Prijava na bazu podataka - SQLTools

Hostname: **???**
Username : **???**
Password: **???**
Host string: **etflab**

Constraints – ograničenja nad tabelama

Šta su constraint-i (ograničenja) na tabelama?
Najednostavnije rečeno, constraint-i su pravila ograničenja nad pojedinim kolonama tabele kojim se osigurava konzistentnost i integtritet podataka u datim kolonama tabele, pa tako i u samoj bazi podataka. Na osnovu ovih ograničenja nedozvoljava se narušavanje «očekivanih» podataka u datim tabelama na način koji se očekuje po osnovu projektne dokumentacije šeme baze podataka.

Postoji više vrsta ograničenja koji se mogu uvesti za svaku od kreiranih tabela:

NOT NULL	- je constraint-a koji određuje da kolona koja ga sadrži ne može sadržavati NULL vrijednost.
UNIQUE KEY	- je constraint koji osigurava da kolona ili kombinacija više kolona mora sadržavati jednoznačne vijednosti za sve slogove u tabeli.
PRIMARY KEY	- jedinstveno identificira svaki slog tabele.
FOREIGN KEY	- uspostavlja i osigurava vezu tj. strani ključ između kolone i kolone referencijalne tabele.
CHECK	- određuje uslov koji mora biti istinit.

Svi constraint-i su pohranjeni u data dictionary-u baze podataka. Iz praktičnih razloga bar za constraint-e: UNIQUE, PRIMARY KEY i FOREIGN KEY, trebalo bi voditi računa da im se daju «pametna» imena prilikom njihovog kreiranja kako bi se odmah, po nazivu constraint-a odmah znalo o kojem constraint-u je riječ, i na taj način brže otklanjali problemi kod pokušaja unosa podataka koji nisu konzistentni i očekivani od strane kolone i/ili kolona u naznačenim tabelama, a samim tim i same šeme baze podataka nad kojom su kreirani naznačeni constraint-i.

Svi kreirani constraint-i nad tabelema za korisnika koji je prijavljen na bazu mogu se vidjeti kroz poglede data dictionary-a:

- ALL_CONSTRAINTS
- USER_CONSTRAINTS
- ALL_CONS_COLUMNS
- USER_CONS_COLUMNS

Generalizirana sinteksa za kreiranje constraint-a je data sljedećom strukurom:

```
SQL>  CREATE TABLE  šema.ntable (
      ( kolona1 tip podatka /default-a vrijednost/ /constraint kolone1/,
        kolona2 tip podatka /default-a vrijednost/ /constraint kolone2/,
        kolona3 tip podatka /default-a vrijednost/ /constraint kolone3/,
        ...
        constraint1 tabele,
        ...,
      constraint2 tabele,
```

```
)
```

U sintaksi je:

šema	- je naziv korisnika koji se konektovao na bazu podataka. *Napomena:* Naziv šeme se ne mora navoditi prilikom kreiranja tabele i po default-u to je korisnik koji je prijavljen na bazu, osnosno kreira objekte u bazi podataka.
ntable	- je naziv tabele koja kreira u naznačenoj šemi baze podataka.
kolona 1..n	- je naziv kolone
tip podatka	- je tip podatka kolone
default-a vrijednost	- određuje default-u vrijednost za kolonu, koja se insertuje, po insertu novog sloga, ako po insertovanju novog sloga nije naznačena vrijednost za insert u dotičnu kolonu.
constraint kolone 1..n	- je constraint integriteta, kao dio definicije naznačene kolone.
constraint 1..n tabele	- je constraint integriteta, kao dio definicije naznačene tabele.

Tako na primjer, za tabelu zaposlenih constraint da se ne može unijeti NULL vrijednost za ime, prezime i platu, kao i da kolona id određuje jednoznačno sve ostale atribute tabele data je sljedećim SQL-m:

```
SQL> CREATE TABLE zaposleni(id NUMBER,
                            ime VARCHAR2(50)       NOT NULL,
                            prezime VARCHAR2(50) NOT NULL,
                            plata NUMBER             NOT NULL,
                    CONSTRAINT c_zap_PK PRIMARY KEY (id));
```

Definisanje constraint-a

Kao što je već ranije naznačeno postoje dva nivoa gdje je moguće deklarisati constraint-e, i to:

- constraint na nivou kolone

```
SQL> Kolona /CONSTRAINT naziv_ograničenja/ /tip_ograničenja/,
```

