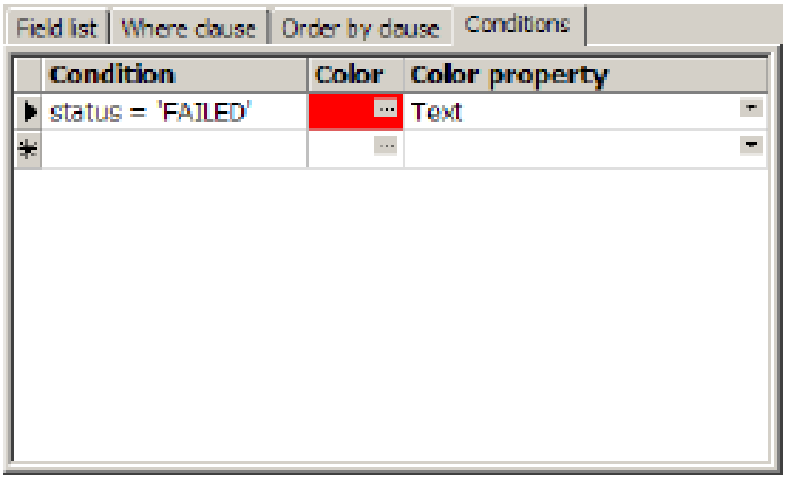
ნახ. 8.22

ჩანართში *Field list* ვირჩევთ ველებს, რომლებიც ანგარიშში უნდა იქნას ასახული.

Where clause განსაზღვრავს თუ რომელი შესრულებაზე გაშვებული დავალება უნდა აისახოს ანგარიშში.

Order by clause განსაზღვრავს ველების სიას, რომელთა დავალებები სრულდება თანამიმდევრულად.

ჩანართში *Conditions* განვსაზღვრავთ ფერებს ეკრანზე მხოლოდ გარკვეული პირობებით გაშვებული დავალებებისათვის, სადაც პირობას შეიძლება წარმოადგენდეს ნებისმიერი SQL გამოსახულება.



ნახ. 8.23

Color property -სათვის შეგვიძლია შევირჩიოთ ტექსტი ან ფონი.

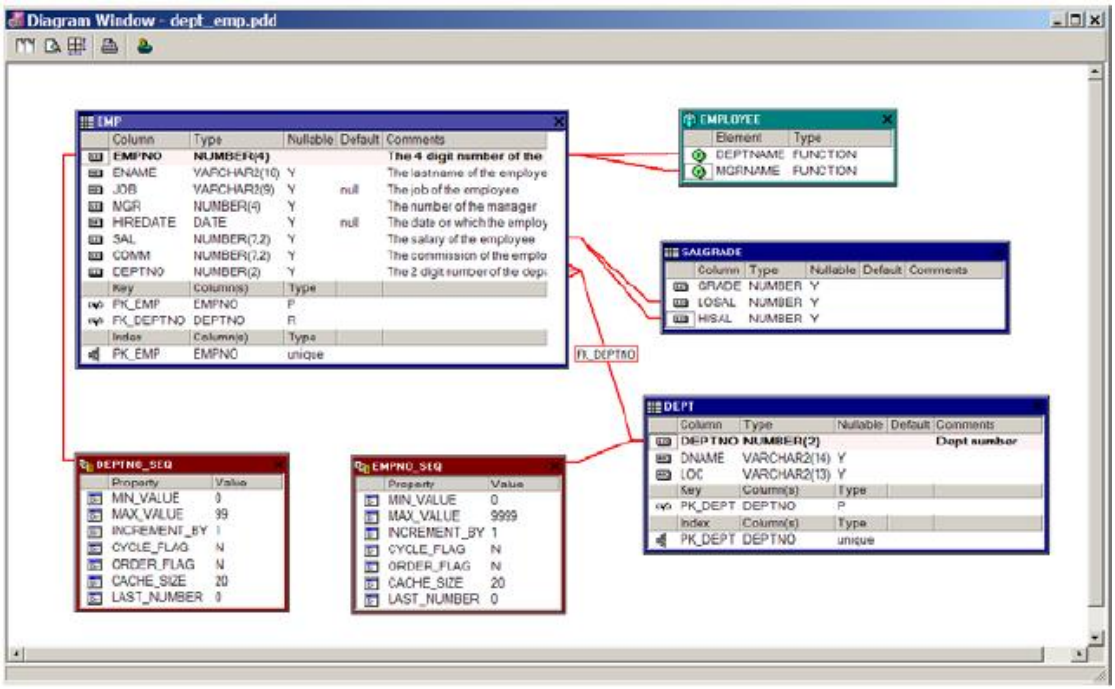
ანგარიშის რედაქტირებისათვის ან წაშლისათვის right-ვაწკაპუნებთ ანგარიშზე და ვირჩევთ შესაბამისად popup მენიუდან.

მიმდინარე დავალების ნახვა

ვაწკაპუნებთ ფოლდერზე *Running jobs*. ეკრანის მარჯვენა მხარეს გამოჩნდება ინფორმაცია მიმდინარე დავალების შესახებ.

დიაგრამები

მონაცემთა ბაზების ობიექტების ურთიერთდამოკიდებულების ვიზუალიზაციის საშუალებას იძლევა:



ნახ. 8.24

დიაგრამის შექმნა

დიაგრამის შექმნისათვის ვასრულებთ ბრძანებას: *File/ New/ Diagram Window*. ჩნდება ცარიელი ფანჯარა, სადაც მონაცემთა ბაზების ობიექტების დამატებისათვის ისინი Object Browser -დან Diagram Window-ში შეგვიძლია drag & drop-ით გადავიტანოთ.

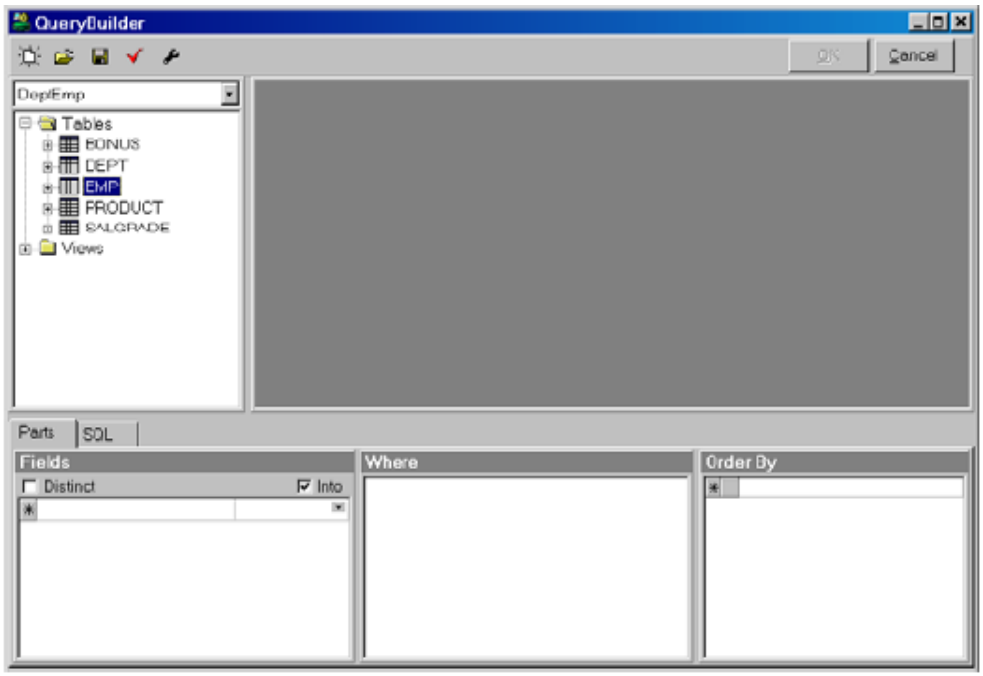
მოთხოვნის ამგები *(Query Builder)*

მოთხოვნის ამგები საშუალებას იძლევა შევქმნათ და შევცვალოთ ამორჩევის (select) ინსტრუქციები ჩვენს PL/SQL და SQL ფაილებში. მაგალითად, თუ ჩვენ გვინდა შევქმნათ join ინსტრუქცია dept და emp ცხრილებს შორის, გვექნება:

```
select e.empno, e.ename, d.deptno, d.dname
from emp e, dept d
order by e.empno
```

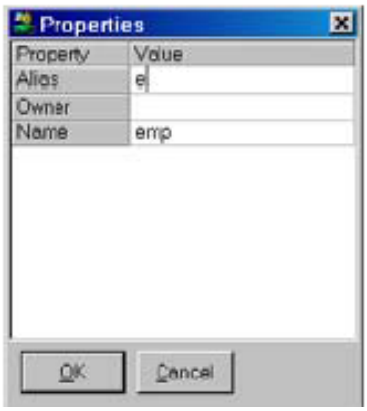
ახალი select ინსტრუქციის შესაქმნელად კურსორს განვათავსებთ ტექსტის იმ პოზიციაში, სადაც გვინდა მისი ჩასმა და ინსტრუმენტების

პანელზე (toolbar) ვაჭერთ ღილაკზე *Query Builder* ან (მენიუდან *Tools* ვირჩევთ *Query Builder*). მივიღებთ ცარიელ Query Builder Form:



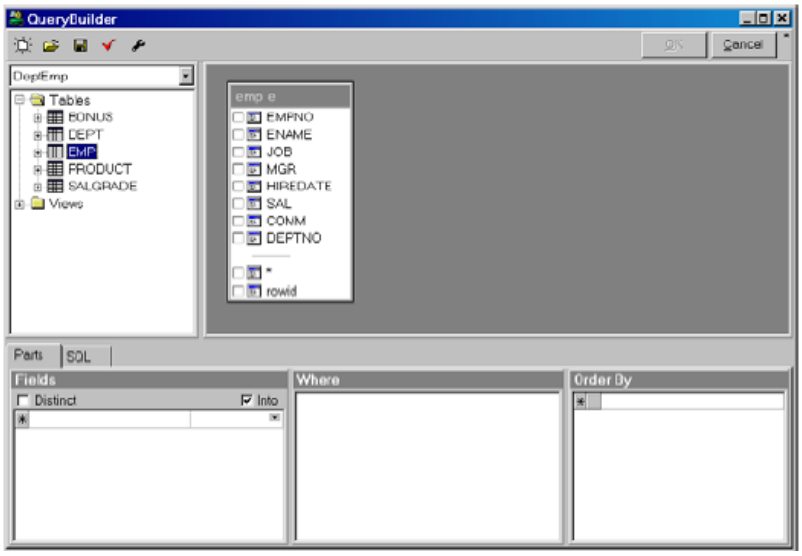
ნახ. 8.25

მარცხენა მხარეს დავინახავთ Object Browser ცხრილებით და წარმოდგენებით. ქვედა ნაწილში ვხედავთ სამ პანელს: ველების სიას, where და order by. ცხრილების ან წარმოდგენების select ინსტრუქციაში ჩასმისათვის, Object Browser-დან გადაგვაქვს ისინი სამუშაო არეში. ყოველი ახალი ცხრილის დამატებისას ჩნდება *ცხრილი Properties* ფანჯარა, რომელსაც ვავსებთ შესაბამისად:



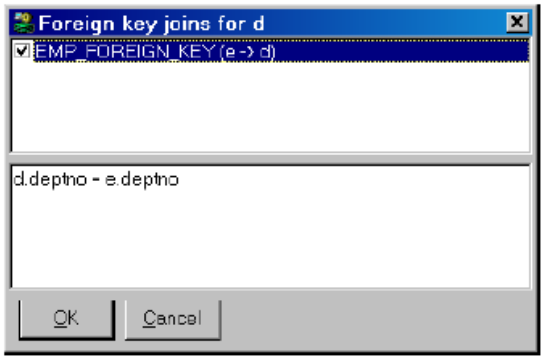
ნახ. 8.26

ჩვენი შემთხვევისათვის *emp* ცხრილისათვის ფსევდონიმში შეგვაქვს ‘e’:



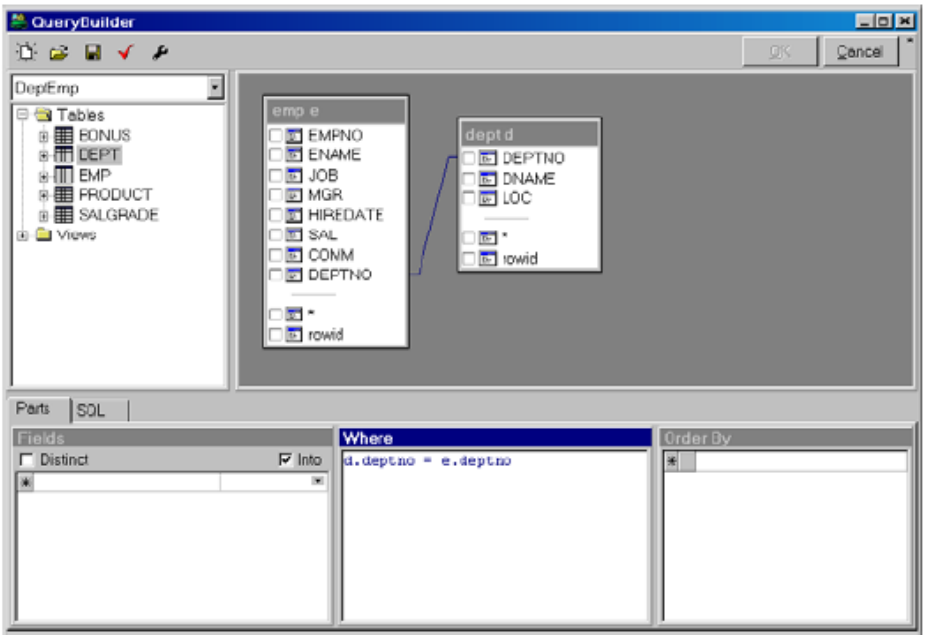
ნახ. 8.27

Dept ცხრილის დამატების შემდეგ, თუ გვინდა ამ ორი ცხრილის შეერთება, ვიყენებთ სვეტს სახელწოდებით *EMP_FOREIGN_KEY*:



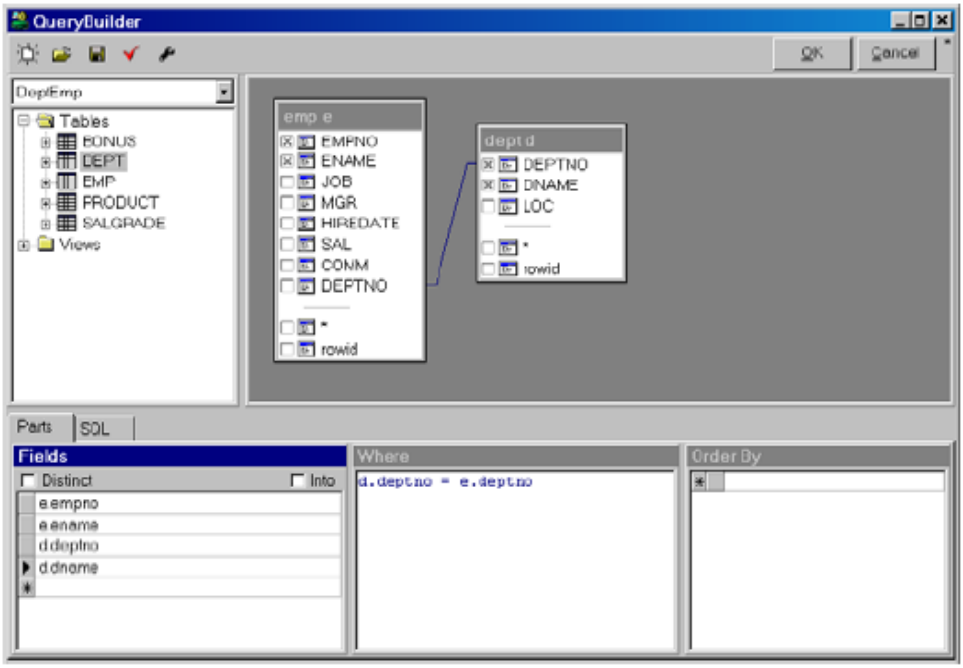
ნახ. 8.28

ფორმის ქვედა ნაწილში დავინახავთ join პირობას. შემდეგ ემატება ცხრილი *Dept* და join პირობა აისახება სამუშაო არეში:



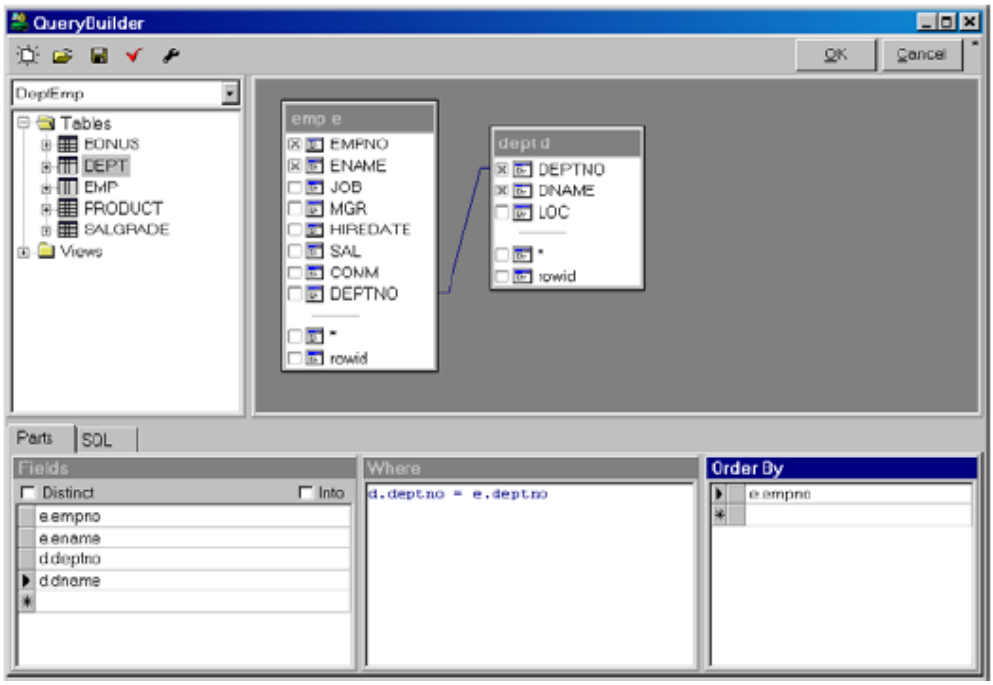
ნახ. 8.29

სვეტების სიის დამატებისათვის select ინსტრუქციაში, ვაწკაპუნებთ სვეტების სახელების checkbox-ზე. შედეგად ველების სახელები აისახება *Fields* პანელზე:



ნახ. 8.30

თუ გვინდა შევქმნათ PL/SQL-ში *select .. into* ინსტრუქცია, საჭიროა *Into* იტემის გამოყენება ველების ასახვისათვის. მოწესრიგების order by წესის შექმნისათვის გადავიტანოთ სვეტი *empno* პანელზე *Order By*:



ნახ. 8.31

რედაქტორში SQL ინსტრუქციების შეტანისათვის ვაჭერთ ღილაკს *OK*.

არსებული მოთხოვნის მოდიფიცირებისათვის ჩვენს PL/SQL ან SQL კოდში right-ვაწკაპუნებთ ამორჩევის (select) ინსტრუქციაზე და ვირჩევთ *Query Builder* პუნქტს popup მენიუდან. შემდეგ Query Builder-ში ვახდენთ select ინსტრუქციის მოდიფიცირების დასრულებას.

ცხრილიდან მოთხოვნის სამუშაო არეში ველების დამატებისათვის ვაწკაპუნებთ checkbox იტემზე ან სვეტი გადაგვაქვს ველების სიაში. ჩვენ შეგვიძლია აგრეთვე სვეტის ან სვეტების ფოლდერის გადატანა უშუალოდ Object Browser -დან ველების სიაში.

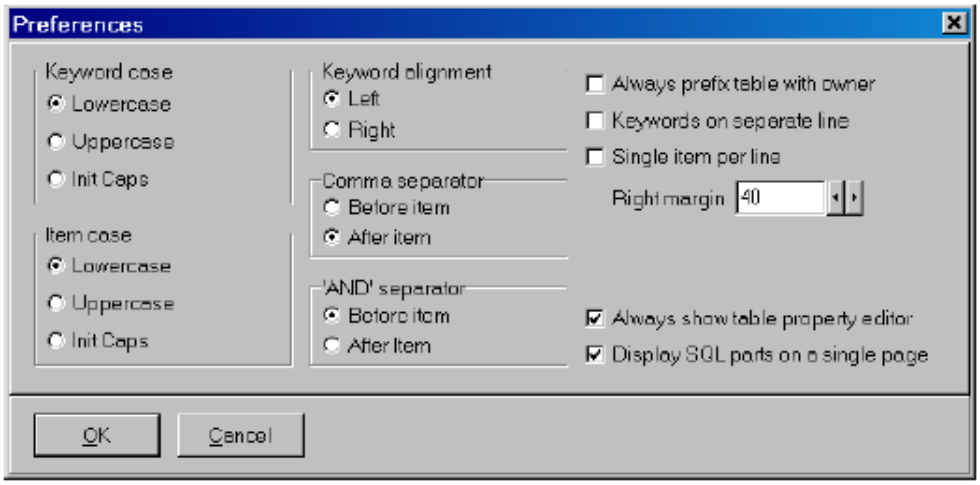
სვეტის წაშლისათვის ველების სიიდან right-ვაწკაპუნებთ ამ უკანასკნელზე და popup მენიუდან ვირჩევთ *Delete selected items*. შესაძლებელია აგრეთვე ველების პოზიციის შეცვლა მათი ამორჩევითა და ახალ პოზიციაზე გადატანით.

რედაქტორში შესაძლებელია ველებისათვის ფსევდონიმის (alias), where და order by წესების შეცვლაც, აგრეთვე join პირობის განსაზღვრა და

foreign key შეზღუდვის დამატება სამუშაო არეში ცხრილის კონტექსტური მენიუდან *Foreign keys* იტემის არჩევით.

Query Builder პრეფერენციები (Preferences)

ინსტრუმენტების პანელის ღილაკზე *Preferences* დაჭერით გამოდის შესაბამისი ფანჯარა, სადაც შეგვიძლია Query Builder-ის ქცევის შეცვლა.



ნახ. 8.32

აწყობის პროცესში ვაყენებთ შესაბამის გადამრთველებსა და დროშებს შემდეგი პრეფერენციებისათვის:

- Keyword case
- Item case
- Keyword alignment
- Comma separator
- ‘და’ separator
- Always prefix ცხრილი with owner
- Keywords on separate line
- Single item per line
- Right margin
- Always show ცხრილი property editor
- Display SQL parts on a single page

Query Builder Plug-Ins

თუ Query Builder Plug-In დაინსტალირებულია, მაშინ *Query Builder* ღილაკზე Right-დაწკაპუნებით ჩვენ შეგვიძლია ავირჩიოთ *Query Builder* ყველა დაინსტალირებული query builder-ების სიიდან.

ლაბორატორიული სამუშაო 9
PL/SQL პროგრამირება

სამუშაოს მიზანი:

1. კურსორებთან მუშაობის შესწავლა;
2. პროცედურებსა და ფუნქციებთან მუშაობის შესწავლა.

კურსორები

კურსორი – არის ცვლადი, რომელიც შედგება რომელიმე ცხრილის ან მოთხოვნის შედეგების სტრიქონებისაგან. კურსორის ჩამოყალიბებით შესაძლებელია პროგრამაში კურსორის თითოეული სტრიქონის წაკითხვა და დამუშავება. თუ კურსორი აგებულია ცხრილის საფუძველზე, მაშინ შეიძლება კურსორის მიმდინარე პოზიციის მოდიფიცირება და წაშლა.

მაგალითად:

```
1) DECLARE
   /* ცვლადები შედეგების შესანახად: */
2)    a T1.e%TYPE;
3)    b T1.f%TYPE;
   /* კურსორის გამოცხადება: */
4)    CURSOR T1Cursor IS
5)       SELECT e, f
6)       FROM T1
7)       WHERE e < f
8)       FOR UPDATE;
9) BEGIN
10)   OPEN T1Cursor;
11)   LOOP
/* გადავარჩევთ ზემოთ ფორმულირებული მოთხოვნის სტრიქონებს, */
      /* და შეგვაქვს სტრიქონები PL/SQL ცვლადებში:     */
12)      FETCH T1Cursor INTO a, b;
/* თუ სტრიქონები  აღარ არის,  მაშინ ციკლიდან გამოვდივართ: */
13)      EXIT WHEN T1Cursor%NOTFOUND;
   /* ვშლით მიმდინარე სტრიქონს: */
14)      DELETE FROM T1 WHERE CURRENT OF T1Cursor;
```

```
  /* ვსვამთ შებრუნებულ სტრიქონს : */
15)       INSERT INTO T1 VALUES(b, a);
16)    END LOOP;
 /* ვათავისუფლებთ კურსორს: */
17)    CLOSE T1Cursor;
18) END;
19) .
20) run;
```

PL/SQL ობიექტების კავშირი მონაცემთა ბაზის ცხრილებთან

იმისათვის, რომ PL/SQL პროგრამამ შეძლოს მონაცემთა ბაზებში არსებულ ინფორმაციასთან მუშაობა, აუცილებელია მნიშვნელობათა გაცვლის ორგანიზება მონაცემთა ბაზების სვეტებსა და PL/SQL ცვლადებს შორის.

ცნობილია, რომ მონაცემთა ამორჩევისათვის გამოიყენება ბრძანება SELECT, რომლის შესრულებისათვის Oracle ქმნის სპეციალურ სამუშაო არეს. PL/SQL-ს გააჩნია ამ სამუშაო არესთან წვდომის რამდენიმე მექანიზმი. ერთ-ერთი ასეთი მექანიზმია კურსორი, რომელიც OPEN, FETCH და CLOSE ბრძანებების მეშვეობით იღებს მოთხოვნათა შედეგების ცალკეულ სტრიქონთან წვდომის საშუალებას.

