Database Programming with SQL
kurs 2017 – database design and programming with sql
students slajdovi

12-1 INSERT Statements

- Treba znati kako promeniti podatke u db; u biznisu db su dinamičke, konstantno u procesu ubacivanja, updejta i brisanja podataka; korišćenje DML iskaza za promene u db

Copy Tables Before Inserting

- Programer je odgovoran za menjanje i povraćanje tabela u svojoj schema (šemi)
- Da bi se čuvale šema tabele u svom originalnom stanju, prave se kopije svake tabele pre završetka praktičnih aktivnosti
- U svakoj praktičnoj aktivnosti, koristiće se kopija tabele koja se kreira a ne original
- Svaka kopija tabele se imenuje kao: copy_tablename
- Kopije tabele neće naslediti pridružene primary-to-foreign-key integrity pravila (ograničenja relacija) originalnih tabela
- Tipovi podataka kolona su nasleđene u kopiranim tabelama

Syntax to Create a Copy of a Table

- Create table syntax:

```
CREATE TABLE copy_tablename
   AS (SELECT * FROM tablename);
```

- For example:

```
CREATE TABLE copy_employees
AS (SELECT * FROM employees);
```

```
CREATE TABLE copy_departments
AS (SELECT * FROM departments);
```

- To verify and view the copy of the table, use the following DESCRIBE and SELECT statements:

```
DESCRIBE copy_employees;
```

```
SELECT * FROM copy_employees;
```

```
DESCRIBE copy_departments;
```

```
SELECT * FROM copy_departments;
```

- Struktura i sadržaj tabele se može takođe videti korišćenjem Object Browser u APEX

INSERT

- Iskaz INSERT se koristi za dodavanje novog reda u tabelu i zahteva tri vrednosti: ime tabele, imena kolona u tabeli koja će se uneti, odgovarajuće vrednosti za svaku kolonu
- Kako se unosi nov podatak u sledeću tabelu kreiranjem novog sektora u copy_departments tabeli ?

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
200	Human Resources	205	1500

- Sledeća sintaksa koristi INSERT za dodavanje novog sektora u copy_departments tabeli i ovaj iskaz eksplicitno lista svaku kolonu kako se ona pojavljuje u tabeli
- Vrednosti za svaku kolonu su izlistane u istom redosledu; brojne vrednosti nisu pod apostrofima

```
INSERT INTO copy_departments
   (department_id, department_name, manager_id, location_id)
VALUES
   (200,'Human Resources', 205, 1500);
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
200	Human Resources	205	1500

- Redosled kolona se može izlistati u bilo kojem redosledu (iako nema koristi u ovome) sve dok vrednosti kolona su izlistane u istom redosledu kao i imena kolona
- Drugi način za unos vrednosti u tabeli je implicitno dodavanje njih izbegavajući imena kolona
- Upozorenje: vrednosti za svaku kolonu moraju odgovarati tačno difolt redosled po kojem se pojavljuju u tabeli (opisano u DESCRIBE iskazu) a vrednost mora biti omogućena sa svakom kolonom
- U sledećem primeru, INSERT se piše bez eksplicitnog imenovanja kolona
- Zbog jasnoće, ipak najbolje je koristiti imena kolona u INSERT izrazu

```
INSERT INTO copy_departments
VALUES
  (210,'Estate Management', 102, 1700);
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
210	Estate Management	102	1700

Check The Table First

- Pre unosa podataka u tabelu, mora se proveriti nekoliko detalja u tabeli
- DESCRIBE tablename iskaz će vratiti opis strukture tabele u tabelarnoj summary chart
- COPY_DEPARTMENTS TABLE SUMMARY:

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable
DEPARTMENT_ID	NUMBER	-	4	0	-	✓
DEPARTMENT_NAME	VARCHAR2	30	-	-	-	-
MANAGER_ID	NUMBER	-	6	0	-	✓
LOCATION_ID	NUMBER	-	4	0	-	✓

- Pošto je ova tabela kreirana kao kopija departments tabele, ograničenja osnovnog ključa nisu kopirana iz originala

Table Summary

- Zbirna tabela daje informaciju o svakoj koloni u tabeli:
  - the allowance of duplicate values
  - the type of data allowed
  - the amount of data allowed
  - the allowance of NULL values

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable
EMPLOYEE_ID	NUMBER	-	6	0	1	-
FIRST_NAME	VARCHAR2	20	-	-	-	✓
LAST_NAME	VARCHAR2	25	-	-	-	-
EMAIL	VARCHAR2	25	-	-	-	-
PHONE_NUMBER	VARCHAR2	20	-	-	-	✓
HIRE_DATE	DATE	7	-	-	-	-
JOB_ID	VARCHAR2	10	-	-	-	-
SALARY	NUMBER	-	8	2	-	✓
COMMISSION_PCT	NUMBER	-	2	2	-	✓
MANAGER_ID	NUMBER	-	6	0	-	✓
DEPARTMENT_ID	NUMBER	-	4	0	-	✓
BONUS	VARCHAR2	5	-	-	-	✓