- constraint na nivou tabele

```
SQL> Kolona, ...
        /CONSTRAINT naziv_ograničenja/ /tip_ograničenja/
        (kolona, ...),
```

Constraint-i se obično kreiraju u isto vrijeme kao i sama tabela. Constraint-i se mogu naknadno kreirati poslije kreiranja tabele, ali i privremeno onemogućiti po potrebi. Constraint-i se mogu definisati na dva nivou kao što su:

Nivo constraint-a	Opis
Kolona	Odnosi se na samo jednu kolonu i moguće je difinisati bilo koji tip constraint-a ali samo za datu kolonu.
Tabela	Odnosi se na jednu ili više kolona i definiše se odvojeno od definicija kolona u tabeli. Na ovom nivou moguće je definisati bilo koji tip constraint-a izuzev NOT NULL.

U sintaksi je:

naziv_ograničenja - je naziv constraint-a, koji se proizvoljno definiše od strane dizajnera šeme baze podataka.

tip_ograničenja - je tip constraint-a, koji je neophodno specificirati prilikom deklaracije constraint-a, tj. da bi se znalo o kojoj vrsti constrait-a je riječ.

NOT NULL constraint

NOT NULL constraint osigurava da se ne mogu unijeti NULL vrijednosti za one kolone nad kojima je definisan ovaj constraint. Ovaj constraint je moguće definisati jedino na nivou kolone.

```
SQL> CREATE TABLE tab_primjer1(sifra NUMBER,
                               naziv VARCHAR2(50) NOT NULL);
```

UNIQUE KEY constraint

UNIQUE KEY integrity constraint osigurava da svaka vrijednost u koloni ili grupi kolona (ključ) ima jedinstvenu vrijednost, tj. ne mogu postojati dva sloga u tabeli koji sadrže duple vrijednosti u navedenoj koloni ili grupi kolona. Kolona ili grupa kolona uključenih u definiciju UNIQUE constraint-a naziva se UNIQUE KEY-m.

```
SQL> CREATE TABLE tab_primjer2(sifra NUMBER,
                               naziv VARCHAR2(50),
        CONSTRAINT tab_pri2_uk UNIQUE (sifra));
```

PRIMARY KEY constraint

PRIMARY KEY constraint kreira primarni ključ za tabelu. Samo se jedan primarni ključ može kreirati za svaku tabelu. PRIMARY KEY constraint je kolona ili grupa kolona koja jednoznačno određuje svaki slog u tabeli. Kolona ili grupa kolona nad kojom je kreiran PRIMARY KEY constraint osigurava da se u datoj koloni i/ili grupi kolona ne može unijeti NULL vrijesnost, u protivnom slog neće biti prihvačen od strane baze podataka.

```
SQL> CREATE TABLE tab_primjer3(sifra NUMBER,
                               naziv VARCHAR2(50),
        CONSTRAINT tab_pri3_pk PRIMARY KEY (sifra));
```

FOREIGN KEY constraint

FOREIGN KEY je referencijalni integrity constraint, difinisan nad kolonom ili grupom kolona kao strani ključ i uspostavlja vezu između PRIMARY ili UNIQUE KEY-a iste ili različite tabele. Vrijednost koja se može unijeti u kolonu ili grupu kolona nad kojim je definisan FOREIGN KEY mogu biti samo one vrijednosti koji su predefinisane sa referencijalnom vezom ili NULL vrijednost.

```
SQL> CREATE TABLE tab_primjer4(sifra NUMBER,
                               naziv VARCHAR2(50),
                               odjel NUMBER,
        CONSTRAINT tab_pri4_fk FOREIGN KEY (odjel)
        REFERENCES odjeli (sifra_odjela));
```

CHECK constraint

CHECK constraint definiše uslov koji mora zadovoljiti svaki slog tabele. Ovaj constraint može se definisati i na nivou kolone i na nivou tabele. Jedna od interesantnih stvari je da ne postoji broj ograničenja koliko je moguće definisati CHECK constraint-a nad jednom kolonom tabele.

```
SQL> CREATE TABLE tab_primjer5(sifra NUMBER,
                               naziv VARCHAR2(50),
                               plata NUMBER,
        CONSTRAINT tab_pri_plata_ck
        CHECK (plata BETWEEN 0 AND 1000));
```