ცხადი კურსორები

კურსორთან მუშაობის დაწყებამდე საჭიროა მისი გამოცხადება DECLARE ბრძანებით შემდეგი სინტაქსისის გამოყენებით:

CURSOR cursor_name [(parameter [, parameter, ...])] IS SELECT ... ,

სადაც:

cursor_name - კურსორის სახელი;

SELECT ... - წინადადება SELECT, რომელიც კურსორის სტრიქონს განსაზღვრავს;

parametr - გააჩნია შემდეგი სინტაქსისი:

variable_name [IN] type_name [{ := | DEFAULT } value] ,

type_name - PL/SQL მონაცემთა ტიპი (ქვეტიპი) შეზღუდვების (მაგალითად, სიმბოლური მნიშვნელობის სიგრძის) მითითების გარეშე.

მაგალითი:

```
DECLARE
CURSOR s1 (otd INTEGER := 102, grup VARCHAR2 DEFAULT 'Staff',
        tdat DATE := '1.1.1996') IS
  SELECT (TO_CHAR(razr)||' '||imya_dolg) razr_dolg, stavka
    FROM shtat x, dolgnosti y, grup_dolg z
   WHERE x.dolgn = y.dolgn  და y.grup_dolg = z.grup_dolg  და otdel =
otd
     და tdat BETWEEN nachalo და konec  და imya_grup_dolg = grup
   ORDER BY razr_dolg DESC;
```

ბრძანებას OPEN გააჩნია შემდეგი სინტაქსისი

```
OPEN cursor_name [ (value [,value]...) ];
```

სადაც:

მნიშვნელობათა სია("value") გამოიყენება კურსორის პარამეტრების გადასაცემად. ამასთან, მონაცემთა ტიპისა და რაოდენობის მიხედვით უნდა ემთხვეოდეს ამ პარამეტრების აღწერას.

ბრძანებას FETCH, რომელიც გამოიყენება კურსორის მიმდინარე სტრიქონის მაჩვენებლის ერთი ბიჯით გადაადგილებისათვის, გააჩნია შემდეგი სინტაქსისი:

```
FETCH cursor_name INTO {variable_name1[,variable_name2]...} | record_name ;
```

მაგალითი:

```
CREATE PROCEDURE pr_shtat IS
 CURSOR s1 (otd INTEGER := 102, grup VARCHAR2 DEFAULT 'Staff',
        tdat DATE := '1.1.1996') IS
  SELECT (TO_CHAR(razr)||' '||imya_dolg) razr_dolg, stavka
    FROM shtat x, dolgnosti y, grup_dolg z
   WHERE x.dolgn = y.dolgn  და y.grup_dolg = z.grup_dolg  და otdel =
otd
     და tdat BETWEEN nachalo და konec  და imya_grup_dolg = grup
   ORDER BY razr DESC;
  sh_raz  VARCHAR2(45);      -- ცვლადი razr_dolg -ის მნიშვნელობის
შესანახად.
  sh_stav shtat.stavka%TYPE;   -- ცვლადი stavka -ის მნიშვნელობის
შესანახად.
```

```
raz VARCHAR(500);      -- ცვლადი, რომელშიც თანმიმდევრულად
                          დაგროვდება ტექსტი „razr_ dolg“.
OPEN s1;
 LOOP
  FETCH s1 INTO sh_raz,sh_stav;
  EXIT WHEN s1%NOTFOUND; -- გამოსვლა დაბრუნების
                          სტრიქონის არქონის შემთხვევაში.
  raz := raz||sh_raz||';';
    ...
 END LOOP;
CLOSE s1;
END pr_shtat;
```

ამ მაგალითში ციკლის შიგნით შეიძლება sh_raz და sh_stav ცვლადების მნიშვნელობების გამოყენება. ჩანაწერში მიმდინარე სტრიქონის მნიშვნელობის ამორჩევის დროს, მაგალითად ShRec სახელწოდებით, საჭიროა ცოტათი შეიცვალოს როგორც აღწერა, ისე პროცედურის ბლოკის ტანი:

```
...
 ORDER BY razr DESC;
TYPE ShRecTyp IS RECORD (raz_dol VARCHAR(45),
   -- ShRec ჩანაწერის მონაცემთა აღწერა stav shtat.stavka%TYPE);
ShRec ShRecTyp;          -- ShRec ჩანაწერის გამოცხადება;
raz VARCHAR(500);      -- ცვლადი, რომელშიც თანმიმდევრულად
                          დაგროვდება ტექსტი „razr_ dolg“.
OPEN s1;
 LOOP
  FETCH s1 INTO ShRec;
  EXIT WHEN s1%NOTFOUND;   -- გამოსვლა დაბრუნების
                          სტრიქონის არქონის შემთხვევაში.
  raz := raz||ShRec.raz_dol||';';
    ...
```

ამჯერად, მნიშვნელობები, რომლებსაც ადრე ვიღებდით sh_raz და sh_stav-დან, შეიძლება მივიღოთ ShRec ჩანაწერის ველებიდან ShRec.raz_dol და ShRec.stav.

ბრძანება CLOSE გამოიყენება ყველა იმ რესურსის გამოსათავისუფლებლად, რომლებიც კურსორის გახსნით იყო დაკავებული. მისი სინტაქსისია: CLOSE cursor_name;

FOR ციკლში კურსორის გამოყენების სინტაქსისს გააჩნია შემდეგი სახე:

```
FOR var_rec_name IN cursor_name [ (value [,value]...) ] LOOP
        ციკლის ტანი
END LOOP;
```

მაგალითი:

```
DROP PROCEDURE pr_shtat;
CREATE PROCEDURE pr_shtat IS
CURSOR s1 (otd INTEGER := 102, grup VARCHAR2 DEFAULT 'Staff',
      tdat DATE := '1.1.1996') IS
 SELECT otdel, (TO_CHAR(razr)||' '||imya_dolg) razr_dolg, stavka
   FROM shtat x, dolgnosti y, grup_dolg z
  WHERE x.dolgn = y.dolgn  და y.grup_dolg = z.grup_dolg  და otdel =
otd
    და tdat BETWEEN nachalo და konec  და imya_grup_dolg = grup
  ORDER BY razr DESC;
 raz  VARCHAR(500);  -- ცვლადი, რომელშიც თანმიმდევრულად
                        დაგროვდება ტექსტი „razr_ dolg“.
BEGIN
 FOR s1_rec IN s1 (102,'Specialists','1.6.1996') LOOP
  raz := raz||s1_rec.razr_dolg||';';
    ...
 END LOOP;
END pr_shtat;
```

კურსორის მიმდინარე სტრიქონის ცვლილების ან წაშლისათვის გამოიყენება შესაბამისი ბრძანებები:

```
UPDATE [schema.]{ცხრილი | წარმოდგენა}[@dblink] [alias]
    SET { (column [, column] ...) = (subquery)
       | column = { expr | (subquery) } }
     [, { (column [, column] ...) = (subquery)
       | column = { expr | (subquery) } } ] ...
    WHERE CURRENT OF cursor_name;
  DELETE [FROM] [schema.]{ცხრილი |
 წარმოდგენა}[@dblink] [alias]
```

WHERE CURRENT OF cursor_name;

ამისათვის აუცილებელია კურსორის გამოცხადების დროს ბრძანება SELECT ... შეიცავდეს

FOR UPDATE OF [[schema.]{ცხრილი | წარმოდგენა}.]column
[, [[schema.]{ცხრილი | წარმოდგენა}.]column] ... ;

არაცხადი კურსორები (SQL კურსორი)

არაცხადი კურსორის შემთხვევაში, რომლისადმი მიმართვა ხდება SQL% სახელით, არ შეიძლება OPEN, FETCH და CLOSE ბრძანებების გამოყენება. იმ შემთხვევებში, როცა პროგრამაში უნდა გვქონდეს სვეტების მნიშვნელობები ცხრილის ერთი სტრიქონიდან, შეიძლება გამოვიყენოთ ბრძანება SELECT ... INTO, შემდეგი ფორმატით:

SELECT [DISTINCT | !!under!!ALL]
{ [schema.]{ცხრილი | წარმოდგენა | snapshot}.expr [c_alias] }
[, { [schema.]{ცხრილი | წარმოდგენა | snapshot}.expr [c_alias] }] ... }
INTO { variable_name [, variable_name] ... } | record_name
FROM ცხრილი_list [WHERE condition]
[GROUP BY expr [, expr] ...] [HAVING condition]
[{UNION | UNION ALL | INTERSECT | MINUS} SELECT commდა]
[ORDER BY {expr | c_alias | position}
[!!under!!ASC | DESC] [, {expr | c_alias | position}
[!!under!!ASC | DESC]]]...
[FOR UPDATE [OF [[schema.]{ცხრილი | წარმოდგენა}.]column
[, [[schema.]{ცხრილი | წარმოდგენა}.]column] ...]
[NOWAIT]]

SELECT...INTO, INSERT, UPDATE და DELETE ბრძანებების შესრულების შედეგების ანალიზისათვის გამოიყენება სამი ცვლადი: SQL%NOTFOUND, SQL%FOUND და SQL%ROWCOUNT. SELECT...INTO, INSERT, UPDATE და DELETE ბრძანებების შესრულების წინ ცვლადებს SQL%NOTFOUND და SQL%FOUND გააჩნიათ მნიშვნელობა NULL. ცვლადი SQL%NOTFOUND იღებს TRUE მნიშვნელობას, თუ INSERT,

UPDATE და DELETE ბრძანებებმა არ მოახდინეს ცხრილებში ცვლილება ან SELECT...INTO-ს არ აბრუნებს სტრიქონს (ამასთან ცვლადი SQL%FOUND იღებს მნიშვნელობას FALSE).

წინააღმდეგ შემთხვევაში ცვლადი SQL%NOTFOUND იღებს FALSE მნიშვნელობას, ხოლო ცვლადი SQL%FOUND მნიშვნელობას - TRUE.

მაგალითი:

```
UPDATE shtat SET stavka = stavka + 1 WHERE dolgn = 'პროფესორი' და razr = 15;
IF SQL%NOTFOUND THEN       -- ცვლილება არ შესრულდა
INSERT INTO temp VALUES (...);
END IF;
```

პროცედურები და ფუნქციები

პროცედურის ან ფუნქციის შექმნა SQL Commands Page-ის მეშვეობით

ჩვენ შეგვიძლია გამოვიყენოთ SQL Commands გვერდი შენახული პროცედურების ან ფუნქციების შესაქმნელად. ამისათვის:

1. დავრეგისტრირდეთ Database Home გვერდზე.

2. home გვერდზე ვაწკაპუნებთ **SQL** იკონაზე. ჩნდება SQL გვერდი.

3. ვაწკაპუნებთ **SQL Commands** იკონაზე. ჩნდება SQL Commands გვერდი.

4. SQL Commands გვერდზე შეგვაქვს PL/SQL კოდი PL/SQL პროცედურის ან ფუნქციისათვის.

5. ვირჩევთ (გამოყოფით) კოდს პროცედურის ან ფუნქციის შექმნისათვის. შემდეგ ვაწკაპუნებთ **Run** ღილაკზე.

6. ვირჩევთ (გამოყოფით) კოდს პროცედურის ან ფუნქციის გამოძახებისათვის. შემდეგ ვაწკაპუნებთ **Run** ღილაკზე.

7. PL/SQL კოდის შენახვისათვის ვაწკაპუნებთ **Save** ღილაკზე.

8. ველში **Name** შეგვაქვს PL/SQL კოდის სახელი. ვაწკაპუნებთ **Save** ღილაკზე.

9. PL/SQL კოდის წვდომისათვის ვაწკაპუნებთ ჩანართზე **Saved SQL** და ვირჩევთ PL/SQL კოდის სახელს.

პროცედურის ან ფუნქციის შექმნა Object Browser გვერდის მეშვეობით

შენახული პროცედურების ან ფუნქციების შექმნის მიზნით Object Browser გვერდის გამოყენების შემთხვევაში:

1. დავრეგისტრირდეთ Database Home გვერდზე. მაგალითის სახით გამოვიყენოთ მომხმარებელი HR.

2. Database Home გვერდზე ვაწკაპუნებთ **Object Browser** იკონაზე.

3. ობიექტების სიაში **Create** ვირჩევთ **Procedure**.

4. შეგვაქვს პროცედურის სახელი (award_bonus), ვაყენებთ **Include Arguments** და შემდეგ ვაწკაპუნებთ **Next** ღილაკზე.

5. შეგვაქვს ინფორმაცია არგუმენტებისათვის.

შემდეგ ვაწკაპუნებთ **Next**. მაგალითად:

Name IN/OUT Type
emp_id IN NUMBER
bonus_rate IN NUMBER

6. შეგვაქვს საწყისი კოდი პროცედურის ტანისათვის. შემდეგ ვაწკაპუნებთ **Next** ღილაკზე. შეგვაქვს PL/SQL საწყისი კოდი.

7. ვაწკაპუნებთ ჩანართზე პროცედურის ტანის **SQL** საწყისი კოდის ნახვისათვის.

თუ საჭიროა კორექტირება, ვაწკაპუნებთ **Previous** ღილაკზე.

8. დასასრულს, ვაწკაპუნებთ **Finish** ღილაკზე. ქვეპროგრამების განახლებისათვის (მაგალითად, დამატებითი ცვლადების BEGIN .. END ბლოკის გარე ნაწილში დამატებისათვის) ვაწკაპუნებთ **Edit** ღილაკზე.

9. პროცედურის კომპილაციისათვის ვაწკაპუნებთ **Compile** ღილაკზე. თუ შეცდომები აღმოჩნდება, საწყის კოდში შეგვაქვს ცვლილებები და განმეორებითი კომპილაციის შემდეგ ვინახავთ ცვლილებებს.

10. დასასრულს, ვაწკაპუნებთ **Finish** ღილაკზე.

პროცედურის ან ფუნქციის ნახვა Object Browser გვერდის მეშვეობით

შენახული პროცედურების ან ფუნქციების ნახვის მიზნით Object Browser გვერდის გამოყენების შემთხვევაში:

1. დავრეგისტრირდეთ Database Home გვერდზე.

მაგალითის სახით გამოვიყენოთ მომხმარებელი HR.

2. Database Home გვერდზე ვაწკაპუნებთ **Object Browser** იკონაზე.

3. ობიექტების სიაში ვირჩევთ **Procedures** ან **Functions**. შემდეგ ვაწკაპუნებთ პროცედურის ან ფუნქციის სახელზე, რომლის ეკრანზე გამოტანაც გვინდა.
მაგალითისათვის, შეგვიძლია შევირჩიოთ **Procedures**, შემდეგ ვაწკაპუნებთ პროცედურის სახელზე (AWARD_BONUS), რომელიც წინასწარ არის შექმნილი.

შენახული პროცედურების შექმნა SQL CREATE PROCEDURE მეშვეობით

ბრძანება SQL CREATE PROCEDURE შენახული პროცედურების შემნის საშუალებას იძლევა. ჩვენ შეგვიძლია აგრეთვე გამოვიყენოთ ოპციური პირობა OR REPLACE, რომელიც ცვლის არსებულ პროცედურას პირვანდელი პროცედურის წაშლის გარეშე.

მარტივი შენახული პროცედურის შექმნა

მაგალითი:

```
CREATE OR REPLACE PROCEDURE today_is AS
BEGIN
-- display  current system date in long format
DBMS_OUTPUT.PUT_LINE( 'Today is ' || TO_CHAR(SYSDATE, 'DL') );
END today_is;
/
-- to call  procedure today_is, you can use  following block
BEGIN
today_is(); --  parenses are optional here
END;
/
```

პროცედურების ან ფუნქციების რედაქტირება

პროცედურების ან ფუნქციების რედაქტირებისათვის შეგვიძლია გამოვიყენოთ Object Browser გვერდი, SQL Command გვერდი ან SQL Command Line-ის მეშვეობით ბრძანება SQL CREATE OR REPLACE.

ა) SQL Command *გვერდში პროცედურის რედაქტირება:*

1. დავრეგისტრირდეთ Database Home გვერდზე.
2. home გვერდზე ვაწკაპუნებთ **SQL** იკონაზე. ჩნდება SQL გვერდი.
3. ვაწკაპუნებთ **SQL Command** იკონაზე. ჩნდება SQL Command გვერდი.

4. ვაწკაპუნებთ ჩანართზე **Saved SQL** შენახული SQL მოდულების ეკრანზე გამოსაჩენად.
5. ვაწკაპუნებთ SQL სახელზე, რომელიც შეიცავს ჩვენთვის საინტერესო პროცედურის ან ფუნქციის კოდს.
6. ვცვლით საწყის კოდს პროცედურის ან ფუნქციისათვის. რედაქტირების შესრულებისათვის ვაწკაპუნებთ **Run** ღილაკზე.
7. დასასრულს, ვაწკაპუნებთ **Save** ღილაკზე.

გ) ქვეპროგრამის რედაქტირება Object Browser -ის მეშვეობით:

1. დავრეგისტრირდეთ Database Home გვერდზე.
მაგალითის სახით გამოვიყენოთ HR მომხმარებელი .
2. ვაწკაპუნებთ **Object Browser** იკონაზე. ჩნდება Object Browser გვერდი.
3. ობიექტების სიაში ვირჩევთ **Procedures** ან **Functions.** შემდეგ ვაწკაპუნებთ ქვეპროგრამაზე.
4. ჩნდება ქვეპროგრამა. მისი კოდის რედაქტირებისათვის ვაწკაპუნებთ **Edit** ღილაკზე.
5. ვაწკაპუნებთ **Compile** ღილაკზე. თუ შეცდომები აღმოჩნდება, კოდში შეგვაქვს ცვლილებები და განმეორებითი კომპილაციის შემდეგ ვინახავთ ცვლილებებს.

პროცედურის ან ფუნქციის წაშლა

პროცედურის ან ფუნქციის წაშლისათვის შეგვიძლია გამოვიყენოთ Object Browser გვერდი ან SQL DROP ბრძანება. Object Browser გვერდის გამოყენების შემთხვევაში:
1. დავრეგისტრირდეთ Database Home გვერდზე.
მაგალითის სახით გამოვიყენოთ HR მომხმარებელი .
2. Database Home გვერდზე ვაწკაპუნებთ **Object Browser** იკონაზე.
3. ობიექტების სიაში ვირჩევთ **Procedures** ან **Functions**, შემდეგ კი ვაწკაპუნებთ პროცედურის ან ფუნქციის სახელზე, რომლის წაშლაც გვინდა.
4. ვაწკაპუნებთ **Drop** ღილაკზე.
5. დასასრულს წაშლის ოპერაციის დადასტურებისათვის ვაწკაპუნებთ **Finish** ღილაკზე.

SQL ბრძანებების მეშვეობით პროცედურის ან ფუნქციის წაშლისათვის გამოიყენება SQL DROP PROCEDURE ან DROP FUNCTION ბრძანება.

ქვეპროგრამების წაშლა DROP ბრძანების მეშვეობით

მაგალითი:

```
-- drop  procedure award_bonus to remove from  database
DROP PROCEDURE award_bonus;
-- drop  function emp_sal_ranking to remove from database
DROP FUNCTION emp_sal_ranking;
```

ლაბორატორიული სამუშაო 10

ტრიგერები. პაკეტები. ტრანზაკციების მართვა

სამუშაოს მიზანი:

1. ტრიგერებთან მუშაობის შესწავლა;
2. პაკეტებთან მუშაობის შესწავლა;
3. ტრანზაკციების მართვის შესწავლა.

ტრიგერები

ტრიგერი – არის PL/SQL ენის სპეციალური პროცედურა. ჩვეულებრივი პროცედურა სრულდება მისი სპეციალური ბრძანებით გამოძახების შემდეგ. ტრიგერი კი იწყებს შესრულებას, როცა სრულდება გარკვეული ხდომილობა ცხრილთან, კერძოდ ბრძანებების INSERT, DELETE ან UPDATE შესრულებამდე ან შემდეგ.

ტრიგერის განსაზღვრის სინტაქსისის გამარტივებული ფორმატი შემდეგია:

```
CREATE [OR REPLACE] TRIGGER <ტრიგერის_სახელი>
{BEFORE|AFTER} {INSERT|DELETE|UPDATE} ON <ცხრილის_სახელი>
[FOR EACH ROW [WHEN (<ტრიგერის_პირობა>)]]
<ტრიგერის_ტანი>
```

ცხრილებისათვის შესაძლებელია მხოლოდ BEFORE ან AFTER ტრიგერების განსაზღვრა.

მაგალითისათვის განვსაზღვროთ 2 ცხრილი:

```
CREATE ცხრილი T4 (a INTEGER, b CHAR(10));
CREATE ცხრილი T5 (c CHAR(10), d INTEGER);
```

განვსაზღვროთ ტრიგერი, რომელიც T5 ცხრილში ამატებს სტრიქონს (კორტეჟს), თუ T4 ცხრილში ემატება სტრიქონი (კორტეჟი). ამასთან, ტრიგერი ამოწმებს, დამატებულ სტრიქონში პირველი კომპონენტი თუ არის 10-ზე ნაკლები ან ტოლი. თუ ასეთი კომპონენტი არის, T5 ცხრილში დაემატება შებრუნებული სტრიქონი (კორტეჟი):

```
CREATE TRIGGER trig1
AFTER INSERT ON T4
FOR EACH ROW
```

```
WHEN (NEW.a <= 10)
BEGIN
  INSERT INTO T5 VALUES(:NEW.b, :NEW.a);
END trig1;
.
run;
```

ბრძანება CREATE TRIGGER სრულდება წერტილით და ბრძანებით run;, ოღონდ ამით სრულდება ტრიგერის შექმნა და არა მისი შესრულებაზე გაშვება. მხოლოდ ტრიგერთან დაკავშირებული ხდომილობა, კერძოდ T4 ცხრილში სტრიქონის ჩასმა გამოიწვევს მის შესრულებას.

შექმნილი ტრიგერების შესახებ ინფორმაციის ნახვისათვის გამოიყენება ბრძანებები:

```
select trigger_name from user_triggers;
select trigger_type, ცხრილი_name, triggering_event
    from user_triggers
 where trigger_name = '<ტრიგერის_სახელი>';
```

ტრიგერის წაშლისათვის გამოიყენება ბრძანება:

```
drop trigger <ტრიგერის_სახელი>;
```

ტრიგერის წვდომის ან მიუწვდომელობის თვისებების განსაზღვრისათვის გამოიყენება ბრძანება:

```
alter trigger <ტრიგერის_სახელი> {disable|enable};
```

შეცდომების ძებნა

PL/SQL ყოველთვის არ იძლევა შეტყობინებას კომპილაციის შეცდომების შესახებ. ამის ნაცვლად გამოდის შეტყობინება "procedure created with compilation errors" ("პროცედურა შექმნილია კომპილაციის შეცდომებით"). თუ შეცდომა გაუგებარია, უნდა გამოვიყენოთ ბრძანება:

```
show errors procedure <პროცედურის_სახელი>;
```

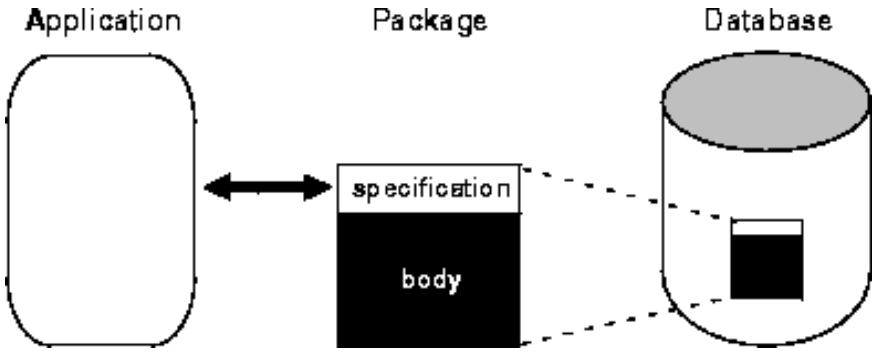
ანალოგიურია ბრძანება ტრიგერის შეცდომების ნახვის თაობაზე:

```
show errors trigger <ტრიგერის_სახელი>;
```

გარდა ამისა, "SHO ERR" (შემოკლებულად "SHOW ERRORS ") ბრძანების გამოყენებით შესაძლებელია კომპილაციის შეცდომების ნახვა.

პაკეტები

პაკეტი წარმოადგენს სქემას, რომელიც აკავშირებს დანართისა და მონაცემთა ბაზის ობიექტებს, აჯგუფებს ლოგიკურად დაკავშირებულ PL/SQL ტიპებს, ცვლადებს და ქვეპროგრამებს.



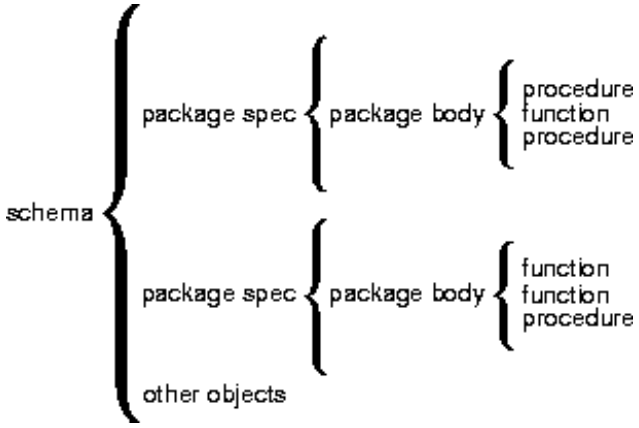
ნახ. 10.1

პაკეტი ჩვეულებრივ შედგება ორი ნაწილისაგან:

1) სპეციფიკაცია (ინტერფეისი), რომელშიც ხდება ტიპების, ცვლადების, კონსტანტების, კურსორების, ქვეპროგრამების გამოცხადება;

2) ტანი, სადაც განისაზღვრება მოთხოვნები კურსორებისათვის და კოდები ქვეპროგრამებისათვის.

პაკეტის ობიექტთა მოქმედების არეალი განისაზღვრება შემდეგი სტრუქტურით.



ნახ. 10.2

ზოგადად, მისი სინტაქსისი შემდეგია:

```
CREATE [OR REPLACE] PACKAGE package_name
 [AUTHID {CURRENT_USER | DEFINER}] {IS | AS}
 [type_definition [type_definition] ...]
 [cursor_spec [cursor_spec] ...]
 [item_declaration [item_declaration] ...]
 [{subprogram_spec | call_spec} [{subprogram_spec | call_spec}]...]
END [package_name];

[CREATE [OR REPLACE] PACKAGE BODY package_name {IS | AS}
 [type_definition [type_definition] ...]
 [cursor_body [cursor_body] ...]
 [item_declaration [item_declaration] ...]
 [{subprogram_spec | call_spec} [{subprogram_spec | call_spec}]...]
[BEGIN
 sequence_of_statements]
END [package_name];]
```

პაკეტის შექმნის, მოდიფიცირებისა და წაშლისათვის გამოიყენება Object Browser, SQL Commands, Script Editor ან SQL Command Line (SQL*Plus). არსებული პაკეტების ნახვა შესაძლებელია Object Browser გვერდის გამოყენებით.