- Primetiti gde “Data Type“ kolona je znakovni tip podataka “Length“ kolona specificira maksimalni broj dozvoljenih karaktera
- First_name ima tip podataka VARCHAR2 i Length 20, što znači da do 20 karaktera se može uneti za ovu kolonu
- Za brojčane tipove podataka zagrade određuju preciznost i skalu
- Preciznost je totalan broj cifara, a skala je broj cifara sa desne strane decimalnog zareza
- Kolona Salary omogućava brojeve sa preciznosti od 8 i skalom od 2
- Maksimalna vrednost dozvoljena u ovoj koloni je 999999.99

Inserting Rows With Null Values

- Iskaz INSERT ne mora da se specificira za svaku kolonu – kolone sa null se mogu izbeći
- Ako svakoj koloni koja zahteva vrednost je dodeljena vrednost, insert radi
- U ovoj koloni, EMAIL kolona je definisana kao NOT NULL kolona
- Implicitan pokušaj dodavanja tabele kao što je prikazano će proizvesti grešku

```
INSERT INTO copy_employees
  (employee_id, first_name, last_name, phone_number, hire_date,
   job_id, salary)
VALUES
  (302,'Grigorz','Polanski', '8586667641', '15-Jun-2015',
   'IT_PROG',4200);
```

```
ORA-01400: cannot insert NULL into
("US_A009EMEA815_PLSQL_T01"."COPY_EMPLOYEES"."EMAIL")
```

- EMAIL kolona je izbačena iz insert, pa insert pada pošto NULL vrednost nije dozvoljena za EMAIL kolonu
- Jedan implicitan insert će automatski uneti null vrednost u kolonu koja dozvoljava null
- Da bi se eksplicitno dodala null vrednost u kolonu koja dozvoljava null, koristiti NULL službenu reč u VALUES listi
- Za specificiranje praznog stringa i/ili nedostajuće datume, koristiti prazne apostrofe (bez praznog mesta između njih ``) za nedostajuće podatke

```
INSERT INTO copy_employees
  (employee_id, first_name, last_name, email, phone_number,
   hire_date, job_id, salary)
VALUES
  (302,'Grigorz','Polanski', 'gpolanski', '', '15-Jun-2015',
   'IT_PROG',4200);
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
302	Grigorz	Polanski	gpolanski	-	15-Jun-2015	IT_PROG	4200

.....	COMM_PCT	MGR_ID	DEPT_ID	BONUS
	-	-	-	-

- Imena kolona na izlazu su izmenjena zbog slike

Inserting Special Values

- Specijalne vrednosti poput SYSDATE i USER mogu biti unešene u VALUES listi u INSERT iskazu
- SYSDATE će staviti trenutni datum i vreme u kolonu
- USER će uneti trenutno username sesije, koje je OAE_PUBLIC_USER u Oracle Application Express
- Ovaj primer dodaje USER kao prezime i SYSDATE za hire date

```
INSERT INTO copy_employees
  (employee_id, first_name, last_name, email, phone_number, hire_date,
   job_id, salary)
VALUES
  (304,'Test',USER, 't_user', 4159982010, SYSDATE, 'ST_CLERK',2500);
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
304	Test	APEX_PUBLIC_USER	t_user	4159982010	15-Jun-2015	ST_CLERK	2500

.....	COMM_PCT	MGR_ID	DEPT_ID	BONUS
	-	-	-	-

Inserting Specific Date Values

- Difolt format model za datum tip podataka je DD-Mon-YYYY
- Sa ovim formatom datuma, difolt vreme za ponoć (00:00:00) je takođe uključeno
- U ranijim sekcijama, prikazano je kako koristiti TO_CHAR funkciju za konvertovanje datuma u string znakova kada treba vratiti i prikazati vrednost datuma u non-default formatu
- Ovo je podsećanje za TO_CHAR:

```
SELECT first_name, TO_CHAR(hire_date,'Month, fmdd, yyyy')
FROM employees
WHERE employee_id = 101;
```