Naknadno dodavanje constrainta tabeli vrši se pomoću komande:

```
SQL> ALTER TABLE naziv_tabele ADD CONSTRAINT naziv_constrainta
tip_constrainta (kolona);
```

Na primjer, predpostavimo da je potrebno dodati PRIMARY KEY constraint našoj kreiranoj tabeli tab_primjer1 nad kolonom sifra:

```
SQL> ALTER TABLE tab_primjer1 ADD CONSTRAINT c_tab_pri_PK PRIMARY
KEY (sifra);
```

Brisanje kreiranih constraint-a nad datim tabelama vrši se pomoću DROP CONSTRAINT. Na primjer, neka je potrebno obrisati FOREIGN KEY constraint u kreiranoj tabeli tab_primjer4:

```
SQL> ALTER TABLE tab_primjer4 DROP CONSTRAINT tab_pri4_fk;
```

Onemogućavanje i omogućavanje constraint-a vrši se pomoću komandi DISABLE / ENABLE CONSTRAINT, na primjer, predpostavimo da je potrebno onemogućiti CHECK CONSTRAINT u tabeli tab_primjer5.

```
SQL> ALTER TABLE tab_primjer5 DISABLE CONSTRAINT tab_pri_plata_ck;
```

Pogledi

Pogledi se najednostavnije mogu nazvati virtuelnim tabelama. Svrha pogleda je da skrate složene upite, a nakon što se i kreira nad skupom podataka neke i/ili nekih tabela može se treirati kao i sama tabela. Međutim, za razliku od tabele za poglede postoje specijalna ograničenja kod promjene podataka, s tom razlikom da se promijene nad podacima u tabelama automatski reflektuju i u pogledima koji koriste navedene tabele. Pogledi za razliku od tabela ne zauzimaju fizički prostor u bazi podataka.

Postoje dvije klasifikacije pogled i to: jednostani i složeni. Osnovna razlika između navedenih klasifikacija pogleda je u upotrebi DML (insert, update i delete) operacija.

Jednostavni pogled je jedan od onih koji:

- Vraća podatke iz samo jedne tabele
- Ne sadrži funkcije ili grupne podatke
- Mogu se izvršavati DML operacije kroz navedeni pogled

Složeni pogled je jedan od onih koji:

- Vraća podatke iz više tabela
- Sadrži funkcije ili grupne podatke
- Nisu uvijek dozvoljene DML operacije kroz nadeveni pogled

Generalizirana sintaksa za kreiranje pogleda data je sljedećom strukturom:

```
SQL> CREATE /OR REPLACE/  /FORCE|NOFORCE/  VIEW naziv_pogleda
/(alias1, alias2, ...)/
AS
podupit
/WITH CHECH OPTION /CONSTRAINT ograničenje//
/WITH READ ONLY/
```

U sintaksi je:

CREATE OR REPLACE	- ključne riječi za kreiranje i re-kreiranje pogleda ako već postoji kreiran u bazi podataka.
FORCE	- pogled se kreira bez obzira da li postoje ili ne osnovne tabele neophodne za kreiranje pogleda koje koristi podupit.
NOFORCE	- pogled se kreira jedino ako postoje osnovne tabele neophodne za pogled. Ovo je podešeno po default-u.
naziv_pogleda	- je naziv pogleda koje korisnik određuje po njegovom kreiranju.
podupit	- je SELECT koji predstavlja izvor informacija za predefinisani podupit.
WITH CHECK OPTION	- specificira da jedino prihvatljivi slogovi mogu biti insetovani i ažurirani u tabeli, koji predstavljaju izvor informacija za podupit.
ograničenje	- je naziv constraint-a specificiranog sa CHECK OPTION constraint-m.
WITH READ ONLY	- osigurava da se ne može izvršiti nijedna DML opercija nad definisanim pogledom.