პაკეტების შექმნა

პაკეტების შექმნა SQL Commands გვერდში

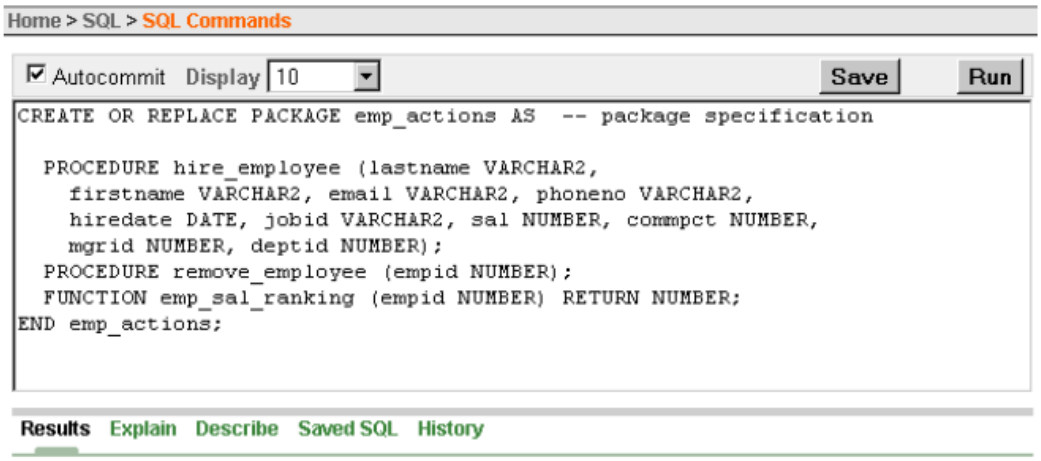
პაკეტის სპეციფიკაციის ან ტანის შექმნისა და შესრულებაზე გაშვებისათვის **SQL Commands** გვერდში:

1. დავრეგისტრირდეთ Database Home გვერდზე.

2. home გვერდში ვაწკაპუნებთ იკონაზე **SQL.** ჩნდება SQL გვერდი.

3. ვაწკაპუნებთ იკონაზე **SQL Commands.**

4. SQL Commands გვერდზე შეგვყავს PL/SQL კოდი პაკეტის სპეციფიკაციის ან ტანისათვის.



ნახ. 10.3

5. ვაწკაპუნებთ ღილაკზე **Run** პაკეტის სპეციფიკაციის ან ტანის შესაქმნელად.

6. PL/SQL კოდის შენახვისათვის ვაწკაპუნებთ ღილაკზე **Save**.

7.ველში **Name** შეგვაქვს PL/SQL კოდის სახელი (emp_actions_pkg_spec). ჩვენ შეგვიძლია აგრეთვე შევიტანოთ მისი ოპციური აღწერილობაც. ვაწკაპუნებთ ღილაკზე **Save**.

8. PL/SQL კოდის წვდომისათვის, ვაწკაპუნებთ ჩანართზე **Saved SQL** და ვირჩევთ შენახული PL/SQL კოდის სახელს.

9. პაკეტის ტანისათვის PL/SQL კოდის შექმნის, შესრულებაზე გაშვების და შენახვის მიზნით ვიმეორებთ შესაბამის ბიჯებს.

პაკეტების შექმნა Object Browser გვერდში

Object Browser გვერდში პაკეტის სპეციფიკაციის შექმნისათვის:

1. დავრეგისტრირდეთ Database Home გვერდში.

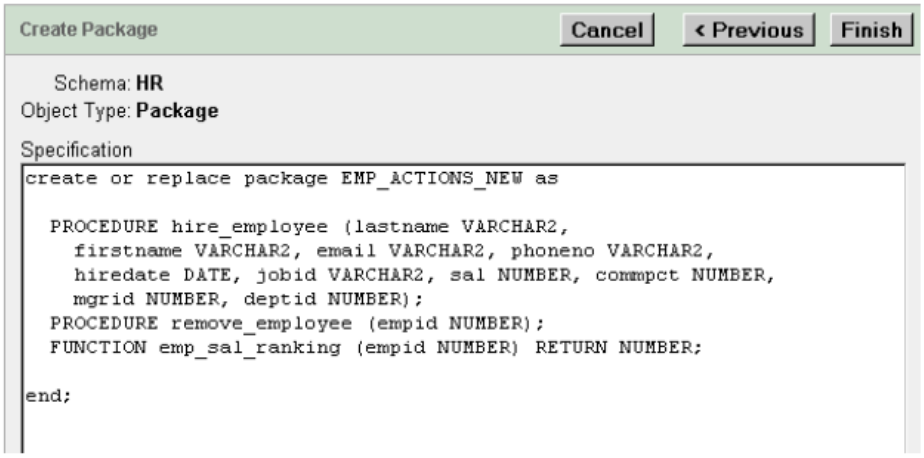
2. Database Home გვერდზე ვაწკაპუნებთ იკონაზე **Object Browser**.

3. Detail პანელზე **Create** მენიუდან ვირჩევთ **Package**.

4. Create Package გვერდში ვირჩევთ ოპციას **Specification** და ვაწკაპუნებთ **Next**.

5. შეგვაქვს პაკეტის სახელი (emp_actions_new) დავაწკაპუნებთ ღილაკზე **Next**.

6. შეგვაქვს PL/SQL საწყისი კოდი პაკეტის სპეციფიკაციისათვის.



ნახ. 10.4

7. პაკეტის სპეციფიკაციისათვის კოდის შეტანის შემდეგ ვაწკაპუნებთ ღილაკზე **Finish**.

8. ვაწკაპუნებთ ჩანართზე **Body**, შემდეგ ღილაკზე **Edit** პაკეტის ტანისათვის საწყისი კოდის შესატანად.

9. პაკეტის შესრულებაზე გაშვებისათვის ვაწკაპუნებთ ღილაკზე **Compile**. შეცდომის შემთხვევაში საწყის კოდში შეგვაქვს შესწორება და თავიდან ვაკომპილირებთ,რის შემდეგაც პაკეტს ვინახავთ.

10. დამთავრებისათვის ვაწკაპუნებთ ღილაკზე **Finish**.

პაკეტების ნახვა Object Browser გვერდის მეშვეობით

პაკეტების და პაკეტის ტანების ნახვისათვის ვიყენებთ Object Browser გვერდს:

1. დავრეგისტრირდეთ Database Home გვერდში.

2. Database Home გვერდზე ვაწკაპუნებთ იკონაზე **Object Browser**.

3. ობიექტთა სიაში ვირჩევთ **Packages** და ვაწკაპუნებთ სასურველი პაკეტის სახელზე.

4. როდესაც სპეციფიკაცია ეკრანზე გამოჩნდება, პაკეტის ტანის ნახვისათვის ვაწკაპუნებთ ჩანართზე **Body**.

პაკეტის სპეციფიკაციის შექმნა

მაგალითი:

CREATE OR REPLACE **packet emp_actions** AS – packets specification

PROCEDURE **hire_employee** (lastname VARCHAR2,
firstname VARCHAR2, email VARCHAR2, phoneno VARCHAR2,
hiredate DATE, jobid VARCHAR2, sal NUMBER, commpct NUMBER,
mgrid NUMBER, deptid NUMBER);
PROCEDURE **remove_employee** (empid NUMBER);
FUNCTION **emp_sal_ranking** (empid NUMBER) RETURN NUMBER;
END **emp_actions**;
/

პაკეტის ტანის შექმნა

მაგალითი:

```
CREATE OR REPLACE Packets BODY emp_actions AS -- Packets body
-- hire_employee პროცედურის კოდი, რომელიც ამატებს ახალ employee -ს:
PROCEDURE hire_employee (lastname VARCHAR2, firstname
VARCHAR2, email VARCHAR2, phoneno VARCHAR2, hiredate
DATE, jobid VARCHAR2, sal NUMBER, commpct NUMBER, mgrid
NUMBER, deptid NUMBER) IS
min_sal employees.salary%TYPE;
max_sal employees.salary%TYPE;
seq_value NUMBER;
BEGIN
SELECT employees_seq.NEXTVAL INTO seq_value FROM DUAL;
INSERT INTO employees VALUES (seq_value, lastname, firstname,
email, phoneno, hiredate, jobid, sal, commpct, mgrid, deptid);
SELECT min_salary INTO min_sal FROM jobs WHERE job_id =
jobid;
SELECT max_salary INTO max_sal FROM jobs WHERE job_id =
jobid;
IF sal > max_sal then
DBMS_OUTPUT.PUT_LINE('Warning: ' || TO_CHAR(sal)
|| ' is greater than  maximum salary '
|| TO_CHAR(max_sal) || ' for  job classification ' || jobid );
ELSIF sal < min_sal then
DBMS_OUTPUT.PUT_LINE('Warning: ' || TO_CHAR(sal)
```

```
|| ' is less than  minimum salary '
|| TO_CHAR(min_sal) || ' for  job classification ' || jobid );
END IF;
END hire_employee;
```

პროცედურის remove_employee კოდი, რომელიც შლის არსებულ employee -ს:

```
PROCEDURE remove_employee (empid NUMBER) IS
firstname employees.first_name%TYPE;
lastname employees.last_name%TYPE;
BEGIN
SELECT first_name, last_name INTO firstname, lastname FROM
employees
WHERE employee_id = empid;
DELETE FROM employees WHERE employee_id = empid;
DBMS_OUTPUT.PUT_LINE('Employee: ' || TO_CHAR(empid) || ', '
|| firstname || ', ' || lastname || ' has been deleted.');
EXCEPTION
WHEN NO_DATA_FOUND შემდეგ
DBMS_OUTPUT.PUT_LINE('Employee ID: ' || TO_CHAR(empid) || '
not found.');
END remove_employee;
```

ფუნქციის emp_sal_ranking კოდი:

```
FUNCTION emp_sal_ranking (empid NUMBER) RETURN
NUMBER IS
minsal employees.salary%TYPE;
maxsal employees.salary%TYPE;
jobid employees.job_id%TYPE;
sal employees.salary%TYPE;
BEGIN
SELECT job_id, salary INTO jobid, sal FROM employees
WHERE employee_id = empid;
SELECT min_salary, max_salary INTO minsal, maxsal FROM jobs
WHERE job_id = jobid;
RETURN ((sal - minsal)/(maxsal - minsal));
END emp_sal_ranking;
END emp_actions;
/
```

პაკეტების რედაქტირება

პაკეტების და პაკეტის ტანების რედაქტირებისათვის შეგვიძლია გამოვიყენოთ: Object Browser გვერდი, SQL Commands გვერდი ან SQL CREATE OR REPLACE ბრძანება SQL Command Line -ში.

ა) SQL Commands გვერდში პაკეტის რედაქტირებისათვის:

1. დავრეგისტრირდეთ Database Home გვერდში.

2. On home გვერდი, ვაწკაპუნებთ იკონაზე **SQL.** ჩნდება SQL გვერდი.

3. ვაწკაპუნებთ იკონაზე **SQL Commands.** ჩნდება SQL Commands გვერდი.

4. ვაწკაპუნებთ ჩანართზე **Saved SQL.** ჩნდება შენახული SQL მოდულები.

5. ვაწკაპუნებთ SQL სახელზე, რომელის რედაქტირებაც გვინდა.

6. ვცვლით საწყის კოდს პაკეტისათვის. ვაწკაპუნებთ ღილაკზე **Run**, თუ პაკეტის შესრულება გვინდა.

7. დასასრულს, ვაწკაპუნებთ ღილაკზე **Save**.

ბ) Object Browser გვერდში პაკეტის რედაქტირებისათვის:

1. დავრეგისტრირდეთ Database Home გვერდზე. მაგალითის სახით გამოვიყენოთ HR მომხმარებელი .

2. On Database Home გვერდი, ვაწკაპუნებთ იკონაზე **Object Browser**.

3. ობიექტების სიიდან ვირჩევთ **Packets** და შემდეგ ვაწკაპუნებთ პაკეტზე, რომლის ეკრანზე გამოტანა გვინდა. ჩნდება პაკეტის სპეციფიკაცია.

4. ვაწკაპუნებთ ღილაკზე **Edit.** არსებული საწყისი კოდის რედაქტირებისათვის ვაწკაპუნებთ ჩანართზე **Body**.

5. ვაწკაპუნებთ ღილაკზე **Compile**, რათა დავრწმუნდეთ ცვლილებათა უშეცდომოდ შესრულებაში.

კომპილაციის შემდეგ პაკეტი ასევე ინახავს შესრულებულ ცვლილებებს.

პაკეტების წაშლა

ჩვენ შეგვიძლია გამოვიყენოთ SQL DROP ბრძანება ან Object Browser გვერდი პაკეტების და პაკეტის ტანების წაშლისათვის. Object Browser გვერდის გამოყენების შემთხვევაში:

1. დავრეგისტრირდეთ Database Home გვერდში.

2. Database Home გვერდზე ვაწკაპუნებთ იკონაზე **Object Browser**.

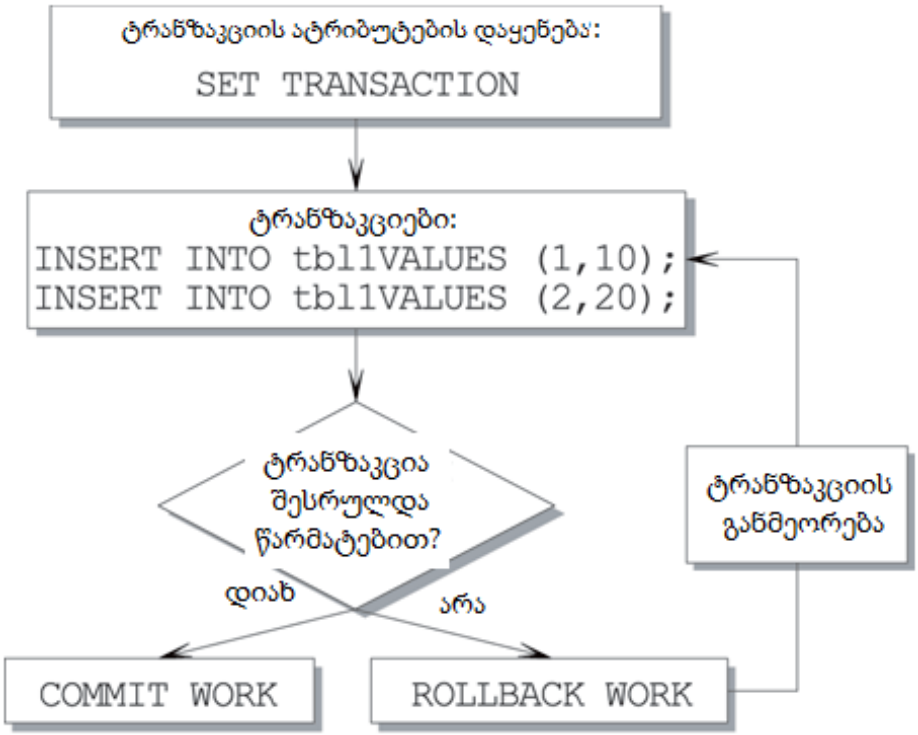
3. ობიექტების სიიდან ვირჩევთ **Packets** და ვაწკაპუნებთ პაკეტზე, რომლის ეკრანზე გამოტანა გვინდა.

4. ვაწკაპუნებთ ღილაკზე **Drop**. ჩვენ შეგვიძლია დავაწკაპუნოთ ჩანართზე **Body** და შემდეგ ღილაკზე **Drop**, თუ მხოლოდ პაკეტის ტანის წაშლა გვინდა.

5. ვაწკაპუნებთ ღილაკზე **Finish** პაკეტის სპეციფიკაციის ან პაკეტის ტანის წაშლის დადასტურებისათვის.

ტრანზაკციების მართვა

ტრანზაკციების მართვის პროცესი ზოგადად შეიძლება წარმოვიდგინოთ შემდეგი სქემის სახით.



ნახ. 10.5

PL/SQL ახორციელებს ტრანზაკციების მართვას შემდეგი ბრძანებების მეშვეობით: COMMIT, ROLLBACK, ROLLBACK TO SAVEPOINT, SAVEPOINT, SET TRANSACTION, LOCK ცხრილი.

COMMIT ოპერატორი

როდესაც ჩვენ ვუშვებთ COMMIT-ს, ამით მოცემული სესიის მეშვეობით ყველა ცვლილება ცხადად იქნება შეტანილი მიმდინარე ტრანზაკციაში. COMMIT ბრძანების სინტაქსისი არის:

COMMIT [WORK] [COMMENT *text*];

სადაც: სიტყვა WORK არ არის აუცილებელი და გამოიყენება მხოლოდ უკეთესად წაკითხვის თვალსაზრისით.

COMMIT ბრძანების გამოყენება მართებულია შემდეგი სახით:

```
COMMIT;
COMMIT WORK;
COMMIT COMMENT 'maintaining account balance'.
```

მაგალითი: COMMIT ბრძანების გამოყენება WRITE -თან ერთად

```
CREATE ცხრილი accounts (account_id NUMBER(6), balance NUMBER
(10,2));
INSERT INTO accounts VALUES (7715, 6350.00);
INSERT INTO accounts VALUES (7720, 5100.50);
DECLARE
transfer NUMBER(8,2) := 250;
BEGIN
UPDATE accounts SET balance = balance - transfer WHERE account_id =
7715;
UPDATE accounts SET balance = balance + transfer WHERE account_id =
7720;
COMMIT COMMENT 'Transfer From 7715 to 7720' WRITE
IMMEDIATE NOWAIT;
END;
/
```

ROLLBACK ოპერატორი

ROLLBACK ოპერატორის სინტაქსისი შემდეგია:

```
ROLLBACK [WORK] [TO [SAVEPOINT
] savepoint_name];
```

არსებობს ROLLBACK -ის გამოყენების ორი გზა: პარამეტრების გარეშე ან TO პირობით იმ savepoint-ის მითითებით, რომელშიც ROLLBACK უნდა გაჩერდეს.

ქვემოთ მოყვანილია ROLLBACK ყველა შესაძლო ვარიანტი:

```
ROLLBACK;
ROLLBACK WORK;
ROLLBACK TO begin_cleanup;
```

მაგალითი: ROLLBACK -ის გამოყენება

```
CREATE ცხრილი emp_name AS SELECT employee_id, last_name
FROM employees;
CREATE UNIQUE INDEX empname_ix ON emp_name (employee_id);
CREATE ცხრილი emp_sal AS SELECT employee_id, salary FROM
employees;
CREATE UNIQUE INDEX empsal_ix ON emp_sal (employee_id);
CREATE ცხრილი emp_job AS SELECT employee_id, job_id FROM
employees;
CREATE UNIQUE INDEX empjobid_ix ON emp_job (employee_id);
DECLARE
emp_id NUMBER(6);
emp_lastname VARCHAR2(25);
emp_salary NUMBER(8,2);
emp_jobid VARCHAR2(10);
BEGIN
SELECT employee_id, last_name, salary, job_id INTO emp_id,
emp_lastname,
emp_salary, emp_jobid FROM employees WHERE employee_id = 120;
INSERT INTO emp_name VALUES (emp_id, emp_lastname);
INSERT INTO emp_sal VALUES (emp_id, emp_salary);
INSERT INTO emp_job VALUES (emp_id, emp_jobid);
EXCEPTION
WHEN DUP_VAL_ON_INDEX შემდეგ
ROLLBACK;
DBMS_OUTPUT.PUT_LINE('Inserts have been rolled back');
END;
/
```

SAVEPOINT ოპერატორი

ეს მარკერი მიუთითებს ROLLBACK TO -ს იმ წერტილს, რომლის შემდეგ ნებისმიერი ცვლილება უნდა წაიშალოს, ნებისმიერი ბლოკირება უნდა გაიხსნას, მაგრამ savepoint-მდე არსებული ყველა ცვლილება და ბლოკირება უნდა შენარჩუნდეს.

SAVEPOINT ოპერატორის სინტაქსისი შემდეგია:

SAVEPOINT *savepoint_name*;

მაგალითი:

```
BEGIN
INSERT INTO student
( student_id, Last_name, zip, registration_date,
created_by, created_date, modified_by,
modified_date
)
VALUES ( student_id_seq.nextval, 'Tashi', 10015,
'01-JAN-99', 'STUDENTA', '01-JAN-99',
'STUDENTA','01-JAN-99'
);
SAVEPOINT A;
INSERT INTO student
( student_id, Last_name, zip, registration_date,
created_by, created_date, modified_by,
modified_date
)
VALUES (student_id_seq.nextval, 'Sonam', 10015,
'01-JAN-99', 'STUDENTB','01-JAN-99',
'STUDENTB', '01-JAN-99'
);
SAVEPOINT B;
INSERT INTO student
( student_id, Last_name, zip, registration_date,
created_by, created_date, modified_by,
modified_date
)
VALUES (student_id_seq.nextval, 'Norbu', 10015,
```

```
'01-JAN-99', 'STUDENTB', '01-JAN-99',
'STUDENTB', '01-JAN-99'
);
SAVEPOINT C;
ROLLBACK TO B;
END;
```

მაგალითი: SAVEPOINT -ის გამოყენება ROLLBACK -თან ერთად.

```
CREATE ცხრილი emp_name AS SELECT employee_id,
last_name, salary FROM employees;
CREATE UNIQUE INDEX empname_ix ON emp_name
(employee_id);
DECLARE
emp_id employees.employee_id%TYPE;
emp_lastname employees.last_name%TYPE;
emp_salary employees.salary%TYPE;
BEGIN
SELECT employee_id, last_name, salary INTO emp_id,
emp_lastname,
emp_salary FROM employees WHERE employee_id = 120;
UPDATE emp_name SET salary = salary * 1.1 WHERE
employee_id = emp_id;
DELETE FROM emp_name WHERE employee_id = 130;
SAVEPOINT do_insert;
INSERT INTO emp_name VALUES (emp_id, emp_lastname,
emp_salary);
EXCEPTION
WHEN DUP_VAL_ON_INDEX შემდეგ
ROLLBACK TO do_insert;
DBMS_OUTPUT.PUT_LINE('Insert has been rolled back');
END;
/
```

SET TRANSACTION ოპერატორი

SET TRANSACTION ოპერატორი წარმოადგენს ტრანზაკციის SQL კოდის პირველ ოპერატორს და ის შესაძლოა მხოლოდ ერთხელ იქნას მოყვანილი. არსებობს ამ ოპერატორის ოთხი სახესხვაობა:

```
SET TRANSACTION READ ONLY;
```

SET TRANSACTION READ WRITE;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE | READ COMMITTED;
SET TRANSACTION USE ROLLBACK SEGMENT rollback_segname;

მაგალითი:

SET TRANSACTION-ის გამოყენება Begin a Read-only Transaction -ში.

```
DECLARE
daily_order_total NUMBER(12,2);
weekly_order_total NUMBER(12,2);
monthly_order_total NUMBER(12,2);
BEGIN
COMMIT; -- ends previous transaction
SET TRANSACTION READ ONLY NAME 'Calculate Order
Totals';
SELECT SUM (order_total) INTO daily_order_total FROM orders
WHERE order_date = SYSDATE;
SELECT SUM (order_total) INTO weekly_order_total FROM
orders
WHERE order_date = SYSDATE - 7;
SELECT SUM (order_total) INTO monthly_order_total FROM
orders
WHERE order_date = SYSDATE - 30;
COMMIT; -- ends read-only transaction
END;
/
```

LOCK ცხრილი ოპერატორი

ეს ოპერატორი მონაცემთა ბაზის რომელიმე ცხრილის ბლოკირების საშუალებას იძლევა. მისი სინტაქსისი შემდეგია:

LOCK ცხრილი

ცხრილი_reference_list IN *lock_mode* MODE [NOWAIT];

სადაც : *ცხრილი_reference_list* არის ერთ ან რამდენიმე ცხრილზე მიმართვის სიას, ხოლო *lock_mode* მიუთითებს ბლოკირების სახეობაზე:

ROW SHARE
ROW EXCLUSIVE

SHARE UPDATE
SHARE
SHARE ROW EXCLUSIVE
EXCLUSIVE

თუ გამოვიყენებთ NOWAIT გასაღებურ სიტყვას, მაშინ Oracle უკვე აღარ დაელოდება ბლოკირების ბრძანებას, როცა ამ ცხრილის ბლოკირება უკვე განხორციელდა სხვა მომხმარებლის მიერ. თუ NOWAIT გასაღებურ სიტყვას გამოვტოვებთ, მაშინ Oracle მოიცდის მანამ, სანამ ეს ცხრილი მოქმედებაშია. არსებობს „LOCK ცხრილი“ ოპერატორის შემდეგი ვარიაციები:

LOCK ცხრილი emp IN ROW EXCLUSIVE MODE;

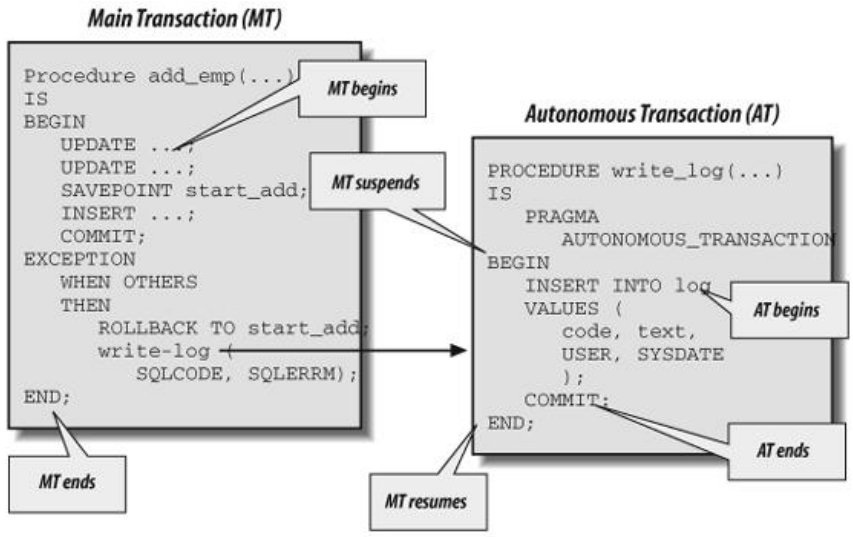
LOCK ცხრილი emp, dept IN SHARE MODE NOWAIT;

LOCK ცხრილი scott.emp@new_york IN SHARE UPDATE MODE;

ავტონომიური ტრანზაკციები

სხვაგვარად მათ შეიძლება „ჩასმული ტრანზაკციებიც“ ვუწოდოთ, როცა ტრანზაკცია იქმნება ტრანზაკციის შიგნით, რომელიც აფიქსირებს ან აუქმებს ცვლილებას „მშობელი“ ტრანზაკციისაგან დამოუკიდებლად. ჩვენ შეგვიძლია შევქმნათ ავტონომიური ტრანზაკცია ბრძანებით:

PRAGMA AUTONOMOUS_TRANSACTION;



ნახ. 10.6

მაგალითი:

პროცედურის წარმოდგენა ავტონომიური ტრანზაკციის სახით.

```
CREATE PROCEDURE lower_salary (emp_id NUMBER, amount
NUMBER) AS
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
UPDATE employees SET salary = salary - amount WHERE
employee_id = emp_id;
COMMIT;
END lower_salary;
```

მაგალითი: Autonomous Function -ის დეკლარირება პაკეტში

```
CREATE OR REPLACE პაკეტი emp_actions AS -- პაკეტი
სპეციფიკაცია
FUNCTION raise_salary (emp_id NUMBER, sal_raise NUMBER)
RETURN NUMBER;
END emp_actions;
/
CREATE OR REPLACE პაკეტი BODY emp_actions AS --
პაკეტი body
-- კოდი for function raise_salary
FUNCTION raise_salary (emp_id NUMBER, sal_raise NUMBER)
RETURN NUMBER IS
PRAGMA AUTONOMOUS_TRANSACTION;
new_sal NUMBER(8,2);
BEGIN
UPDATE employees SET salary = salary + sal_raise WHERE
employee_id = emp_id;
COMMIT;
SELECT salary INTO new_sal FROM employees WHERE
employee_id = emp_id;
RETURN new_sal;
END raise_salary;
END emp_actions;
/
```

მაგალითი: ავტონომიური PL/SQL Block -ის დეკლარირება

```
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
```

```
emp_id NUMBER(6);
amount NUMBER(6,2);
BEGIN
emp_id := 200;
amount := 200;
UPDATE employees SET salary = salary - amount WHERE
employee_id = emp_id;
COMMIT;
END;
/
```

მაგალითი: ავტონომიური ტრიგერის დეკლარირება.

```
CREATE ცხრილი emp_audit ( emp_audit_id NUMBER(6),
up_date DATE,
new_sal NUMBER(8,2), old_sal NUMBER(8,2) );
CREATE OR REPLACE TRIGGER audit_sal
AFTER UPDATE OF salary ON employees FOR EACH ROW
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
-- bind variables are used here for values
INSERT INTO emp_audit VALUES( :old.employee_id,
SYSDATE,
:new.salary, :old.salary );
COMMIT;
END;
/
```

მაგალითი: ავტონომიური ფუნქციის გამოძახება SQL -დან.

```
-- create  debug ცხრილი
CREATE ცხრილი debug_output (msg VARCHAR2(200));
-- create  პაკეტი spec
CREATE პაკეტი debugging AS
FUNCTION log_msg (msg VARCHAR2) RETURN VARCHAR2;
PRAGMA RESTRICT_REFERENCES(log_msg, WNDS, RNDS);
END debugging;
/
-- create  პაკეტი body
CREATE პაკეტი BODY debugging AS
```

```
FUNCTION log_msg (msg VARCHAR2) RETURN VARCHAR2 IS
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
--  following insert does not violate  constraint
-- WNDS because this is an autonomous routine
INSERT INTO debug_output VALUES (msg);
COMMIT;
RETURN msg;
END;
END debugging;
/
-- call  პაკეტით function from a query
DECLARE
my_emp_id NUMBER(6);
my_last_name VARCHAR2(25);
my_count NUMBER;
BEGIN
my_emp_id := 120;
SELECT debugging.log_msg(last_name) INTO my_last_name FROM
employees
WHERE employee_id = my_emp_id;
-- even if you roll back in this scope,  insert into 'debug_output' remains
-- committed because it is part of an autonomous transaction
ROLLBACK;
END;
/
```

ლაბორატორიული სამუშაო 11

ფორმები *(Forms)*

სამუშაოს მიზანი:

1. ფორმების შექმნა Wizard-ის გამოყენებით.
2. Tabular Form-ის შექმნა.
3. Master Detail Form-ის შექმნა.
4. Form-ის ხელით შექმნა.
5. მომხმარებლის ვალიდაცია ფორმებში.

მონაცემთა ბაზებთან სამუშაო დანართის შემუშავებისათვის გამოიყენება სამი ტიპის ფორმა:

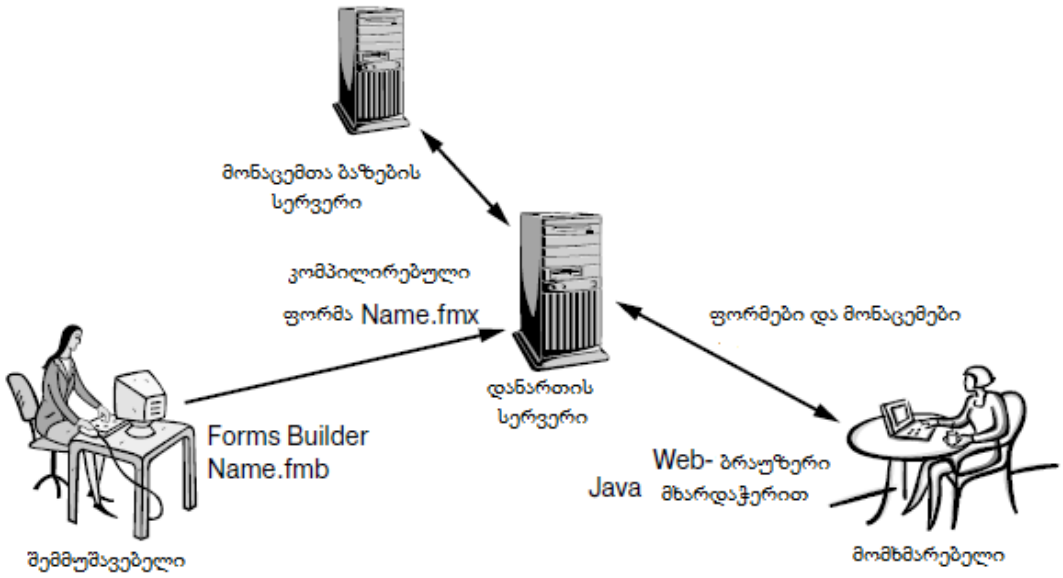
1. ძირითადი, სადაც დროის მოცემულ მომენტში აისახება ერთ სტრიქონში შემავალი მონაცემები;

2. ცხრილური (Tabular), რომელსაც ერთ გვერდზე ერთდროულად გამოაქვს ცხრილის რამდენიმე სტრიქონი;

3. კომბინირებული, სადაც პირველადი ცხრილის მონაცემები გამოიტანება ძირითადი ფორმის სახით, ხოლო დაქვემდებარებული ცხრილის მონაცემები - ცხრილური ფორმის სახით.

Oracle - ის ადრეულ ვერსიებში ფორმებისათვის ძირითადად გამოიყენებოდა კლიენტ/სერვერული არქიტექტურა. ამასთან ფორმები სრულდებოდა კლიენტურ ნაწილში, ხოლო მონაცემების გამოძახება ხდებოდა სერვერიდან.

Oracle -ის თანამედროვე ვერსიებში ფორმები წარმოადგენენ Web - დანართებს Java-ს საფუძველზე. ფორმების მხარდამჭერი გარემოს არქიტექტურა მოცემულია ნახ. 11.1.- ზე.



ნახ. 11.1

ფორმების შექმნა

ფორმების შექმნა Wizard-ის გამოყენებით

ფორმების შექმნისათვის Wizard-ის გამოყენებით:

1. home page-ის სამუშაო სივრცეში (Workspace), ვაწკაპუნებთ იკონაზე **Application Builder**.
2. ვირჩევთ აპლიკაციას.
3. ვაწკაპუნებთ **Create Page**.
4. ვირჩევთ **Form** და ვაწკაპუნებთ **Next**.
5. ვირჩევთ ფორმის ტიპს.

Tabular Form-ის შექმნა

ცხრილური ფორმის შესაქმნელად:

1. home page-ის სამუშაო სივრცეში (Workspace), ვაწკაპუნებთ იკონაზე **Application Builder**.
2. ვირჩევთ აპლიკაციას.
3. ვაწკაპუნებთ **Create Page**.

4. ვირჩევთ **Form** და ვაწკაპუნებთ **Next**.

5. ვირჩევთ **Tabular Form** და ვაწკაპუნებთ **Next**. ჩნდება ცხრილური ფორმის Wizard.

6. ცხრილი/წარმოდგენა მფლობელისათვის (Owner):

ა. განვსაზღვრავთ ცხრილის ან წარმოდგენის მფლობელს, რომლის საფუძველზე გვინდა ცხრილური ფორმის შექმნა.

ბ. ვირჩევთ ცხრილზე შესასრულებელ ოპერაციებს (მაგალითად, Update, Insert და Delete).

გ. ვაწკაპუნებთ **Next**.

7. ვირჩევთ ცხრილის/წარმოდგენის სახელს (Name) და ვაწკაპუნებთ **Next.**

8. ფორმაზე ასახული სვეტებისათვის:

ა. ვირჩევთ სვეტებს ფორმისათვის.

ბ**.** ვაწკაპუნებთ **Next**.

9. Primary Key-სათვის ვირჩევთ პირველადი გასაღების სვეტს და ვაწკაპუნებთ **Next**.

10**.** Primary Key Source-სათვის ვირჩევთ წყაროს ტიპს პირველადი გასაღების სვეტისათვის და ვაწკაპუნებთ **Next**.

11. ფორმაში ასახული ზედა ცხრილისათვის ვირჩევთ სვეტებს, ვაწკაპუნებთ **Next**.

12. Page-სა და Region-ის ატრიბუტებისათვის:

ა. განვსაზღვრავთ Page-სა და Region-ის ინფორმაციას.

ბ. ვირჩევთ Region-ის შაბლონს.

გ. ვირჩევთ ანგარიშის შაბლონს.

დ. ვაწკაპუნებთ **Nex**t.

13. Tab-ზე განვსაზღვრავთ იმპლემენტაციას ამ გვერდისათვის და ვაწკაპუნებთ **Next**.

14. Button Labels-ზე შეგვყავს წარწერა თითოეული ღილაკისათვის და ვაწკაპუნებთ Next.

15. Branching-ზე განვსაზღვრავთ გვერდებს განშტოებებისათვის და ვაწკაპუნებთs **Submit** და **Cancel** ღილაკებზე და შემდეგ ვაწკაპუნებთ **Next**.

16. ვაწკაპუნებთ **Finish**.

Master Detail Form-ის შექმნა

ამ ფორმის გამოყენებით მომხმარებელს შეუძლია მონაცემების შეტანა, შესწორება და წაშლა ორი ცხრილიდან ან წარმოდგენიდან.

Master Detail Form-ის შექმნისათვის:

1. home page-ის სამუშაო სივრცეში (Workspace), ვაწკაპუნებთ იკონაზე **Application Builder**.

2. ვირჩევთ აპლიკაციას.

3. ვაწკაპუნებთ **Create Page**.

4. ვირჩევთ **Form** და ვაწკაპუნებთ **Next**.

5. ვირჩევთ **Master Detail Form** და ვაწკაპუნებთ **Next**.

ჩნდება Master Detail Wizard.

6. **Master** ცხრილზე:

ა. ვირჩევთ ცხრილის ან წარმოდგენის მფლობელს.

ბ. ვირჩევთ ცხრილის ან წარმოდგენის სახელწოდებას.

Available Columns-ის ქვემოთ ჩნდება ამ ობიექტის სვეტები.

გ. ვირჩევთ სვეტებს ფორმისათვის და შემდეგ ვაწკაპუნებთ მათ Displayed Columns-ში გადასატანად.

დ. ვაწკაპუნებთ **Next**.

7. **Detail** ცხრილზე:

ა. ჩვენებისათვის განვსაზღვრავთ მხოლოდ დაკავშირებულ ცხრილებს Yes ან No მეშვეობით.

ბ. ვირჩევთ ცხრილის ან წარმოდგენის მფლობელს.

გ. ვირჩევთ ცხრილის ან წარმოდგენის სახელწოდებას.

Available Columns-ის ქვემოთ ჩნდება ამ ობიექტის სვეტები.

დ. ვირჩევთ სვეტებს ფორმისათვის და შემდეგ ვაწკაპუნებთ მათ Displayed Columns-ში გადასატანად.

ე. ვაწკაპუნებთ **Next**.

8. Primary Key-ზე:

ა. ვირჩევთ Master ცხრილისათვის მაქსიმუმ ორ პირველად გასაღებს.

ბ. ვირჩევთ Detail ცხრილისათვის მაქსიმუმ ორ პირველად გასაღებს.

გ. ვაწკაპუნებთ **Next**.

9. Master Detail Link-სათვის განვსაზღვრავთ დამოკიდებულებას master და detail ცხრილებს შორის და ვაწკაპუნებთ **Next**.

10. Primary Key Source-სათვის:

ვირჩევთ პირველადი გასაღების სვეტს master ცხრილისათვის და ვაწკაპუნებთ **Next**.

შემდეგ ვირჩევთ პირველადი გასაღების სვეტს detail ცხრილისათვის და ვაწკაპუნებთ **Next**.

11. Master Options-სათვის: განვსაზღვრავთ ჩანაწერების ნავიგატორს და ვაწკაპუნებთ **Next**.

12. Layout-სათვის: განვსაზღვრავთ თუ როგორ უნდა ავაგოთ master detail და ვაწკაპუნებთ **Next**.

13. Page Attributes-სათვის: ვახდენთ master გვერდისა და detail გვერდის ინფორმაციის რედაქტირებას და შემდეგ ვაწკაპუნებთ **Next.**

14. Tab-სათვის: განვსაზღვრავთ თუ რას უნდა შეიცავდეს ჩანართები და ვაწკაპუნებთ **Next**.

15. ვაწკაპუნებთ **Create**.

ფორმის ხელით შექმნა

ფორმის ხელით შექმნისათვის სრულდება შემდეგი ბიჯები:

1. გადავაადგილდეთ შესაბამის Page Definition.
2. შევქმნათ HTML არე (region):
    ა. Region-ის ქვემოთ ვაწკაპუნებთ Create icon-ზე.
    ბ. ვირჩევთ HTML ტიპს.
    გ. მივყვებით ეკრანზე მოცემულ ინსტრუქციას.
3. გვერდზე ახალი პუნქტის (Items) დამატებისათვის:
    - Items-ის ქვემოთ ვაწკაპუნებთ Create icon.
    - მივყვებით ეკრანზე მოცემულ ინსტრუქციას.

ფორმის დამუშავება

ფორმის შექმნის შემდგომი ბიჯი არის მონაცემთა დამუშავების პროცესი, რომელიც გულისხმობს მონაცემთა ბაზის ცხრილებში ან წარმოდგენებში შეტანისა და განახლების ოპერაციების შესრულებას. განირჩევა ფორმის დამუშავების სამი გზა:

- სტრიქონის ავტომატური დამუშავება.
- პროცესი, რომელიც შეიცავს ერთ ან მეტ შეტანის ბრძანებას.
- PL/SQL API-ის გამოყენება ფორმის მნიშვნელობათა დამუშავებისათვის.

სტრიქონის ავტომატური დამუშავების (DML) პროცესის შექმნა

სტრიქონის ავტომატური დამუშავების (DML) პროცესის შექმნისათვის:

1. გადავაადგილდეთ შესაბამის Page Definition.
2. Processes-ის ქვემოთ ვაწკაპუნებთ Create icon.
3. ვირჩევთ Data Manipulation.
4. ვირჩევთ დამუშავების კატეგორიას - Automatic Row Processing (DML).
5. განვსაზღვრავთ დამუშავების პროცესის ატრიბუტებს:

ა. ველში Name შეგვაქვს სახელი.

ბ. ველშიSequence განვსაზღვრავთ თანამიმდევრობის რაოდენობას.

გ. პუნქტების (Point) სიიდან ვირჩევთ შესაბამის პუნქტს. უმეტესწილად ვირჩევთ Onload - After Header.

დ. ტიპების (Type) სიიდან ვირჩევთ Automated Row Processing (DML).

6. მივყვებით ეკრანზე მოცემულ ინსტრუქციას.

პროცესის შექმნა, რომელიც შეიცავს ერთ ან მეტ შეტანის ბრძანებას

მაგალითისათვის, დავუშვათ გვაქვს ფორმა სამი პუნქტით (Items):

- P1_ID - დაფარული item მიმდინარე სტრიქონის პირველადი გასაღების ცხრილში შენახვისათვის.
- P1_FIRST_NAME - ტექსტური ველი შეტანისათვის.
- P1_LAST_NAME - ტექსტური ველი შეტანისათვის.

დავუშვათ რომ გვაქვს კიდევ სამი ღილაკი წარწერებით: Insert, Update, და Delete. დავუშვათ რომ გვაქვს აგრეთვე ცხრილი T, რომელიც შეიცავს სვეტებს: id, first_name, და last_name. ცხრილს გააჩნია ტრიგერი, რომელიც ავტომატურად ავსებს ID სვეტს, როცა მასში შესატანი მნიშვნელობა არ არსებობს.

ახალი სტრიქონის შეტანის პროცესისათვის იქმნება PL/SQL ტიპის პროცესი, რომელიც სრულდება როცა ვაწკაპუნებთ Insert ღილაკზე. მაგალითად:

```
BEGIN
INSERT INTO T ( first_name, last_name )
VALUES (:P1_FIRST_NAME, :P1_LAST_NAME);
END;
```

სტრიქონის განახლების პროცესისათვის სრულდება PL/SQL ტიპის სხვა პროცესი. მაგალითად:

```
BEGIN
UPDATE T
SET first_name = :P1_FIRST_NAME,
last_name = :P1_LAST_NAME
WHERE ID = :P1_ID;
END;
```

სტრიქონის წაშლის პროცესისათვის იქმნება სულ სხვა PL/SQL ტიპის პროცესი, რომელიც სრულდება როცა ვაწკაპუნებთ Delete ღილაკზე. მაგალითად:

```
BEGIN
DELETE FROM T
WHERE ID = :P1_ID;
END;
```

ფორმის მნიშვნელობათა დამუშავება PL/SQL API-ის გამოყენებით

ამ შემთხვევაში ფორმის შევსება ხდება შემდეგნაირად:

1. გადავაადგილდეთ შესაბამის Page Definition.
2. Processes-ის ქვემოთ ვაწკაპუნებთ Create icon.
3. ვირჩევთ Data Manipulation.
4. ვირჩევთ დამუშავების კატეგორიას - **Automatic Row Fetch**.
5. განვსაზღვრავთ დამუშავების პროცესის ატრიბუტებს:

ა. ველში Name შეგვაქვს სახელი.

ბ. ველშიSequence განვსაზღვრავთ თანამიმდევრობის რაოდენობას.

გ. პუნქტების (Point) სიიდან ვირჩევთ შესაბამის პუნქტს. უმეტესწილად ვირჩევთ Onload - After Header.

დ. ტიპების (Type) სიიდან ვირჩევთ **Automated Row Fetch**.

6. მივყვებით ეკრანზე მოცემულ ინსტრუქციას.

ხელით შევსების მაგალითის სადემონსტრაციოდ შეიძლება გამოვიყენოთ PL/SQL ტიპის Oracle Application Express შემდეგი კოდი, სადაც P2_ID ფარული იტემის მიმართვის მეშვეობით ხდება ename და sal მნიშვნელობების მინიჭება.

```
FOR C1 in (SELECT ename, sal
FROM emp WHERE ID=:P2_ID)
LOOP
:P2_ENAME := C1.ename;
:P2_SAL := C1.sal;
END LOOP;
```

ლაბორატორიული სამუშაო № 12

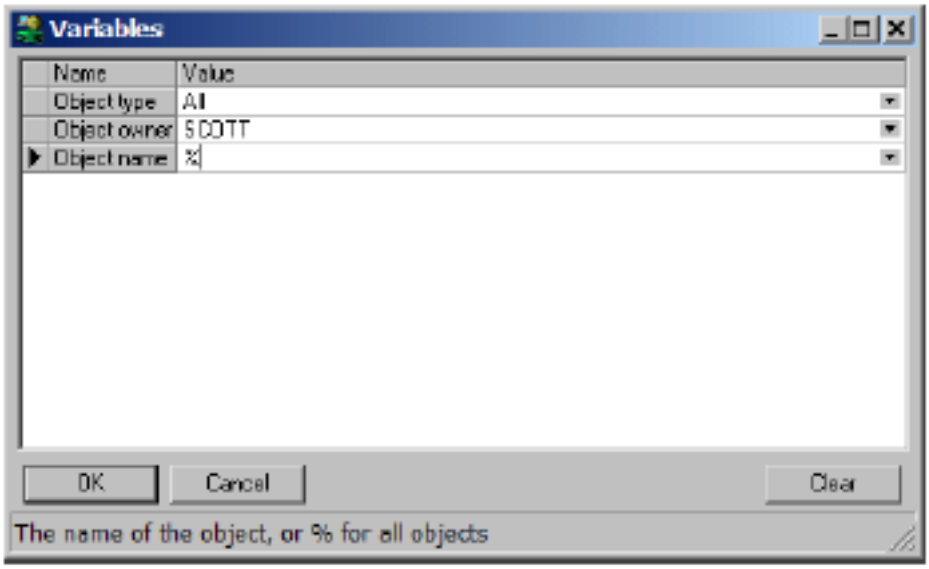
ანგარიშები *(Reports)*

სამუშაოს მიზანი:

1. მარტივი ანგარიშიანი დანართის შემუშავების შესწავლა;
2. დეტალური ანგარიშიანი დანართის შემუშავების შესწავლა;
3 .„მთავარი - დაქვემდებარებული“ ანგარიშიანი დანართის შემუშავების შესწავლა.

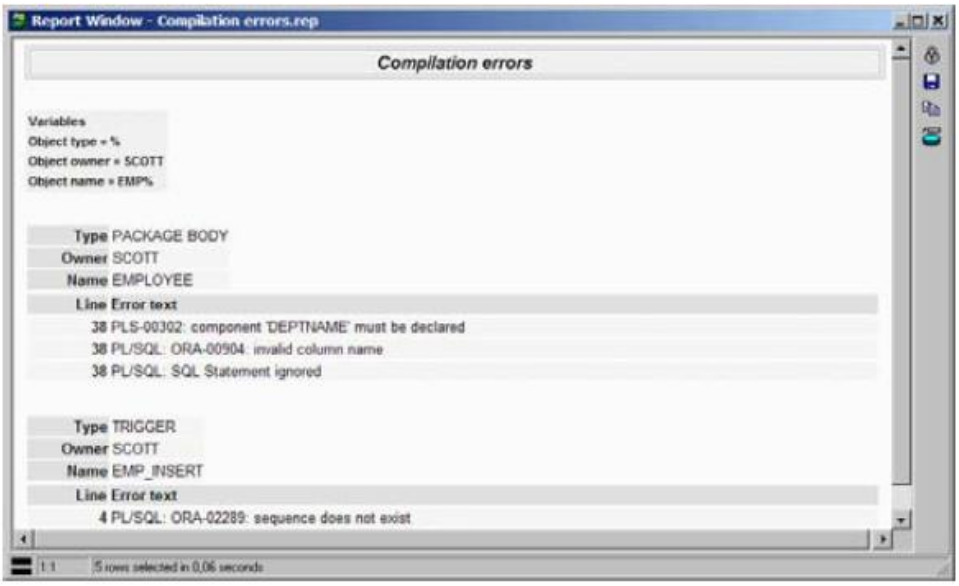
PL/SQL Developer-ის გამოყენებით შესაძლებელია როგორც სტანდარტული, ისე საკუთარი ანუ არჩევითი ანგარიშების შექმნა.