Na primjer, pretpodstavimo da je potrebno kreirati pogled «zap_view» sa istim nazivom kolona kao i tabela zaposlenih, za sve zaposlene iz odjela 30.

```
SQL> CREATE VIEW zap_view
AS
SELECT *
```

```
FROM employees
WHERE department_id = 30;
```

Opis strukture kreiranog pogleda može se vidjeti upotrebom komande DESCribe

```
SQL> DESC zap_view;
```

Kod kreiranja kompleksnih pogleda, najčešće postoji situacija da je potrebno u podupitu koristiti grupne funkcije ili vršiti određena grupisanja kako bi se dobio adekvatan rezultat. Na primjer, pretpostavimo da je potrebno kreirati podupit koji će vratiti odjel, minimalnu, maksimalnu i prosječnu platu po odjelima:

```
SQL> CREATE VIEW plata_odjela(odjel, min_plata, max_plata, sre_plata)
     AS
     SELECT d.department_name, min(e.salary), max(e.salary), avg(e.salary)
     FROM employees e, departments d
     WHERE e.department_id = d.department_id
     GROUP BY d.deparment_name;
```

Brisanje pogleda vrši se jednostavno upotrebom ključne riječi DROP VIEW. Na primjer, predhodni pogled bi se obrisao sljedećom SQL komandom:

```
SQL> DROP VIEW plata_odjela;
```

Sadržaj podupita pogleda može se vidjeti kroz sljedeće objekte data dictionary-a:

- ALL_VIEWS
- USER_VIEWS

Sekvence

Sekvence ja objekat baze podatka koji je specifičan samo za Oracle bazu podatka, a svrha sekvence je da osigurava jednoznačne integer vrijednosti. Najčešći način upotrebe sekvence je automatsko generisanje vrijednosti za primarne ključeve tabela, odnosno vrijednosti koje moraju biti jednoznačne za sve slogove tabele. Dobra strana sekvence je da osigurava jednoznačne vrijednosti svaki put kada je «prozvana», i kako je nezavisna od tabele, znači da se ista sekvenca može koristi za osiguravanje jednoznačnih vrijednosti u više od jedne tabele i od strane više različitih korisnika u isto vrijeme. Međutim, loša strana sekvence je da kada je jednom «prozvana» i vrijednost «potrošena» bez obzira da li se želi ili ne koristiti data vrijednost sekvence u tom momentu, čak i kod upotrebe rollback komande, nema više povratka na staru vrijednost.

Sintaksa za kreiranje sekvence data je sljedećom strukturom:

```
SQL> CREATE SEQUENCE naziv_sekvence
     /INCREMENT BY int_broj/
     /START WITH int_broj/
     /MAXVALUE int_broj | NOMAXVALUE/
     /MINVALUE int_broj | NOMINVALUE/
     /CYCLE | NOCYCLE/
     /CACHE int_broj | NOCACHE/
     ;
```

U sintaksi je:

naziv_sekvence	- je naziv generatora sekvence.
INCREMENT BY int_broj	- je korak ili interval između dvije uzastopno generisane sekvence date brojem int_broj. Ako se ova linija izostavi, prilikom kreiranja sekvence, default-a vrijednost koraka će biti 1.
START WITH int_broj	- određuje od kojeg će se broja int_broj početi sekvence brojeva generisati. U slučaju da se ova linija izostavi, prilikom generisanja sekvence, detault-a vrijednost će biti 1.
MAXVALUE int_broj	- određuje do kojeg broja int_broj će se generisati sekvenca brojeva.
NOMAXVALUE	- određuje maksimalnu vrijednost 10^{27} za rastuću, odnosno –1 za opadajuću sekvencu.
MINVALUE int_broj	- određuje od kojeg broja int_broj će se generisati sekvenca brojeva.
NOMINVALUE	- određuje minimalnu vrijednost -1 za rastuću, odnosno -10^{26} za opadajuću sekvencu.
CYCLE | NOCYCLE	- određuje da li će se nakon što se generišu svi brojevi sekvence u rasponu minimalnih i maksimalnih vrijednost početi sa generisanjem brojeva od početka CYCLE. Default-a postavka je NOCYCLE, tj. da se prestaje sa generisanjem brojeva nakon što se izgenerišu svi mogući brojevi u definisanom rasponu brojeva.
CACHE int_broj | NOCACHE	- određuje koliko će vrijednosti biti prealocirano i sačuvano u bazi podataka. Po default-u biti će sačuvano 20 vrijednosti.

Na primjer, pretpostavimo da je potrebno kreirati sekvencu test_seq koja će početi generisati vrijednosti od broja 50012, sa korakom 1, i maksimalno dozvoljenom vrijednošću 50100. Sekvencu treba kreirati tako da nakon generisanja svih brojeva

sekvenca ne generiše vrijednosti dalje. Nije neophodno alociranje vrijednosti sekvence u bazi podataka.

```
SQL> CREATE SEQUENCE test_seq
     INCREMENT BY 1
     START WITH 50012
     MAXVALUE 50100
     NOCACHE
     NOCYCLE
```

Sekvenca posjeduje, uslovno rečeno, dvije metode na osnovu kojih je moguće generisati i vratiti trenutnu vrijednost sekvence i to:

NEXTVAL - vraća sljedeći broj sekvence.
CURRVAL - vraća trenutno generisani broj sekvence.