სტანდარტული ანგარიშები ხელმისაწვდომია *Reports* მთავარი მენიუს მეშვეობით. თუ, მაგალითად, ვირჩევთ *Compilation errors* ანგარიშს, მაშინ ჯერ უნდა დაყენდეს ობიექტის ტიპი, მფლობელი და სახელი, რომლისთვისაც გვინდა აისახოს მიმდინარე კომპილაციის შეცდომები:



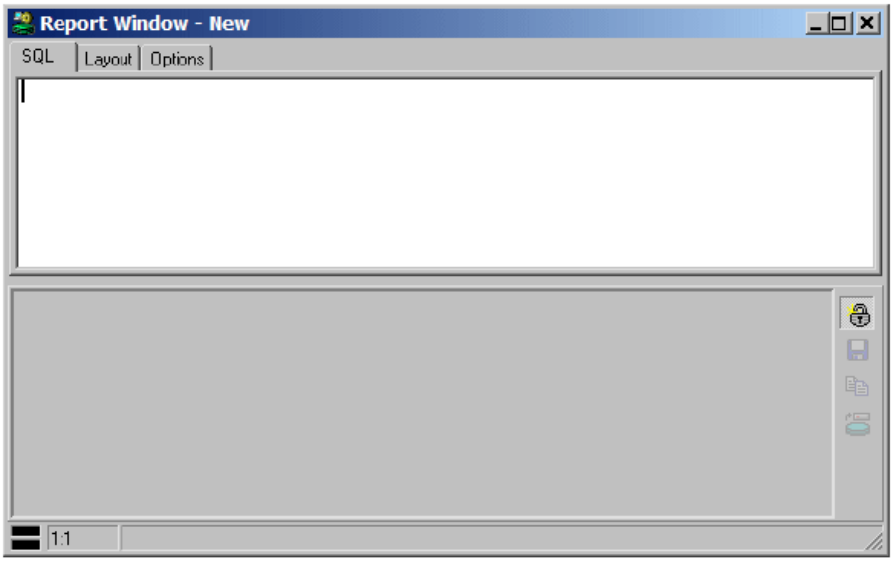
ნახ. 12.1

აღნიშნული მნიშვნელობების შეტანის შემდეგ ვაჭერთ ღილაკს *OK* და ანგარიში ეკრანზე აისახება:



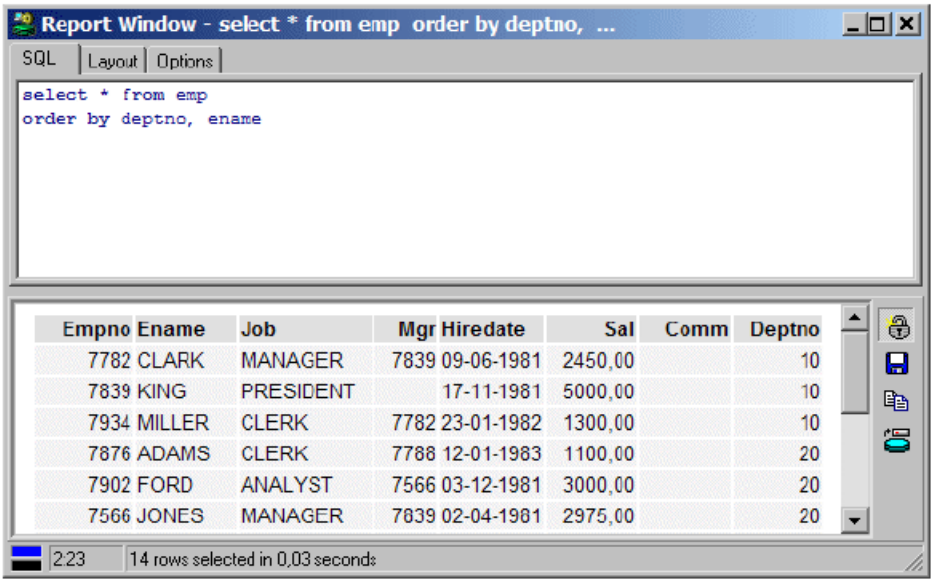
ნახ. 12.2

რაც შეეხება არჩევით ანგარიშებს, მათი შექმნა ძალიან ადვილია. ჯერ ინსტრუმენების პანელზე *New* ღილაკზე დაჭერით და popup მენიუს *Report Window* იტემის არჩევით ვქმნით ახალ, ცარიელ ანგარიშს:



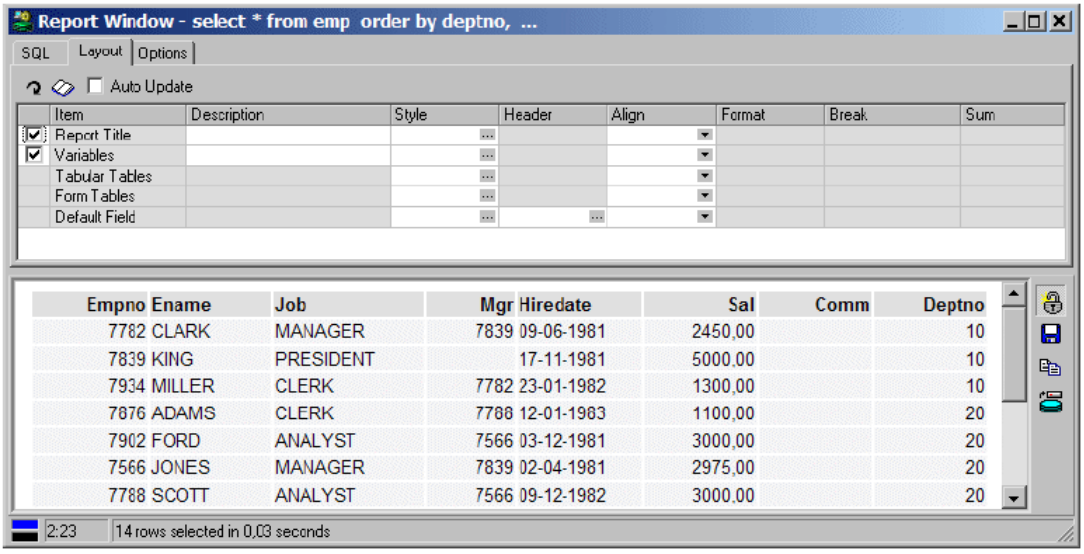
ნახ. 12.3

ავკრიფოთ *emp* ცხრილისათვის select ინსტრუქცია და ინსტრუმენების პანელზე ვაჭერთ ღილაკს *Execute*:



ნახ. 12.4

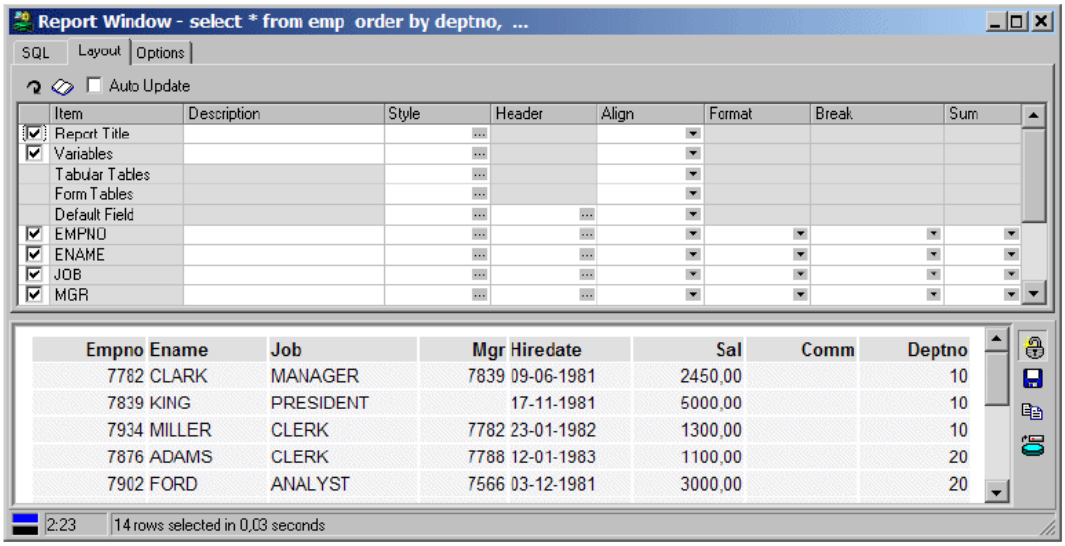
ახალი Report Window-ის გახსნის , select ინსტრუქციის აკრეფისა ანგარიშის შესრულებაზე გაშვების შემდეგ უნდა განისაზღვროს სტილის თვისებები. სტანდარტული თვისებების (სტილი, ფერი და ა.შ.) მისაღებად ჩავრთოთ *Layout* tab page.



ნახ. 12.5

ჩვენ შეგვიძლია აგრეთვე ყველა იტემისათვის განვსაზღვროთ სხვადასხვა layout თვისებები (ანგარიშის სახელი, ცვლადები, ცხრილები

და ველები). ინდივიდუალური ველის თვისებების განსაზღვრისათვის toolbar -ზე ვაჭერთ ღილაკს *Refresh Fieldlist*:



ნახ. 12.6

შედეგად ყველა ველი იქნება ჩართული layout ბადეში და შესაძლებელი გახდება layout თვისებების დაყენება ყოველი ინდივიდუალური ველისათვის. თუ ცარიელი ველისათვის გამოტოვებთ *Style, Header* ან *Align* property, მაშინ შესაბამისი თვისება *Default Field* -დან იქნება გამოყენებული. შესაძლებელია აგრეთვე toolbar -ზე *Auto Update* ოპციის ჩართვა.

დისპლეიზე ასახვა

დისპლეიზე ასახული ანგარიშის სტრუქტურა შედგება შემდეგი კომპონენტებისაგან:

სათაური ***Report Title***.

Variables.

Style

Header Align (Left, Right, Center, Default, None)

Format

მაგალითად:

Format	Value = 1234	Value = 5	Value = 0.1
999G990D00	1,234.00	5.00	0.10
9G990	1,234	5	0
0000	1234	0005	0000
0"%"	1234%	5%	0%
0D000e+00	1.234e+03	5.000e+00	1.000e-01
0D000e-0	1.234e3	5.000e0	1.000e-1

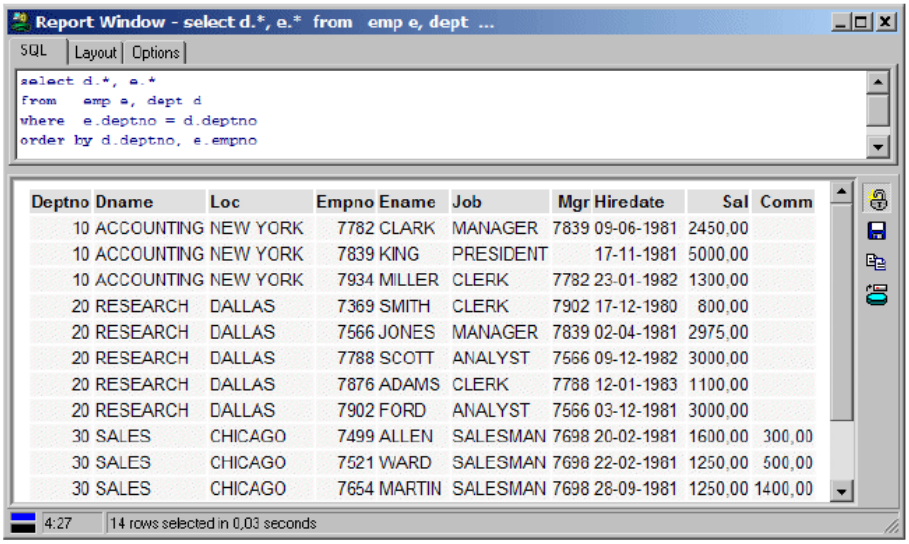
ნახ. 12.7

Break

break layout property ანგარიშის შედეგების სტრუქტურიზების საშუალებას იძლევა. მაგალითად, მოთხოვნისათვის:

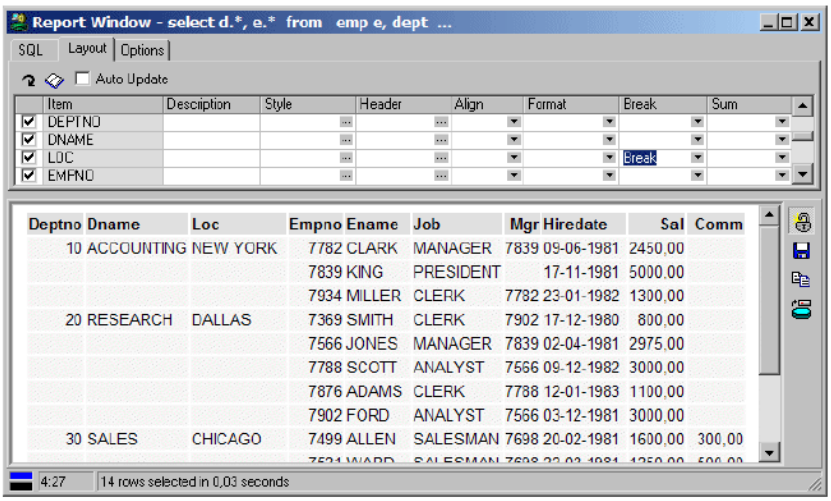
```
select d.*, e.*
from emp e, dept d
where e.deptno = d.deptno
order by d.deptno, e.empno
```

შესაძლებელია ცხრილური ანგარიშის მიღება:



ნახ. 12.8

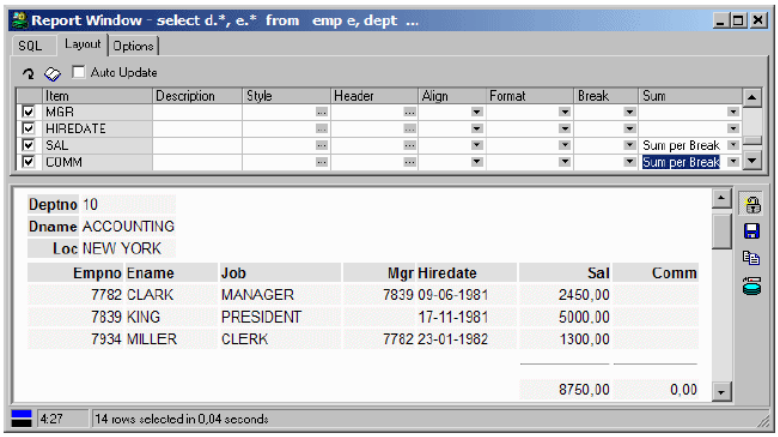
LOC *სვეტზე Break*-ის დაყენებით ანგარიშის შიგნით მივიღებთ შემდეგ სტრუქტურას:



ნახ. 12.9

Sum

თვისება sum საშუალებას იძლევა ჩავრთოთ ჯამი მოცემული ველისათვის:



ნახ. 12.10

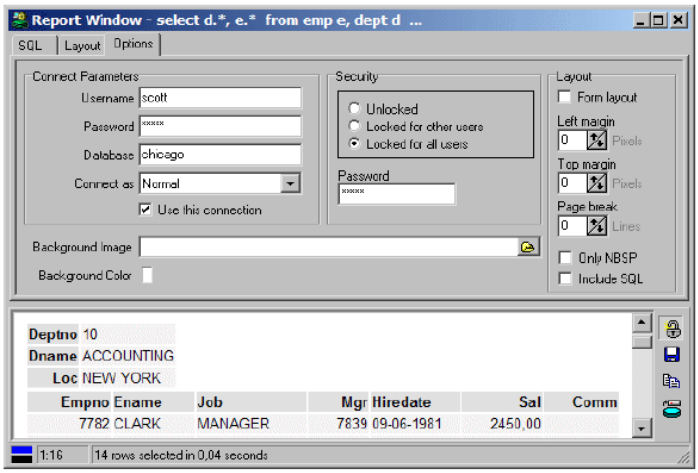
Page headers

Report Title შეიძლება ყოველ გვერდზე განმეორდეს break-ის შემდეგ და აგრეთვე შეიცავდეს შემდეგ ცვლადებს:

- OSUser – მომხმარებლის ოპერაციული სისტემა.
- DBUser – მონაცემთა ბაზასთან მიერთებული მომხმარებელი.
- Database – მომხმარებელთან მიერთებული მონაცემთა ბაზა.
- Date – მიმდინარე თარიღი.
- Time – მიმდინარე დრო.
- Page – მიმდინარე გვერდი.

ოპციები (Options)

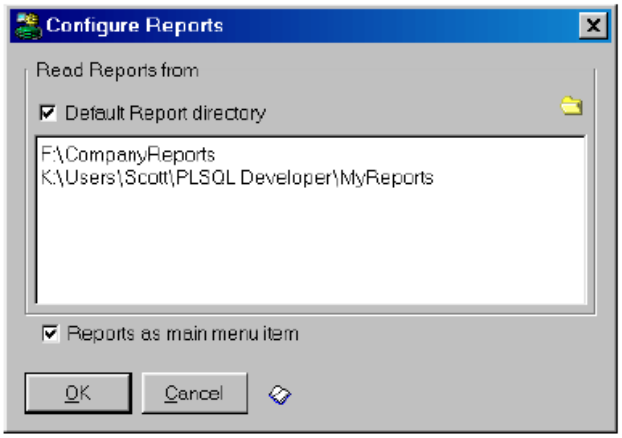
განირჩევა ანგარიშების შემდეგი ოპციები:



ნახ. 12.11

ანგარიშების მენიუ

Reports menu item ჩართულია მთავარ მენიუში და შეიცავს მხოლოდ სტანდარტულ ანგარიშებს, რომლებიც განთავსებული არიან PL/SQL Developer-ის Reports ქვედირექტორიაში. ანგარიშების მენიუდან შერჩეული იტემი ასრულებს ანგარიშს. შეგვიძლია დავამატოთ ან ამოვაგდოთ ანგარიშები ამ დირექტორიიდან. ასევე შეგვიძლია ანგარიშების მენიუს კონფიგურირება *Tools* მენიუში *Conნახ. Reports* იტემის მეშვეობით, რასაც შემდეგი დიალოგი მოჰყვება:



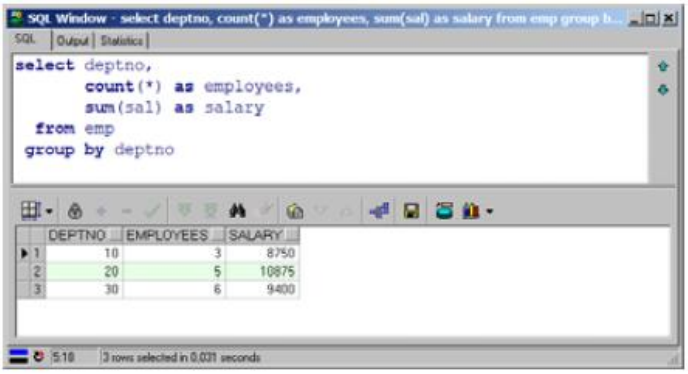
ნახ. 12.12

ოპცია *Default Report directory* მიუთითებს, რომ Reports menu უნდა იქნას ჩართული PL/SQL Developer -დან Reports subdirectory-ში, სადაც

განთავსებულია სტანდარტული ანგარიშები. ჩვენ შეგვიძლია ხელით ავკრიფოთ ან ავირჩიოთ დირექტორია ღილაკზე *Add directory to list* დაჭერით მარჯვნივ.

გრაფიკა

მოთხოვნათა მონაცემების SQL Window ან Report Window-ში გრაფიკულად სწრაფი წარმოდგენისათვის შეგვიძლია გამოვიყენოთ Graph Window. განვიხილოთ შემდეგი მოთხოვნა SQL Window-ში:



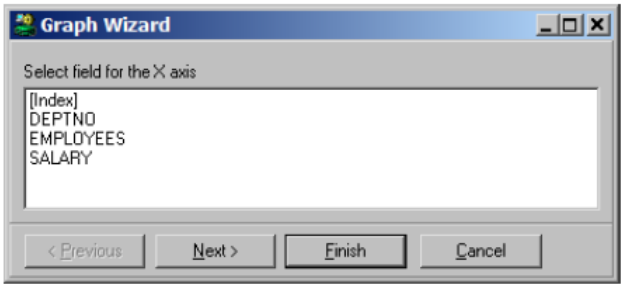
ნახ. 12.13

ვაჭერთ ღილაკს *Graph*: 

ნახ. 12.14

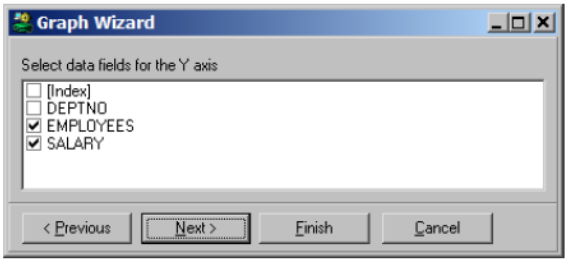
მსგავსი ღილაკი არსებობს Report Window-შიც.

ამოქმედდება *Graph Wizard*. ჩნდება ყველა სვეტი და *[Index]* იტემი, რომლის მნიშვნელობები შეიძლება განთავსდეს X-ღერძზე. თავდაპირველად ვირჩევთ მონაცემებს X-ღერძისათვის:



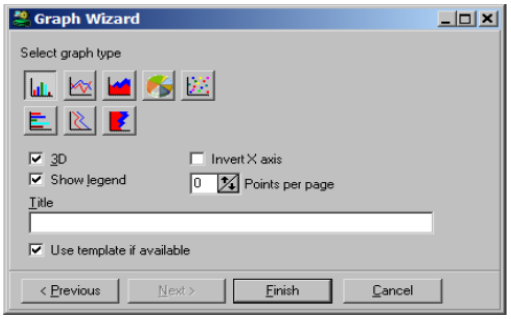
ნახ. 12.15

ვაჭერთ ღილაკს *Next*. შემდეგ ვირჩევთ მონაცემებს Y-ღერძისათვის და ისევ ვაჭერთ ღილაკს *Next*:



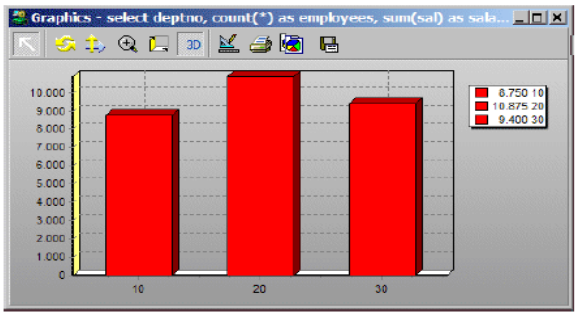
ნახ. 12.16

გრაფიკული სტილის თვისებების შერჩევისათვის ვაჭერთ ღილაკს *Next*:



ნახ. 12.17

და ბოლოს, ვაჭერთ ღილაკს *Finish*, რომლის შემდეგ Graph Window გრაფიკული შედეგები აისახება ეკრანზე:



ნახ. 12.18

Graph Window-ის ინსტრუმენტების პანელზე არსებული ღილაკებით შეიძლება შესრულდეს შემდეგი ფუნქციები:

- Rotate.
- Move.
- Zoom.
- Depth.
- 3D.
- Edit.
- Save as შაბლონი.

ლაბორატორიული სამუშაო № 13

პროექტები *(Projects)*

სამუშაოს მიზანი:

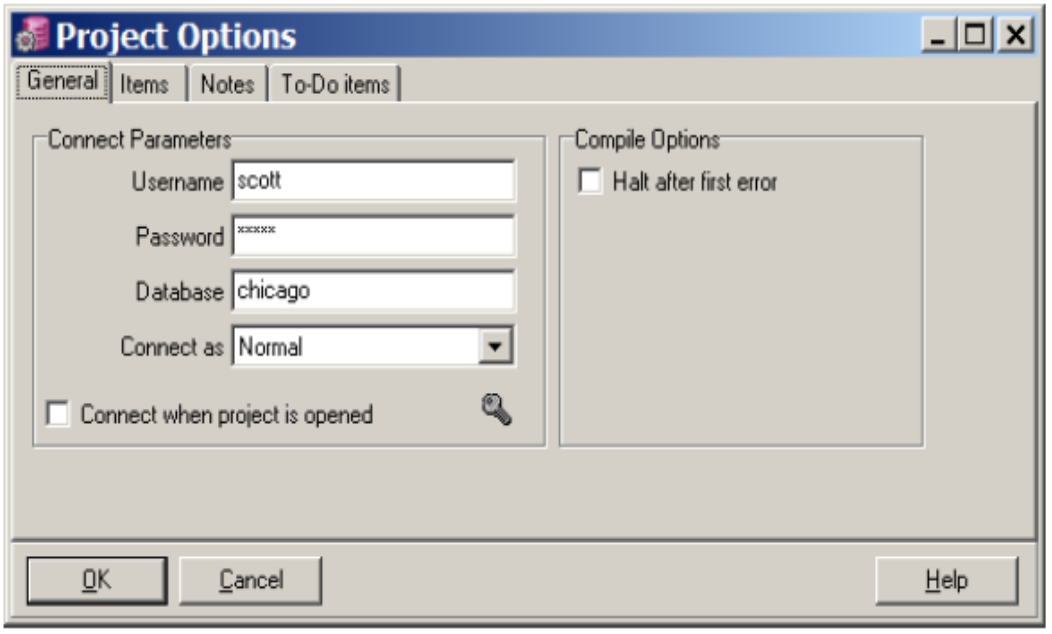
1. მარტივი ანგარიშიანი დანართის შემუშავების შესწავლა;

2. დეტალური ანგარიშიანი დანართის შემუშავების შესწავლა;

3 „მთავარი - დაქვემდებარებული“ ანგარიშიანი დანართის შემუშავების შესწავლა.

PL/SQL Developer-ზე სამუშაოთა ორგანიზებისათვის გამოიყენება პროექტები, რომელიც წარმოადგენს ფაილების, მონაცემთა ბაზების ობიექტების, ოპციების კოლექციას. პროექტი ინარჩუნებს desktop კონფიგურაციას ანუ პროექტის დახურვის შემდეგ მისი ხელმეორედ გახსნისას, იგი იმავე პოზიციაში იხსნება. ეს ხდება *AutoSave Desktop* პრეფერენციის მეშვეობით. ეს ინფორმაცია არ ინახება პროექტის განსაზღვრის ფაილში (*Project*.prj), არამედ desktop ფაილში (*Project*.dsk).

ახალი პროექტის შექმნისათვის *Project* მენიუდან ვირჩევთ *New* იტემს, რომლის შედეგადაც ჩნდება დიალოგური ფანჯარა Project Options:



ნახ. 13.1

ჩანართზე *General* განისაზღვრება, თუ როგორ უნდა მივუერთდეთ, როცა პროექტი გაიხსნება.

ჩანართზე *Items* ჩვენ შეგვიძლია პროექტის იტემების ნახვა. ეს შეიძლება იყოს PL/SQL პროგრამული ნაწილების ფაილები, Test Scripts, SQL Scripts, Reports და ა.შ. პროექტის იტემი აგრეთვე შეიძლება იყოს ობიექტი, რომელიც ინახება მონაცემთა ბაზაში. პროექტის იტემი შეიძლება შეიცავდეს ისეთ ფაილებს, როგორიცაა MS Word დოკუმენტები, HTML დოკუმენტები და ა.შ.

ჩანართი *Notes* პროექტის შენიშვნათა შენახვის საშუალებას იძლევა.

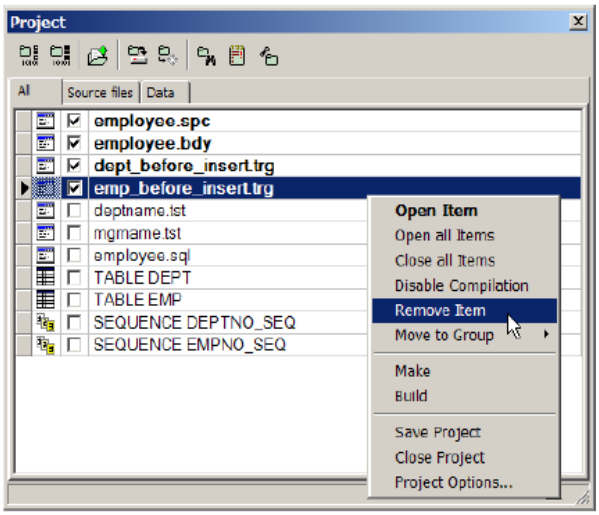
ჩანართი *To-Do*

პროექტის შენახვისათვის *Project* მენიუდან ვირჩევთ *Save* იტემს. შედეგად შეიქმნება პროექტი გაფართოებით .prj.

პროექტში ფაილების დამატებისათვის *Project* მენიუდან ვირჩევთ *Add to Project* იტემს.

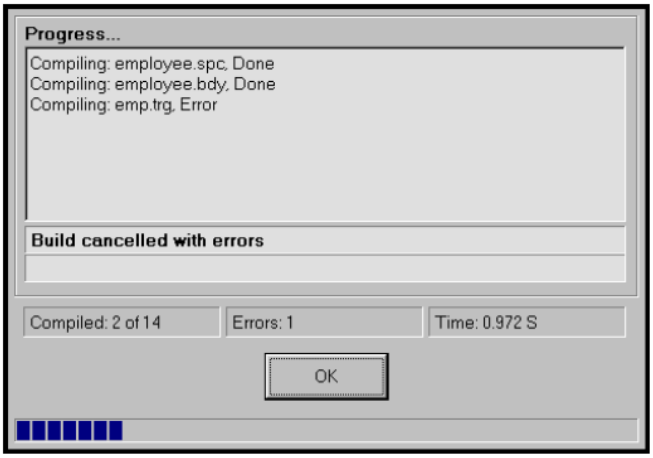
პროექტში მონაცემთა ბაზის ობიექტების დამატებისათვის Object Browser -ზე right-ვაწკაპუნებთ და popup მენიუდან ვირჩევთ *Add to Project*.

პროექტში იტემებთან მუშაობისათვის მათი დამატების შემდეგ *Project* მენიუდან ვირჩევთ *Project Items*:



ნახ. 13.2

პირველი ოთხი პროგრამული ფაილი ჩართულია პროექტის კომპილაციისათვის, რისთვისაც *Project* მენიუდან ვირჩევთ *Build* ან *Make* იტემს. *Build* ფუნქცია ახდენს მიმდინარე შერჩეული ჯგუფის ყველა პროექტის კომპილაციას.

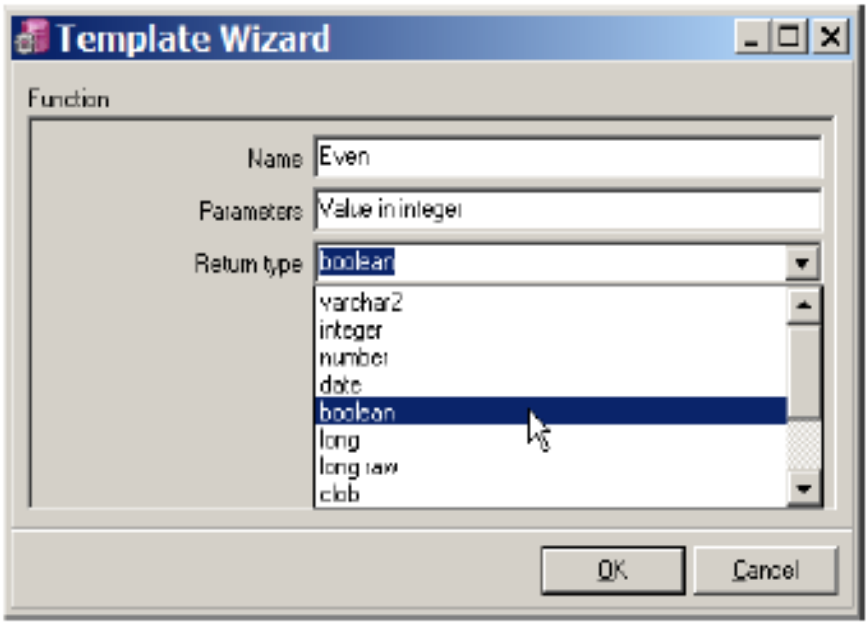


ნახ. 13.3

Make ფუნქცია ახდენს მხოლოდ იმ იტემების კომპილაციას, რომლებიც შეცვლილი იყო წინა კომპილაციის დროს.

პროგრამირება

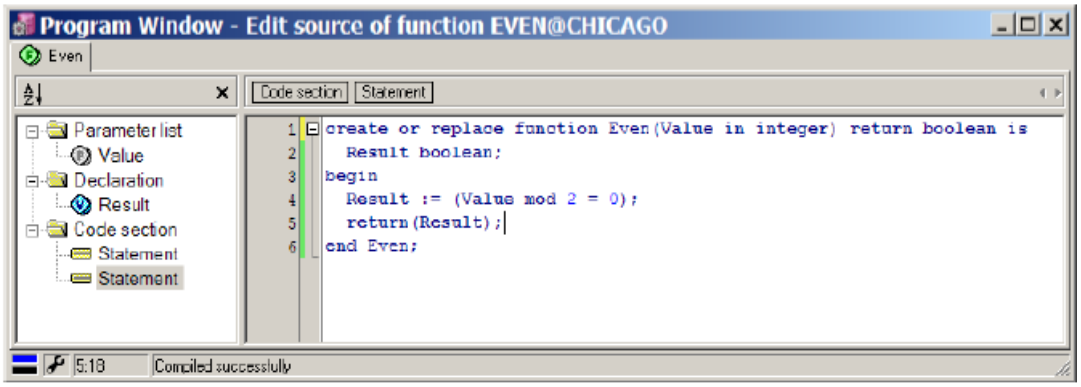
ახალი პროგრამის შექმნისათვის ინსტრუმენტების პანელზე ვაჭერთ *New* ღილაკს, ვირჩევთ *Program Window* იტემს და მაგალითად, *Function* სუბიტემს. ამ შემთხვევაში ჩვენ ვქმნით ფუნქციას, ფუნქციის სახელს, პარამეტრთა სიას და მათი დაბრუნების ტიპს:



ნახ. 13.4

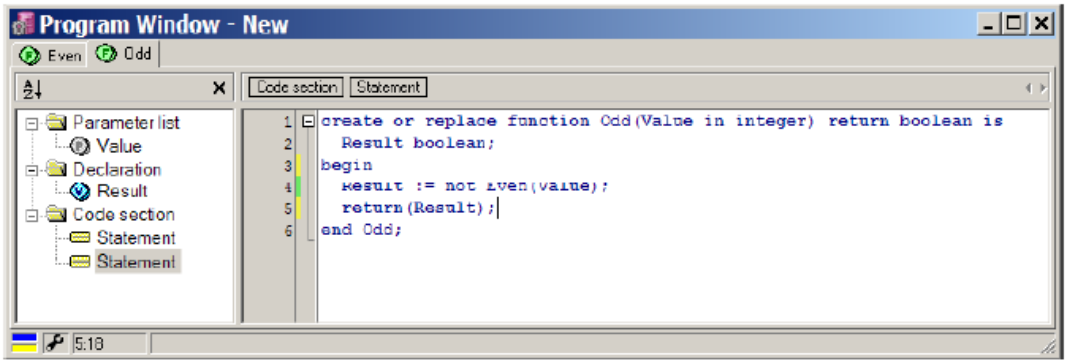
PL/SQL Developer - ს გააჩნია რამდენიმე სტანდარტული შაბლონი, რომელიც შეგვიძლია შევცვალოთ.

ჩვენ შეგვიძლია აგრეთვე განვსაზღვროთ ახალი შაბლონი. ცვლადების შეტანისა და *OK* ღილაკის დაჭერის შემდეგ ჩნდება Program Editor Window ახალ შაბლონი ფუნქციასთან ერთად:



ნახ. 13.5

პროგრამული ერთეული პროგრამულ ფაილში უნდა განთავსდეს ნებისმიერი სხვა პროგრამული ერთეულის შემდეგ რომელთანაც უნდა იქნეს დაკავშირებული. თუ ვქმნით ფუნქციას 'odd' , რომელიც უკავშირდება წინასწარ შექმნილ ფუნქციას 'even', მაშინ პროგრამულ რედაქტორს ექნება შემდეგი სახე:



ნახ. 13.6

პროგრამის შენახვა

პროგრამის შენახვა შეიძლება ღილაკზე *Save* დაჭერით. ქვემოთ მოყვანილია ზოგიერთი სტანდარტული პროგრამული გაფართოება:

Program type	Extension
Function	.fnc
Procedure	.prc
Package specification & body	.pck
Package specification	.spc
Package body	.bdy
Type specification & body	.typ
Type specification	.tps
Type body	.tpb
Trigger	.trg
Java source	.jsp

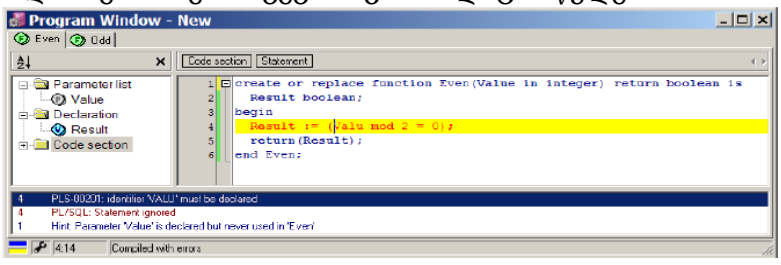
ნახ. 13.7

მაგალითისათვის განვიხილოთ საწყისი ფაილი, რომელიც შეიცავს 'odd' და 'even' ფუნქციებს. იგი შეიცავს ფუნქცია Even(integer), რომელიც აბრუნებს შედეგს ბულის სახით:

```
begin
Result := (Value mod 2 = 0);
return(Result);
end Even;
/
create or replace function Odd(Value in integer) return boolean is
Result boolean;
begin
Result := not Even(Value);
return(Result);
end Odd;
/
```

პროგრამის მოდიფიცირება შესაძლებელია ინსტრუმენტების პანელზე ღილაკი *Open* დაჭერით და *Program file* იტემის შერჩევით.

პროგრამის კომპილაცია შეიძლება ღილაკზე *Execute* დაჭერით. პროგრამული ფაილის ყველა პროგრამული ერთეული კომპილირდება. შეცდომის აღმოჩენის შემთხვევაში კომპილაცია წყდება:



ნახ. 13.8

ლაბორატორიული სამუშაო 14

Oracle 10g XE ადმინისტრირება

სამუშაოს მიზანი:

1.მონაცემთა ბაზების ადმინისტრირების საკითხები;

2.მონაცემთა იმპორტი და ექსპორტი;

3.სარეზერვო ასლის შექმნა და აღდგენა.

1. მონაცემთა ბაზების ადმინისტრირების საკითხები

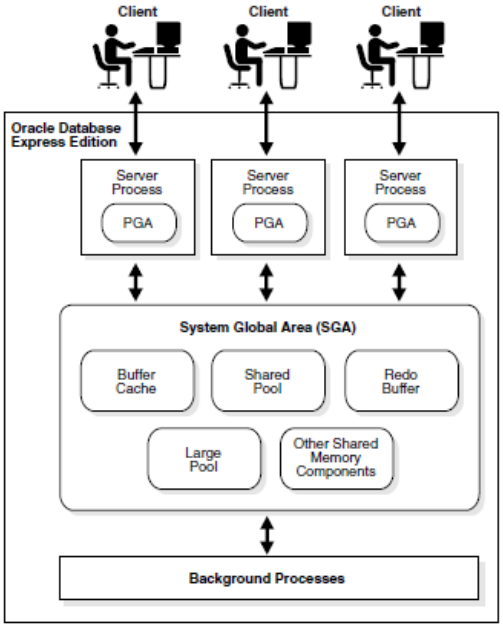
მონაცემთა ბაზის მეხსიერების მართვა

მონაცემთა ბაზის მხარდასაჭერად Oracle XE წარმართავს ისეთ ფონურ (Background) პროცესებს, რომლებიც მეხსიერების განაწილებას უზრუნველყოფს (Oracle instance). განირჩევა ორი ტიპის მეხსიერება:

–სისტემური გლობალური არე (System global area - SGA).

–პროგრამული გლობალური არე (Program global area – PGA).

Oracle-ის სერვერული პროცესები კლიენტის მოთხოვნებს ემსახურება. ამასთან ყოველი ახალი სერვერული პროცესისათვის გამოიყოფა მისი საკუთარი PGA არე.

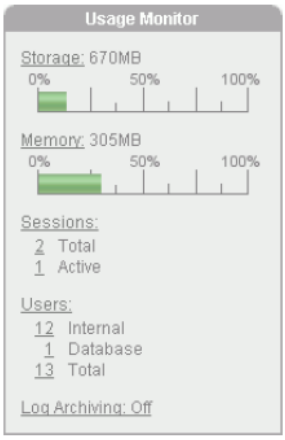


ნახ. 14.1

მეხსიერების გამოყენების მიმდინარე მდგომარეობის ნახვისათვის:

1. მივმართავთ Database Home გვერდს.

2. მონიტორის მარჯვენა ნაწილში ჩნდება Usage Monitor ინფორმაციული ფანჯარა.



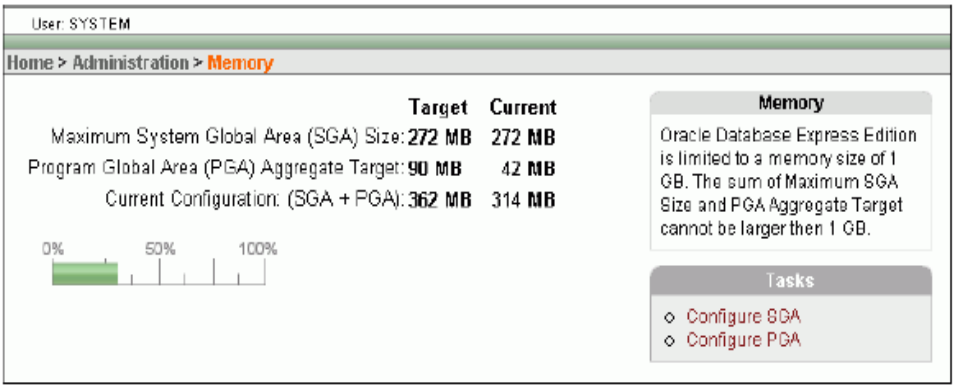
ნახ. 14.2

SGA და PGA მაჩვენებლების ცვლილებისათვის:

1. მივმართავთ Database Home გვერდს.

2. ვაწკაპუნებთ იკონაზე **Administration** და შემდეგ ვაწკაპუნებთ იკონაზე **Memory**.

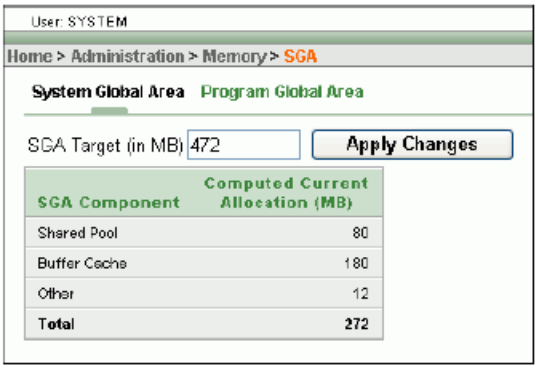
ადმინისტრატორისათვის საჭიროა SYSTEM მომხმარებლის სახელის და პაროლის შეტანა. შემდეგ ვაწკაპუნებთ **Login**. ჩნდება გვერდი Memory.



ნახ. 14.3

3. ამოცანათა სიაში ვაწკაპუნებთ **Conბახ. SGA**.

4. SGA გვერდის SGA Target (MB -ში) ველში შეგვაქვს **472**, რომლის მნიშვნელობა წარმოადგენს ჯამს მიმდინარე SGA ზომისა (272) და 200, რისი დამატებაც გვინდა.



ნახ. 14.4

5. ვაწკაპუნებთ **Apply Changes**. ჩნდება დადასტურების შეტყობინება.

6. ვაწკაპუნებთ ლინკზე **Program Global Area** PGA გვერდთან დასაკავშირებლად.

7. PGA Aggregate Target ველში შეგვაქვს **140** და შემდეგ ვაწკაპუნებთ **Apply Changes**. მნიშვნელობა 140 არის ჯამი მიმდინარე PGA Aggregate Target ზომისა (90) და 50.

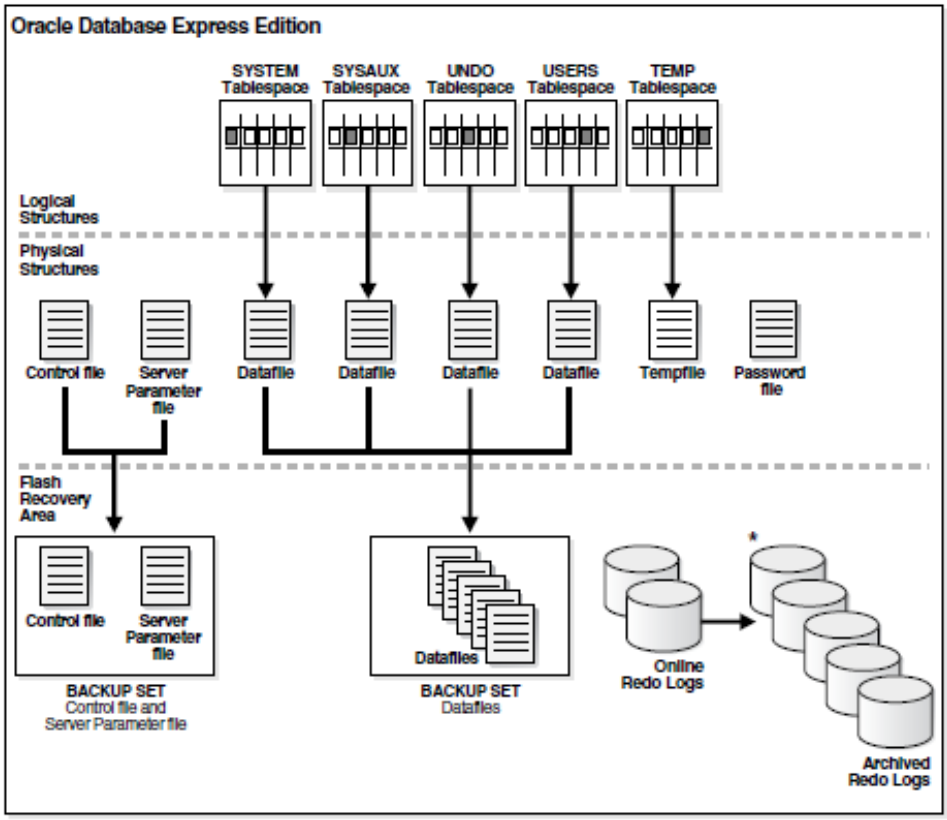
8. SGA ზომის ცვლილების დასაფიქსირებლად გადავტვირთოთ კომპიუტერი.

მონაცემთა ბაზის შენახვის მართვა

Oracle Database XE შედგება შემდეგი შენახვის სტრუქტურებისაგან:

- Logical structures.
- Physical structures.
- Recovery-related structures.

.



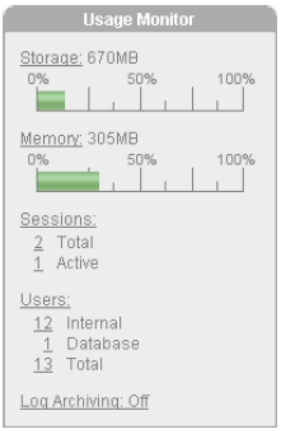
ნახ. 14.5

მონაცემთა ბაზა შეიცავს ერთ ან რამდენიმე ცხრილურ არეს (ცხრილი space).

შენახვის არეს გამოყენების შესახებ ინფორმაციის ნახვისათვის:

1. მივმართავთ Database Home გვერდს.

2. Usage Monitor -ის მარჯვენა მხარეს ვნახულობთ ინფორმაციას **Storage**.



ნახ. 14.6

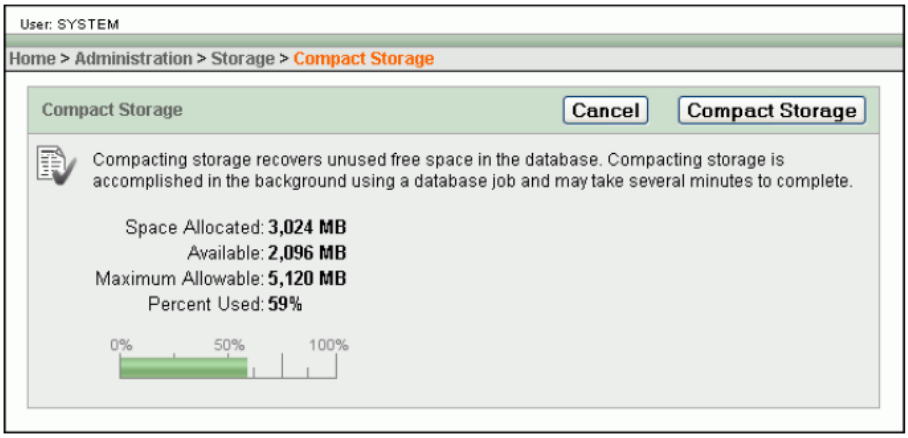
შენახვის მოცულობის შემცირების მიზნით:

1. მივმართავთ Database Home გვერდს.

2. ვაწკაპუნებთ იკონაზე **Administration** და შემდეგ ვაწკაპუნებთ იკონაზე **Storage**.

ადმინისტრატორისათვის საჭიროა SYSTEM მომხმარებლის სახელის და პაროლის შეტანა. შემდეგ ვაწკაპუნებთ **Login**. ჩნდება გვერდი Storage.

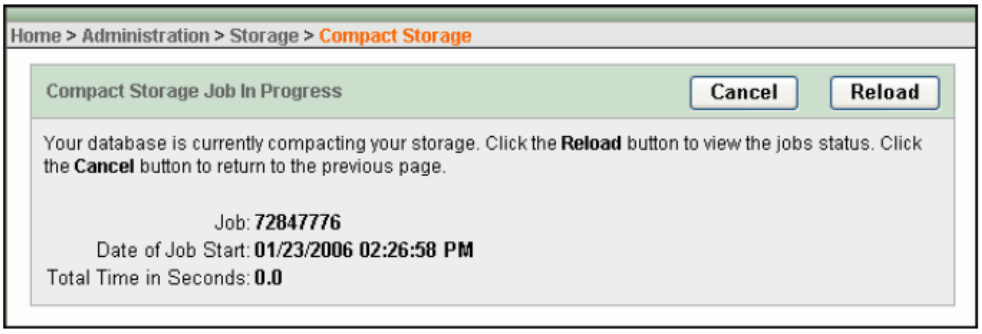
3. ვაწკაპუნებთ **Compact Storage**. ჩნდება გვერდი Compact Storage.



ნახ. 14.7

4. ვაწკაპუნებთ **Compact Storage**. ხელმეორედ ჩნდება გვერდი Storage.

5. Compact Storage გვერდში დაბრუნებისა და დავალების (job) სტატუსის ნახვისათვის ვაწკაპუნებთ **Compact Storage**.



ნახ. 14.8

დავალების სტატუსის განახლებისათვის ვაწკაპუნებთ ღილაკზე **Reload.**

ცხრილური არეს ნახვა

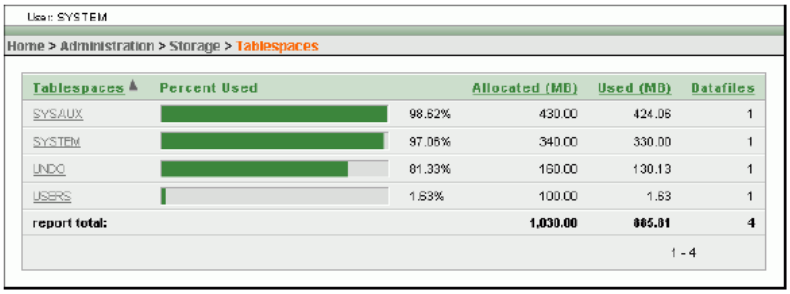
Oracle Database XE ცხრილური არეს ნახვისათვის:

1. მივმართავთ Database Home გვერდს.

2. ვაწკაპუნებთ იკონაზე **Administration** და შემდეგ ვაწკაპუნებთ იკონაზე **Storage**.

ადმინისტრატორისათვის საჭიროა SYSTEM მომხმარებლის სახელის და პაროლის შეტანა. შემდეგ ვაწკაპუნებთ **Login**. ჩნდება გვერდი Storage.

3. ვაწკაპუნებთ წარმოდგენა ცხრილი spaces. ჩნდება გვერდი ცხრილი spaces.



ნახ. 14.9

4. ცხრილური არეს ნახვისათვის ვაწკაპუნებთ ცხრილური არეს სახელზე.

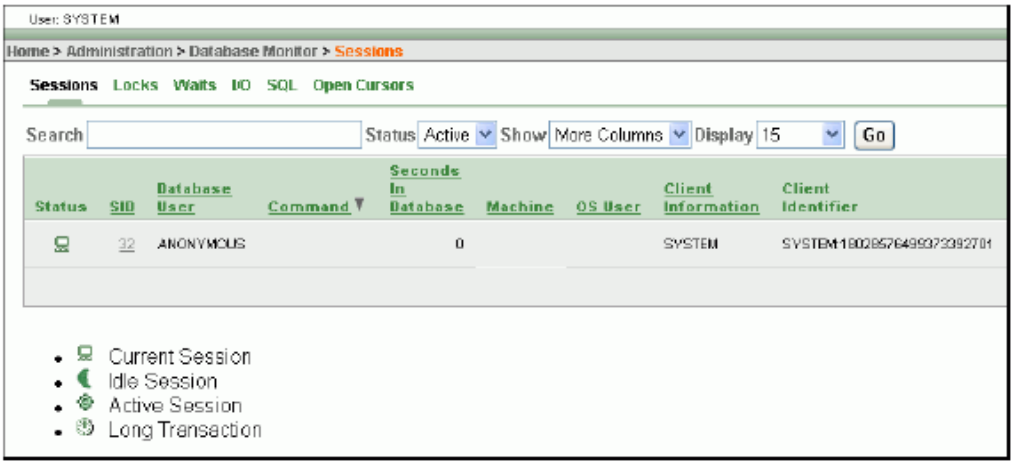
მონაცემთა ბაზის მონიტორინგი

სესიების ნახვა

1. მივმართავთ Database Home გვერდს. ვარეგისტრირებთ SYSTEM მომხმარებელს.