Generisanje nove vrijednosti sekvence test_seq:

```
SQL> SELECT test_seq.NEXTVAL FROM dual;
```

Odnosno, vraćanje zadnje generise vrijednosti:

```
SQL> SELECT test_seq.CURRVAL FROM dual;
```

Sve kreirane sekvence za korisnika koji je prijavljen na bazu mogu se vidjeti kroz poglede data dictionary-a:

- ALL_SEQUENCES
- USER_SEQUENCES

Promjena za neku od definisanih vrijednosti sekvenci moguće je izvršiti putem ALTER SEQUENCE komande, ali sa jednom iznimkom, a to je da nije moguće mijenjati START WITH vrijednost sekvence. Za prethodni primjer neka je potrebno setovati sekvencu kao što je bila nakon kreiranja ali tako da maksimalno dozvoljena vrijednost bude 100000 umjesto 50100.

```
SQL> ALTER SEQUENCE test_seq
     INCREMENT BY 1
     MAXVALUE 100000
     NOCACHE
     NOCYCLE
```

Brisanje sekvence moguće je izvršiti putem komande DROP SEQUENCE

```
SQL> DROP SEQUENCE test_seq;
```

Zadaci

1. Kreirajte vašu tabelu zaposlenih, sa analognim podacima kao i tabela employees, gdje će te dodati novu kolonu «id» nad kojom će biti definisan primary key. Odaberite adekvatan tip podataka «id» kolone prema planiranim i/ili ubačenim vrijednostima.
2. Kreirajte vašu tabelu odjela, sa analognim podacima kao i tabela departments, gdje će te dodati nove kolone «id i datum» nad kojom će biti definisan primary key. Odaberite adekvatne tipove podataka za naznačene kolone.
3. Re-dizajnirajte vašu tabelu zaposlenih tako da je moguće kreirati foreign key između vaše tabele zaposlenih i odjela. Neophodno je ažurirati sve slogove tabele zaposlenih kako bi se prenijele potrebne informacije iz vaše table odjela u vašu tabelu zaposlenih.
4. Provjerite koji sve constraint-i postoje nad vašom šemom baze potom nad šemom «hr», a potom nad šemom «test». Da li svaka tabela u šemi «hr» posjeduje primary key?
5. Prikažite sve objekte koji imaju neke veze sa tabelom EMPLOYEES i DEPARTMENTS iz šeme «hr».
6. Modificirajte tabelu zaposlenih tako što će te dodati novu kolonu plata_dodatak koji će sadržavati platu uvećanu za dodatak na platu samo za zaposlene iz Amerike.
7. Dodajte CHECK constraint za kolonu kreiranu u 6 zadatku za razuman raspon vrijednosti.
8. Kreirajte pogled sa nazivom zap_pog sa sljedećim kolonama: šifra zaposlenog, naziv zaposlenog i naziv odjela, za sve zaposlene koji primaju platu veću od prosječne plate odjela u kojem rade.
9. Prikažite sadržaj kreiranog pogleda. Da li je moguće kombinovati poglede s osnovnim tabelama baze.
10. Kreirajte pogled koji će vratit naziv posla, naziv odjela, prosječnu platu i iznos dodataka na platu po datim poslovima i odjelima za sve poslove i odjele koji u imenu sadrže slova «a», «b», i «c» na bilo kojoj poziciji. Pogled se može koristiti samo za pregled.
11. Modificirajte predhodni pogled tako da uvedete novu kolonu koja će sadržavati prosječnu platu po odjelima
12. Modificirajte predhodni pogled tako da novo-kreirana kolona nosi naziv «prosječna plata odjela»
13. Kreirajte pogled koji će vratiti naziv šefa, broj zaposlenih koji su podređeni šefu, kao i minimalnu i maksimalnu platu odjela u kojem naznačeni šef radi.
14. Modificirajte predhodni primjer tako da se pogled može koristiti samo za čitanje, i dodajte novu kolone koje će sadržavati sumarnu platu sa dodacima na platu za sve zaposlene kojima je naznačeni šef nadređen.