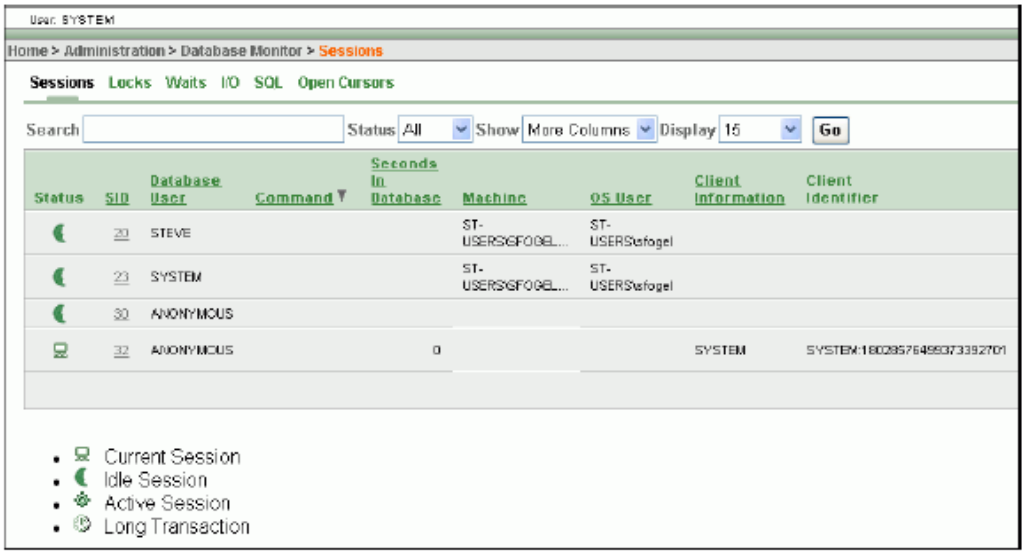
2. Database Home გვერდზე ვაწკაპუნებთ **Administration** და შემდეგ ვაწკაპუნებთ **Monitor**.

3. Database Monitor გვერდზე ვაწკაპუნებთ **Sessions**. ჩნდება გვერდი Sessions, რომელიც ასახავს მიმდინარე აქტიურ სესიებს.



ნახ. 14.10

4. Status სიაში ვირჩევთ **All** და შემდეგ ვაწკაპუნებთ **Go**. გვერდზე აისახება ყველა სესია.



ნახ. 14.11

5. სესიების სიის შემცირების მიზნით საძიებელ ველში **Search** შეგვაქვს ტექსტი და ვაწკაპუნებთ **Go**.

6. საძიებელ ველის Search ზემოთ ვაწკაპუნებთ ნებისმიერ ჰიპერლინკზე ყველა სესიისათვის შემდეგი ინფორმაციის ნახვისათვის: Locks, Waits, Input/Output (I/O), SQL ბრძანებების შესრულება და კურსორების გახსნა.

7. SID სვეტის ქვემოთ ვაწკაპუნებთ სესიის ID -ზე Session Details გვერდის ნახვისათვის. Session Details გვერდი სესიის დასრულების საშუალებას იძლევა.

სესიის დასრულება (kill):

1. ვნახულობთ ყველა სესიას.

2. SID სვეტის ქვემოთ ვაწკაპუნებთ სესიაზე, რომლის დასრულებაც გვინდა. ჩნდება Session Details გვერდი.

3. ვაწკაპუნებთ **Kill Session**. ჩნდება გვერდი, რომელიც მიგვითითებს მოქმედების შემოწმებაზე.

4. განმეორებით ვაწკაპუნებთ **Kill Session**.

სისტემის სტატისტიკის მონიტორინგი

სისტემის სტატისტიკის ნახვისათვის:

1. მივმართავთ Database Home გვერდს. ვარეგისტრირებთ SYSTEM მომხმარებელს.

2. Database Home გვერდზე ვაწკაპუნებთ **Administration** და შემდეგ ვაწკაპუნებთ **Monitor**.

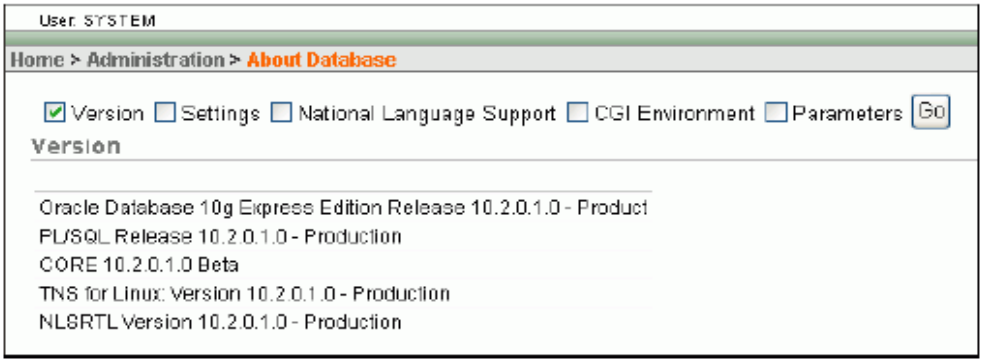
3. Database Monitor გვერდზე ვაწკაპუნებთ **System Statistics**. ჩნდება შესაბამისი გვერდი System Statistics.

მონაცემთა ბაზის ვერსიის ნახვა და დაყენება

მონაცემთა ბაზის ვერსიის ნახვისათვის:

1. მივმართავთ Database Home გვერდს. ვარეგისტრირებთ SYSTEM მომხმარებელს.

2. Database Home გვერდზე ვაწკაპუნებთ **Administration** და შემდეგ ვაწკაპუნებთ **About Database**. ჩნდება გვერდი About Database, რომელიც გვიჩვენებს ინფორმაციას მონაცემთა ბაზის ვერსიის შესახებ.

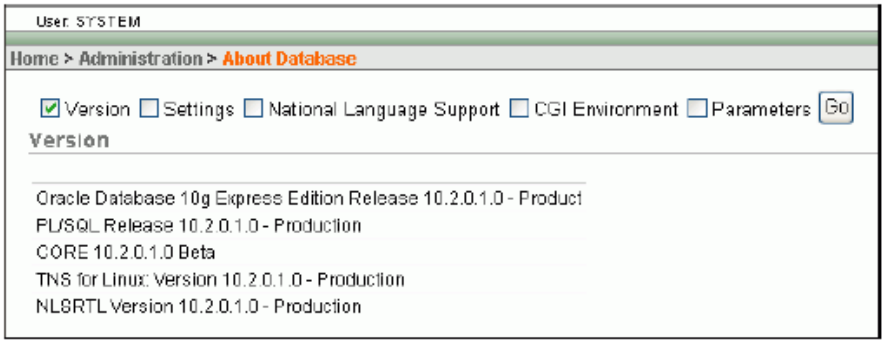


ნახ. 14.12

მონაცემთა ბაზის დაყენების ნახვა:

1. მივმართავთ Database Home გვერდს. ვარეგისტრირებთ SYSTEM მომხმარებელს.

2. Database Home გვერდზე ვაწკაპუნებთ **Administration** და შემდეგ ვაწკაპუნებთ **About Database**. ჩნდება გვერდი About Database.



ნახ. 14.13

3. ვაყენებთ (ან ვაუქმებთ) დროშას გვერდის ზედა ნაწილში და შემდეგ ვაწკაპუნებთ **Go**.

ვერსიის თაობაზე მონაცემების დამატებისათვის შეგვიძლია ეკრანზე გამოვიტანოთ შემდეგი ინფორმაცია:

– **Settings**.

– **National Language Support**.

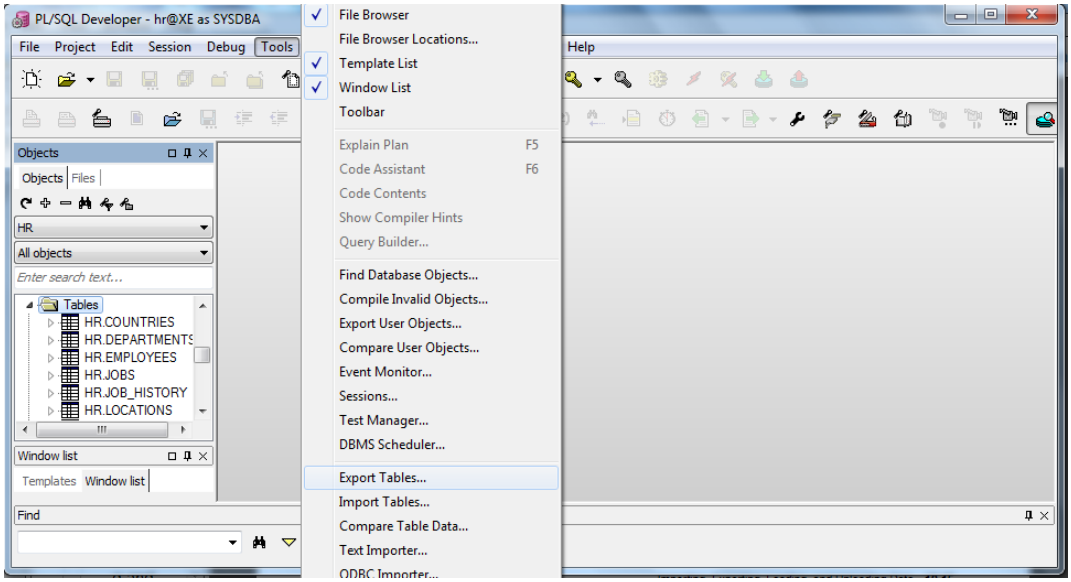
– **CGI Environment**.

– **Parameters**.

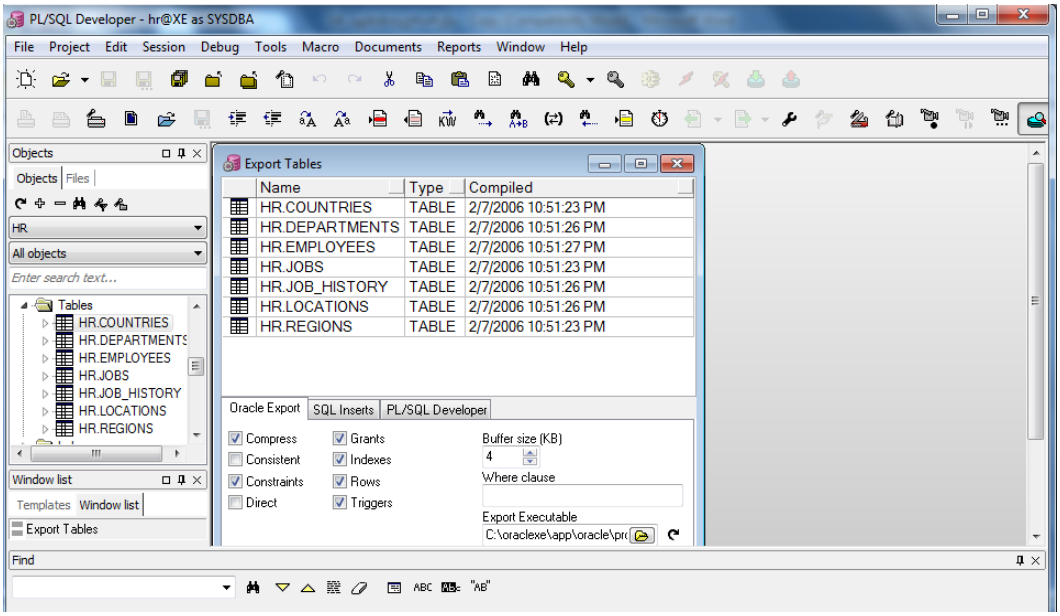
ინიციალიზაციის პარამეტრების ცვლილების შეტანა შესაძლებელია SQL Command Line -ში ALTER SYSTEM ბრძანებების მეშვეობით.

2.მონაცემთა იმპორტი და ექსპორტი

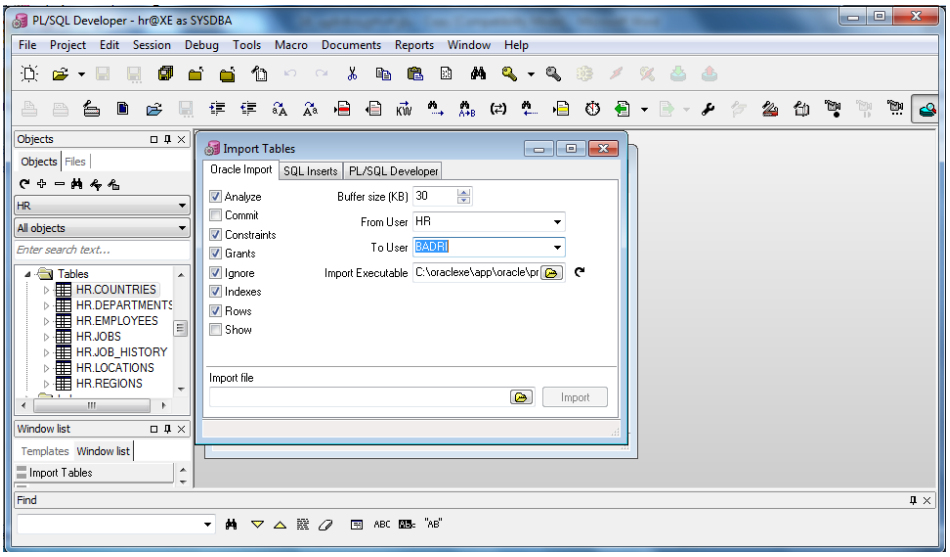
მონაცემთა იმპორტი და ექსპორტი ხორციელდებაშემდეგი ეკრანების მიხედვით:



ნახ. 14.14



ნახ. 14.15



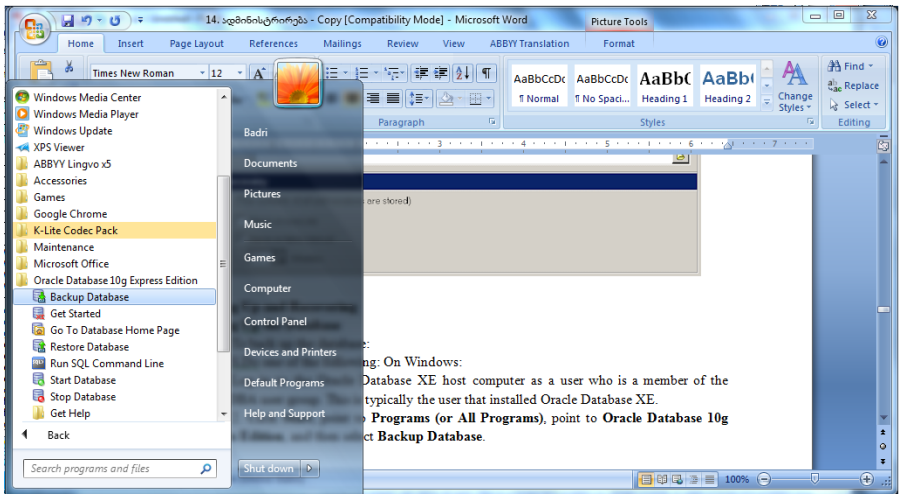
ნახ. 14.16

3.სარეზერვო ასლის შექმნა და აღდგენა

მონაცემთა ბაზის სარეზერვო ასლის შექმნისათვის:

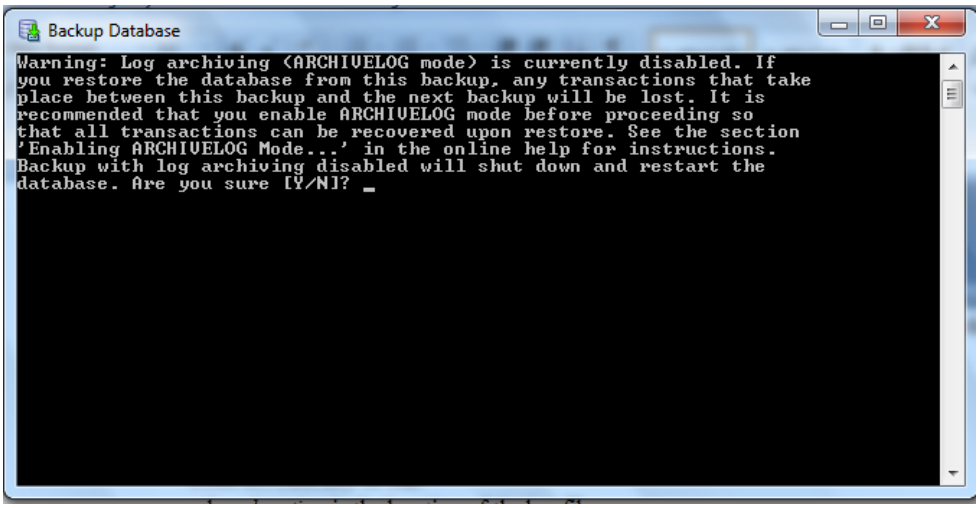
1. დავარეგისტრიროთ Oracle Database XE მთავარი (host) კომპიუტერი, როგორც მომხარებელი, რომელიც არის მომხმარებელთა ჯგუფის ORA_DBA წევრი.

2. თანამიმდევრობით ვაწკაპუნებთ **Start**, **Programs (or All Programs)**, **Oracle Database 10g Ex** ვაჭერთ **Edition** და შემდეგ ვირჩევთ **Backup Database**.



ნახ. 14.17

3. მონაცემთა ბაზის დახურვისა და სარეზერვო ასლის შექმნის მიზნით შეგვაქვს **y** და ვაჭერთ **Enter**.



ნახ. 14.18

სარეზერვო ასლის შექმნის დასრულების შემდეგ ეკრანზე ჩნდება ტექსტი:

Backup of the database succeeded.
Log file is at *location*
გამოსვლისათვის ვაჭერთ ENTER

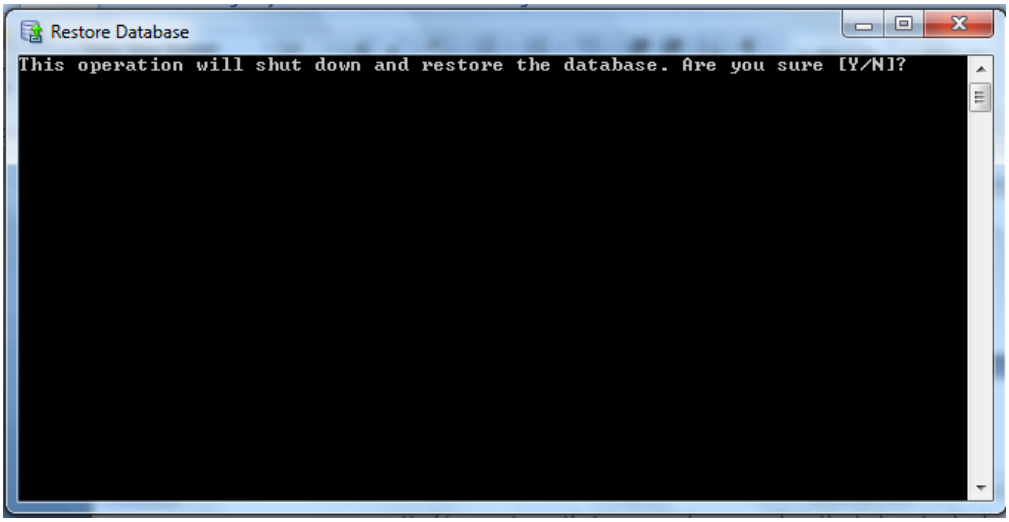
სადაც: *location* - არის სისტემური ჟურნალის ფაილის (log file) ადგილმდებარეობა.

5. სარეზერვო ასლის შექმნის ფანჯრის დახურვისათვის ვაჭერთ **Enter**.

მონაცემთა ბაზის აღდგენისათვის:

1. თანამიმდევრობით ვაწკაპუნებთ Start, Programs (or All Programs), Oracle Database 10g Ex, ვაჭერთ Edition და შემდეგ ვირჩევთ Restore Database.

2. იხსნება კონსოლის ფანჯარა, სადაც წარმოდგენილია შეკითხვა აღდგენის თაობაზე.



ნახ. 14.19

3. აღდგენის დადასტურებისათვის შეგვაქვს **y** და ვაჭერთ **Enter**. შესაბამისად, აღდგენისათვის მონაცემთა ბაზა იხურება და სრულდება სკრიპტი RMAN

4. შედეგად ეკრანზე ჩნდება შემდეგი შეტყობინება:

Restore of the database succeeded.
Log file is at *location*
გამოსვლისათვის ვაჭერთ ENTER

სადაც: *location* - არის სისტემური ჟურნალის ფაილის (log file) ადგილმდებარეობა.

5. ვაჭერთ **Enter** to close the Restore Database window.

წაშლილი ცხრილების აღდგენა

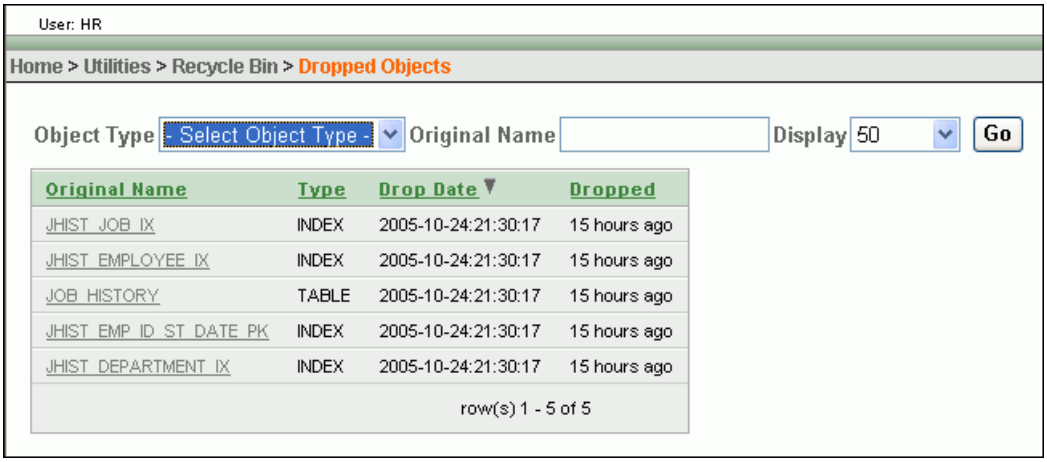
„სანაგვე კალათის“ (recycle bin) შემადგენლობის ნახვისათვის:

1. მივმართოთ Database Home გვერდს და დავრეგისტრირდეთ როგორც სქემის მესაკუთრე (owner).

2. ვაწკაპუნებთ იკონაზე **Utilities** და შემდეგ ვაწკაპუნებთ იკონაზე **Recycle Bin**.

3. ვაწკაპუნებთ იკონაზე **Dropped Objects**.

ჩნდება Dropped Objects გვერდი, სადაც ნაჩვენებია წაშლილი ობიექტების სია.



ნახ. 14.20

4. სიის შემცირების მიზნით საძიებელ ველში **Object Type** ვირჩევთ ტიპს და ვაწკაპუნებთ **Go**.

5. საძიებელ ველში **Original Name** შეგვაქვს ტექსტი და ვაწკაპუნებთ **Go**.

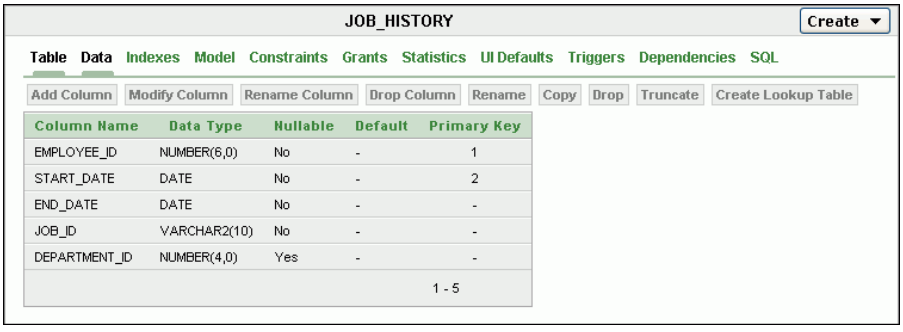
6. დეტალური ინფორმაციის ნახვისათვის ვაწკაპუნებთ ობიექტის ლინკზე.

მაგალითი: ცხრილის აღდგენა Recycle Bin - დან

1. მივმართოთ Database Home გვერდს და დავრეგისტრირდეთ როგორც HR მომხმარებელი.

2. ვაწკაპუნებთ იკონაზე **Object Browser**. ჩნდება Object Browser გვერდი, სადაც წარმოდგენილია HR სქემა.

3. ვაწკაპუნებთ ცხრილზე **JOB_HISTORY**. დეტალური ინფორმაცია ცხრილის შესახებ ჩნდება გვერდის მარჯვენა მხარეს.

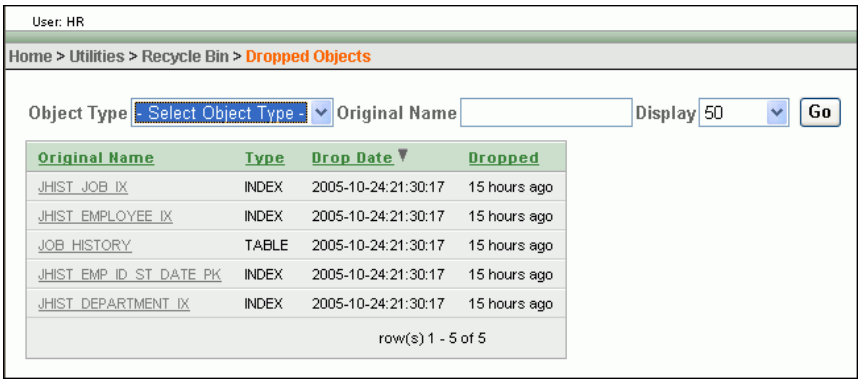


ნახ. 14.21

4. ვაწკაპუნებთ ღილაკზე **Drop**. ჩნდება Drop Object Confirmation ფანჯარა.
5. ცხრილის წაშლისათვის ვაწკაპუნებთ **Finish**.

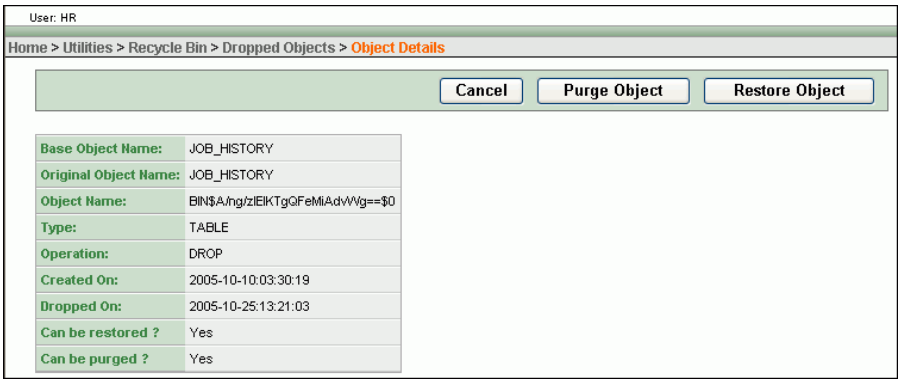
შედეგად ცხრილი ამოიღება ცხრილების სიიდან გვერდის მარცხენა მხარეს.

6. Database Home გვერდზე დაბრუნებისათვის ვაწკაპუნებთ **Home**.
7. ვაწკაპუნებთ იკონაზე **Utilities** და შემდეგ ვაწკაპუნებთ იკონაზე **Recycle Bin**.
8. ვაწკაპუნებთ იკონაზე **Dropped Objects**. ჩნდება წაშლილი ობიექტები.



ნახ. 14.22

9. Original Name სვეტის ქვემოთ ვაწკაპუნებთ ცხრილზე **JOB_HISTORY**.
10. ჩნდება გვერდი Object Details, სადაც წარმოდგენილია ინფორმაცია `JOB_HISTORY` ცხრილის შესახებ.



ნახ. 14.23

11. ვაწკაპუნებთ **Restore Object**. ჩნდება დადასტურების გვერდი. ცხრილი და მასზე დამოკიდებული ყველა ობიექტი აღდგენილი იქნება.

Recycle Bin -ის გასუფთავება

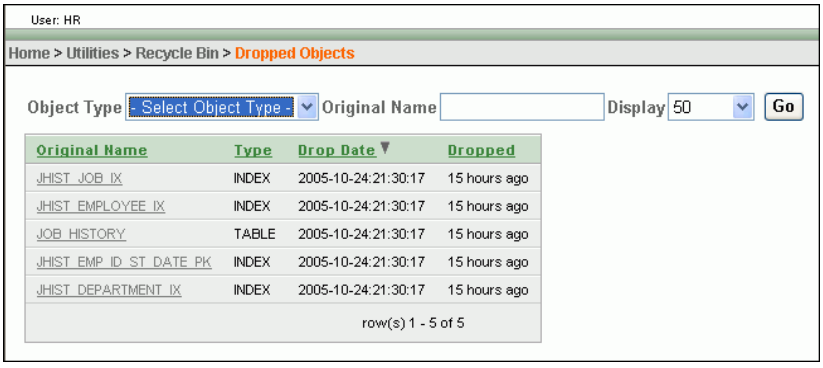
„სანაგვე კალათის“ (recycle bin) სრული შემადგენლობის გასუფთავებისათვის:

1. მივმართოთ Database Home გვერდს და დავრეგისტრირდეთ როგორც სქემის მესაკუთრე.
2. ვაწკაპუნებთ იკონაზე **Utilities** და შემდეგ ვაწკაპუნებთ იკონაზე **Recycle Bin**.
3. ვაწკაპუნებთ იკონაზე **Purge Recycle Bin**. ჩნდება ოპერაციის დადასტურების გვერდი.
4. განმეორებით ვაწკაპუნებთ იკონაზე **Purge Recycle Bin**. ჩნდება დადასტურების გვერდი.

ინდივიდუალური ობიექტის გასუფთავება Recycle Bin -დან

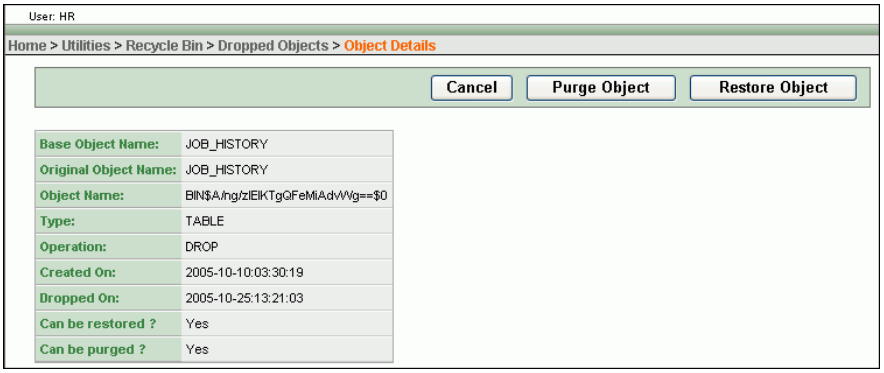
ინდივიდუალური ობიექტის გასუფთავებისათვის:

1. მივმართოთ Database Home გვერდს და დავრეგისტრირდეთ როგორც სქემის მესაკუთრე.
2. ვაწკაპუნებთ იკონაზე **Utilities** და შემდეგ ვაწკაპუნებთ იკონაზე **Recycle Bin**.
3. ვაწკაპუნებთ იკონაზე **Dropped Objects**. ჩნდება Dropped Objects გვერდი, სადაც წარმოდგენილია წაშლილი ობიექტების სია.



ნახ. 14.24

4. Original Name სვეტის ქვემოთ ვაწკაპუნებთ ობიექტის სახელზე, რომლის გასუფთავებაც გვინდა. ჩნდება Object Details გვერდი, რომელზეც აისახება ინფორმაცია ობიექტის შესახებ.



ნახ. 14.25

5. ვაწკაპუნებთ **Purge Object**. ჩნდება დადასტურების გვერდი. ობიექტი და მასზე დამოკიდებული ყველა ობიექტი იქნება გასუფთავებული.

ლაბორატორიული სამუშაო 15

მონაცემთა ბაზების უსაფრთხოება

სამუშაოს მიზანი:

1.მონაცემთა ბაზების უსაფრთხოების სისტემის გაცნობა;

2.მომხმარებლები;

3.სისტემური პრივილეგიები;

4.როლები.

1. მონაცემთა ბაზების უსაფრთხოების სისტემა

მომხმარებლის სააღრიცხვო ჩანაწერი (user account) გაიგივებულია მომხმარებლის სახელთან და განისაზღვრება შემდეგი ატრიბუტებით:

- პაროლი მონაცემთა ბაზის აუთენტიფიკაციისათვის;
- პრივილეგიები და როლები;
- ცხრილური არე (ცხრილი space) მონაცემთა ბაზის ობიექტე-ბისათვის;
- დროებითი ცხრილური არე მოთხოვნათა დამუშავებისათვის.

როდესაც ვქმნით მომხმარებელს, ჩვენ არაცხადი სახით ვქმნით აგრეთვე სქემას ამ მომხმარებლისათვის. სქემა (schema) წარმოაადგენს ლოგიკურ კონტეინერს მონაცემთა ბაზის ისეთი ობიექტებისათვის (როგორიცაა ცხრილები, წარმოდგენები, ტრიგერები და ა.შ.), რომელთაც მომხმარებელი ქმნის. სქემას გააჩნია იგივე სახელი, რაც თვითონ მომხმარებელს. სქემის მეშვეობით ცალსახად შეიძლება მომხმარებლის კუთვნილებაში არსებული ყველა ობიექტებისადმი მიმართვა.

2.მომხმარებლები

მომხმარებლის შექმნა

მომხმარებლის შესაქმნელად გამოიყენება ბრძანება CREATE USER:

CREATE USER *user_name* IDENTIFIED BY *password*
[DEFAULT ცხრილიSPACE *def_tabspace*]
[TEMPORARY ცხრილიSPACE *temp_tabspace*];

სადაც:

- *user_name* ადგენს მომხმარებლის სახელს.

- *password* ადგენს მომხმარებლის პაროლს.
- *def_tabspace* ადგენს ცხრილის არეს, სადაც ობიექტები ინახება. თუ თქვენ გვერდს აუვლით მის განსაზღვრას, მაშინ ავტომატურად SYSTEM ცხრილის არე იქნება გამოყენებული..
- *temp_tabspace* განსაზღვრავს ცხრილის არეს, სადაც დროებითი ობიექტები ინახება. თუ თქვენ გვერდს აუვლით მის განსაზღვრას, მაშინ ავტომატურად SYSTEM ცხრილის არე იქნება გამოყენებული..

მაგალითი:

```
CONNECT system/manager
CREATE USER jason IDENTIFIED BY price;
```

მაგალითი:

```
CREATE USER henry IDENTIFIED BY hooray
DEFAULT ცხრილიSPACE users
TEMPORARY ცხრილიSPACE temp;
```

მონაცემთა ბაზაში ნებისმიერი ქმედებისათვის აუცილებელია შესაბამისი ნებართვების მინიჭება, რითვისაც გამოიყენება ბრძანება GRANT.

მაგალითი:

```
GRANT CREATE SESSION TO jason;
```

მაგალითი:

```
CREATE USER steve IDENTIFIED BY button;
CREATE USER gail IDENTIFIED BY seymour;
GRANT CREATE SESSION TO steve, gail;
```

მომხმარებლის პაროლის შეცვლა

მომხმარებლის პაროლის შესაცვლელად გამოიყენება ბრძანება ALTER USER.

მაგალითი:

```
ALTER USER jason IDENTIFIED BY marcus;
```

მომხმარებლის ამოგდება

მომხმარებლის ამოგდებისათვის გამოიყენება ბრძანება DROP USER.

მაგალითი:

CONNECT system/manager
DROP USER jason;

3.სისტემური პრივილეგიები

სისტემური პრივილეგიები (*system privilege)* მომხმარებელს ნებას აძლევს შეასრულოს ზოგიერთი ქმედება მონაცემთა ბაზასთან DDL ბრძანებების გამოყენებით.

System Privilege	**Allows You to …**
CREATE SESSION	მონაცემთა ბაზასთან შეერთება.
CREATE SEQUENCE	მიმდევრობის შექმნა.
CREATE SYNONYM	სინონიმის შექმნა.
CREATE ცხრილი	ცხრილის შექმნა.
CREATE ANY ცხრილი	ცხრილის შექმნა ნებისმიერ სქემაში.
DROP ცხრილი	ცხრილის წაშლა.
DROP ANY ცხრილი	ცხრილის წაშლა ნებისმიერ სქემაში.
CREATE PROCEDURE	შენახული პროცედურის შექმნა.
EXECUTE ANY PROCEDURE	პროცედურის შესრულება ნებისმიერ სქემაში.
CREATE USER	მომხმარებლის შექმნა.
DROP USER	მომხმარებლის წაშლა.
CREATE წარმოდგენა	წარმოდგენის შექმნა.

მომხმარებლისათვის სისტემური პრივილეგიების მინიჭება

მომხმარებლისათვის სისტემური პრივილეგიების მინიჭებისათვის გამოიყენება ბრძანება GRANT.

მაგალითი:

GRANT CREATE SESSION, CREATE USER, CREATE ცხრილი TO steve;

მაგალითი:

GRANT EXECUTE ANY PROCEDURE TO steve WITH ADMIN OPTION;

მაგალითი:

CONNECT steve/button
GRANT EXECUTE ANY PROCEDURE TO gail;

მაგალითი:

CONNECT system/manager
GRANT EXECUTE ANY PROCEDURE TO PUBLIC;

მომხმარებლისათვის სისტემური პრივილეგიების შემოწმება

თქვენ შეგიძლიათ სისტემური პრივილეგიების შემოწმება user_sys_privs მოთხოვნის მეშვეობით, რომელიც აისახება შემდეგ ცხრილში.

Column	Type	Description
username	VARCHAR2(30)	მიმდ. მომხმარებ. სახელი.
privilege	VARCHAR2(40)	სისტემური პრივილეგია.

მაგალითი:

```
CONNECT steve/button
SELECT *
FROM user_sys_privs;
```

```
USERNAME            PRIVILEGE                      ADM
------------------- ------------------------------  ---
PUBLIC               EXECUTE ANY PROCEDURE          NO
STEVE               CREATE SESSION                  NO
STEVE               CREATE ცხრილი                   NO
STEVE               CREATE USER                     NO
STEVE               EXECUTE ANY PROCEDURE           YES
```

მაგალითი:

```
CONNECT gail/seymour
SELECT *
FROM user_sys_privs;
```

USERNAME	PRIVILEGE	ADM
GAIL	CREATE SESSION	NO
GAIL	EXECUTE ANY PROCEDURE	NO
PUBLIC	EXECUTE ANY PROCEDURE	NO

სისტემური პრივილეგიების გაუქმება

სისტემური პრივილეგიების გაუქმებისათვის გამოიყენება ბრძანება REVOKE.

მაგალითი:

```
CONNECT system/manager
REVOKE CREATE ცხრილი FROM steve;
```

მაგალითი:

```
REVOKE EXECUTE ANY PROCEDURE FROM steve;
```

მაგალითი:

```
CONNECT gail/seymour
SELECT *
FROM user_sys_privs;
```

USERNAME	PRIVILEGE	ADM
GAIL	CREATE SESSION	NO
GAIL	EXECUTE ANY PROCEDURE	NO
PUBLIC	EXECUTE ANY PROCEDURE	NO

ობიექტური პრივილეგიები

ობიექტური პრივილეგიები (*object privilege)* მომხმარებელს ნებას აძლევს შეასრულოს გარკვეული ქმედებები მონაცემთა ბაზის ობიექტებზე, როგორიცაა DML ბრძანებების შესრულება.

Object Privilege	**Allows a User to …**
SELECT	select-ის შესრულება
INSERT	insert-ის შესრულება
UPDATE	update-ის შესრულება
DELETE	delete-ის შესრულება

Object Privilege	Allows a User to …
EXECUTE	შენახული პროცედურის შესრულება

მომხმარებლისათვის ობიექტური პრივილეგიების მინიჭება

მაგალითი:

```
CONNECT store/store_password
GRANT SELECT, INSERT, UPDATE ON store.products TO steve;
GRANT SELECT ON store.employees TO steve;
```

მაგალითი:

```
GRANT UPDATE (last_name, salary) ON store.employees TO steve;
```

მაგალითი:

```
GRANT SELECT ON store.customers TO steve WITH GRANT
OPTION;
```

მაგალითი:

```
CONNECT steve/button
GRANT SELECT ON store.customers TO gail;
```

ობიექტური პრივილეგიების შემოწმება

Column	Type	Description
grantee	VARCHAR2(30)	პრივილეგიის მიმღები მომხმარებელი.
table_name	VARCHAR2(30)	ცხრილის სახელი, რომელსაც პრივილეგია მიენიჭა.
grantor	VARCHAR2(30)	პრივილეგიის მიმნიჭებელი მომხმარებელი.
privilege	VARCHAR2(40)	პრივილეგია ობიექტზე.
grantable	VARCHAR2(3)	ერთერთი, ვისაც სხვისთვის პრივილეგიის მინიჭება შეუძლია. YES ან NO.
hierarchy	VARCHAR2(3)	ერთერთი პრივილეგია, რომელიც იერარქიის ნაწილს აყალიბებს. YES ან NO.

მაგალითი:

```
CONNECT store/store_password
SELECT *
FROM user_tab_privs_made
WHERE ცხრილი_name = 'PRODUCTS';
```

თქვენ შეგიძლიათ სვეტზე ობიექტური პრივილეგიების შემოწმება user_col_privs მოთხოვნის მეშვეობით, რომელიც აისახება შემდეგ ცხრილში.

Column	**Type**	**Description**
grantee	VARCHAR2(30)	პრივილეგიის მიმღები მომხმარებელი.
table_name	VARCHAR2(30)	ცხრილის სახელი, რომელსაც პრივილეგია მიენიჭა.
column_name	VARCHAR2(30)	სვეტის სახელი, რომელსაც პრივილეგია მიენიჭა.
grantor	VARCHAR2(30)	პრივილეგიის მიმნიჭებელი მომხმარებელი.
privilege	VARCHAR2(40)	პრივილეგია ობიექტზე.
grantable	VARCHAR2(3)	ერთერთი, ვისაც სხვისთვის პრივილეგიის მინიჭება შეუძლია. YES ან NO.

მაგალითი: :

```
SELECT *
FROM user_col_privs_made;
```

თქვენ შეგიძლიათ ცხრილზე ობიექტური პრივილეგიების შემოწმება user_tab_privs მოთხოვნის მეშვეობით, რომელიც აისახება შემდეგ ცხრილში.

Column	**Type**	**Description**
owner	VARCHAR2(30)	ობიექტის ფლობელი მომხმარებელი.

Column	Type	Description
table_name	VARCHAR2(30)	ცხრილის სახელი, რომელსაც პრივილეგია მიენიჭა.
grantor	VARCHAR2(30)	პრივილეგიის მიმნიჭებელი მომხმარებელი..
privilege	VARCHAR2(40)	პრივილეგია ობიექტზე.
grantable	VARCHAR2(3)	ერთერთი, ვისაც სხვისთვის პრივილეგიის მინიჭება შეუძლია. YES ან NO.
hierarchy	VARCHAR2(3)	ერთერთი პრივილეგია, რომელიც იერარქიის ნაწილს აყალიბებს. YES ან NO.

მაგალითი:

```
CONNECT steve/button
SELECT *
FROM user_tab_privs_recd;
```

ობიექტური პრივილეგიების გამოყენება

მაგალითი:

```
CONNECT steve/button
SELECT *
FROM store.customers;
```

```
CUSTOMER_ID FIRST_NAME LAST_NAME   DOB         PHONE
  ----------- ---------- ---------- --------- ------------
        1     John       Brown      01-JAN-65 800-555-1211
        2     Cynthia    Green      05-FEB-68 800-555-1212
        3     Steve      White      16-MAR-71 800-555-1213
        4     Gail       Black                800-555-1214
        5     Doreen     Blue       20-MAY-70
```

სინონიმები

თქვენ შეგიძლიათ საჭირო ცხრილის წვდომა სხვა სქემიდან თუ მივუთითებთ ცხრილის სახელს. თქვენ შეგიძლიათ ცხრილის სახელის ნაცვლად გამოიყენოთ ცხრილის *სინონიმი*, რომელიც იქმნება CREATE SYNONYM ბრძანების მეშვეობით.

მაგალითი:

```
CONNECT system/manager
GRANT CREATE SYNONYM TO steve;
```

მაგალითი:

```
CONNECT steve/button
CREATE SYNONYM customers FOR store.customers;
```

და შემდეგ სინონიმს ვიყენებთ:

```
SELECT *
FROM customers;
```

მაგალითი:

```
SQL> CONNECT gail/seymour
Connected.
SQL> SELECT * FROM products;
SELECT * FROM products
         *
ERROR at line 1:
ORA-00942: ცხრილი or წარმოდგენა does not exist
```

თუ gail-ს გააჩნია SELECT ობიექტური პრივილეგია ცხრილზე store.products, მაშინ ბრძანება SELECT წარმატებით შესრულდება.

ობიექტური პრივილეგიების გაუქმება

ობიექტური პრივილეგიების გაუქმებისათვის გამოიყენება ბრძანება REVOKE.

მაგალითი:

```
CONNECT store/store_password
REVOKE INSERT ON products FROM steve;
```

მაგალითი:

```
REVOKE SELECT ON store.customers FROM steve;
```

4.როლები

როლი (*role*) წარმოადგენს პრივილეგიების ჯგუფს, რომელიც მომხმარებელს მიეწერება. როლის შესაქმნელად საჭიროა CREATE ROLE სისტემური პრივილეგიის ქონა. მომხმარებლის არსებობა ითხოვს სისტემური პრივილეგიის CREATE USER მინიჭებას ADMIN ოპციასთან ერთად.

მაგალითი:

```
CONNECT system/manager
GRANT CREATE ROLE TO store;
GRANT CREATE USER TO store WITH ADMIN OPTION;
```

როლის შესაქმნელად გამოიყენებაბრძანება CREATE ROLE :

```
CONNECT store/store_password
CREATE ROLE product_manager;
CREATE ROLE hr_manager;
CREATE ROLE overall_manager IDENTIFIED by manager_password;
```

როლები

Role Name	**Has Permissions to …**
product_manager	SELECT, INSERT, UPDATE და DELETE ოპერაციების შესრულება product_types და products ცხრილებისათვის.
hr_manager	SELECT, INSERT, UPDATE და DELETE ოპერაციების შესრულება salary_grades და employees ცხრილებისათვის. აგრეთვე, hr_manager-ს შეუძლია მომხმარებლის შექმნა.
overall_manager	SELECT, INSERT, UPDATE და DELETE ოპერაციების შესრულება ყველა წინა როლებში ნაჩვენებ ცხრილებში; overall_manager იქნება გრანტირებული წინა როლებით.

როლებისათვის პრივილეგიების მინიჭება

მაგალითი:

```
GRANT SELECT, INSERT, UPDATE, DELETE
 ON product_types TO product_manager;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE
 ON products TO product_manager;
GRANT SELECT, INSERT, UPDATE, DELETE
 ON salary_grades TO hr_manager;
GRANT SELECT, INSERT, UPDATE, DELETE
 ON employees TO hr_manager;
GRANT CREATE USER TO hr_manager;
GRANT product_manager, hr_manager TO overall_manager;
```

მომხმარებლებისათვის როლების მინიჭება

user_role_privs მომხმარებლის ზოგიერთი სვეტი

Column	**Type**	**Description**
username	VARCHAR2(30)	მომხმარებლის სახელი, რომელსაც როლი ექნება მინიჭებული.
granted_role	VARCHAR2(30)	მომხმარებლისათვის მინიჭებული როლის სახელი.
admin_option	VARCHAR2(3)	ერთერთი მომხმარებელი, ვისაც სხვისთვის როლის მინიჭება შეუძლია. YES ან NO.
default_role	VARCHAR2(3)	ერთერთი როლი, რომელიც უსიტყვო რეჟიმში გამორთულია, როცა მომსმარებელი უერთდება მონაცემთა ბაზას. YES ან NO.
os_granted	VARCHAR2(3)	ერთერთი როლი, რომელიც ოპერაციული სისტემის მიერ არის მინიჭებული.

ქვემოთ მოყვანილ მაგალითში ხდება მომხმარებლისა და user_role_privs მოთხოვნის შეერთება:

```
CONNECT steve/button
SELECT *
FROM user_role_privs;
```

ანალოგიურად შემდეგ მაგალითში ხდება მომხმარებლისა და user_role_privs მოთხოვნის შეერთება:

```
CONNECT store/store_password
SELECT *
FROM user_role_privs;
```

როლისათვის სისტემური პრივილეგიების მინიჭების შემოწმება

სისტემური პრივილეგიების როლისათვის მინიჭება ხდება role_sys_privs მოთხოვნის მეშვეობით:

role_sys_privs როლის ზოგიერთი სვეტი

Column	Type	Description
role	VARCHAR2(30)	როლის სახლწოდება.
privilege	VARCHAR2(40)	როლისათვის სისტემური პრივილეგიის მინიჭება.
admin_option	VARCHAR2(3)	ერთერთი პრივილეგია, რომელიც მიენიჭება ADMIN ოპციით. YES ან NO.

```
SELECT *
FROM role_sys_privs;
```

როლისათვის ობიექტური პრივილეგიების მინიჭების შემოწმება

ობიექტური პრივილეგიების როლისათვის მინიჭება ხდება role_tab_privs მოთხოვნის მეშვეობით:

role_tab_privs როლის ზოგიერთი სვეტი

Column	Type	Description
role	VARCHAR2(30)	მომხმარებელი, ვისაც პრივილეგია ენიჭება.
owner	VARCHAR2(30)	ობიექტის მფლობელი მომხმარებელი.
table_name	VARCHAR2(30)	ობიექტის სახელი, რომელსაც პრივილეგია მიენიჭა.

role_tab_privs როლის ზოგიერთი სვეტი

Column	Type	Description
column_name	VARCHAR2(30)	სვეტის სახელი (თუ გამოიყენება).
privilege	VARCHAR2(40)	პრივილეგია ობიექტზე.
gran table	VARCHAR2(3)	ერთერთი პრივილეგია, რომელიც ენიჭება GRANT ოპციით. YES ან NO.

მაგალითი:

```
SELECT *
FROM role_tab_privs
WHERE role='HR_MANAGER';
```

როლებისათვის SELECT ობიექტური პრივილეგიების მინიჭება

მაგალითი:

```
CONNECT steve/button
SELECT p.name, pt.name
FROM store.products p, store.product_types pt
WHERE p.product_type_id = pt.product_type_id;
```

როლის გაუქმება

როლის გაუქმებისათვის გამოიყენება ბრძანება REVOKE.

მაგალითი:

```
CONNECT store/store_password
REVOKE overall_manager FROM steve;
```

როლიდან პრივილეგიის გაუქმება

მაგალითი:

```
REVOKE ALL ON products FROM product_manager;
REVOKE ALL ON product_types FROM product_manager;
```

როლის ამოგდება

როლის ამოგდებისათვის გამოიყენება ბრძანება DROP ROLE.

მაგალითი:

```
DROP ROLE overall_manager;
DROP ROLE product_manager;
DROP ROLE hr_manager;
```

ლიტერატურა

1. მეფარიშვილი ბ. `მონაცემთა ბაზების მართვის სისტემები", სახელმძღვანელო, სტუ, 2009. 681.3.016(02)/13.
2. მეფარიშვილი ბ. `მონაცემთა ბაზების ადმინისტრირება", სახელმძღვანელო, სტუ, 2009. 681.3.016(02)/10.
3. მეფარიშვილი ბ. `მონაცემთა ბაზების მართვის სისტემები", მეთოდური მითითებანი ლაბორატორიული და პრაქტიკული მეცადინეობებისათვის, სტუ, 2008. 681.3.016(02)/17.
4. რომან სამხარაძე SQL სერვერი © საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი||,2008. 681.3.016(02)/16. ISBN 978-9941-14-190-4. http://www.gtu.ge/publishinghouse/
5. სურგულაძე გ., შონია ო., ყვავაძე ლ. მონაცემთა ბაზების მართვის სისტემები (Ms SQL Server). დამხმ.სახელმძღ., სტუ, 2004. ბიბლ.: 681.3.06 (02).38; http://www.gtu.ge/katedrebi/kat94/pdf/sql.pdf