FIRST_NAME	TO_CHAR(HIRE_DATE,'MONTH,FMDD,YYYY')
Neena	September, 21, 1989

- Slično, ako želimo INSERT red sa non-difolt formatom za kolonu datuma, mora se koristiti TO_DATE funkcija za konverziju vrednosti datuma (string znakova) u datum

```
INSERT INTO copy_employees
  (employee_id, first_name, last_name, email, phone_number, hire_date,
   job_id, salary)
VALUES
   (301,'Katie','Hernandez', 'khernandez','8586667641',
    TO_DATE('July 8, 2015', 'Month fmdd, yyyy'), 'MK_REP',4200);
```

- Sledeći primer TO_DATE omogućava unos specifičnog vremena u danu, menjajući difolt vreme u ponoć

```
INSERT INTO copy_employees
  (employee_id, first_name, last_name, email, phone_number, hire_date,
   job_id, salary)
VALUES
  (303,'Angelina','Wright', 'awright','4159982010',
   TO_DATE('July 10, 2015 17:20', 'Month fmdd, yyyy HH24:MI'),
   'MK_REP', 3600);
```

```
SELECT first_name, last_name,
   TO_CHAR(hire_date, 'dd-Mon-YYYY HH24:MI') As "Date and Time"
FROM copy_employees
WHERE employee_id = 303;
```

FIRST_NAME	LAST_NAME	Date and Time
Angelina	Wright	10-Jul-2015 17:20

Using A Subquery To Copy Rows

- Svaki INSERT iskaz dodaje samo jedan red u tabeli
- Ali ako treba kopirati 100 redova iz jedne tabele u drugu to se izvodi preko podupita unutar INSERT iskaza
- Svi rezultati iz podupita su uneti u tabelu
- Kako se vidi u primeru, nova tabela zvana SALES_REPS je popunjena sa kopijama nekih redova i kolona iz tabele EMPLOYEES
- WHERE iskaz selektuje one zaposlene koji imaju job_id kao '%REP%'

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
    SELECT employee_id, last_name, salary, commission_pct
    FROM   employees
    WHERE  job_id LIKE '%REP%';
```

- Da bi startovali iznad INSERT iskaza, sales_reps tabela treba da se prvo napravi
- Broj kolona i njihovi tipovi podataka u listi kolona u INSERT iskazu moraju odgovarati broju kolona i njihovim tipovima podataka u podupitu
- Podupit nije zatvoren u zagradama kao što se radi u podupitima u WHERE iskazu kao delu SELECT iskaza
- Ako želimo kopirati sve podatke, sve redove i sve kolone, sintaksa je još jednostavnija
- Izaberi redove iz EMPLOYEES tabele i unesi ih u SALES_REPS tabelu, iskaz bi bio napisan:

```
INSERT INTO sales_reps
   SELECT *
   FROM   employees;
```

- Ovo će raditi samo ako obe tabele imaju isti broj kolona sa odgovarajućim tipovima podataka i po istom redosledu

12-2 Updating Column Values and Deleting Rows

- Treba objasniti kako FK i PK integritet ograničava efekte UPDATE i DELETE iskaza
- Posao DBA je updejtovanje, insertovanje, brisanje i upravljanje podataka

UPDATE

- Iskaz UPDATE se koristi za modifikovanje postojećih redova u tabeli
- UPDATE zahteva četiri vrednosti:
    - ime tabele

- ime kolona čije vrednosti će biti modifikovane
- nova vrednost za svake od kolona se modifikuje
- uslov koji identifikuje koji redovi u tabeli će biti modifikovani

- Nova vrednost za kolonu može biti rezultat single-row podupita
- Koriste se samo copy_tablename tabele u svim DML transakcijama
- Primer koristi UPDATE iskaze za promenu telefonskog broja jednog zaposlenog u employees tabeli

```
UPDATE copy_employees
SET phone_number = '123456'
WHERE employee_id = 303;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	PHONE_NUMBER
303	Angelina	Wright	123456

- Mogu se promeniti nekoliko kolona i/ili nekoliko redova u jednom UPDATE iskazu
- Ovaj primer menja i telefonski broj i prezime za dva zaposlena

```
UPDATE copy_employees
SET phone_number = '654321', last_name = 'Jones'
WHERE employee_id >= 303;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	PHONE_NUMBER
303	Angelina	Jones	654321
304	Test	Jones	654321

- Pri updejtovanju više kolona one se odvajaju zarezima
- Ako se izostavi WHERE, svaki red u tabeli će biti updejtovan a to možda i nije bio cilj

```
UPDATE copy_employees
SET phone_number = '654321', last_name = 'Jones';
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	PHONE_NUMBER
100	Steven	Jones	654321
101	Neena	Jones	654321
102	Lex	Jones	654321
200	Jennifer	Jones	654321
205	Shelley	Jones	654321
206	William	Jones	654321
149	Eleni	Jones	654321
174	Ellen	Jones	654321
...	...	...	...

Updating a Column with a value from a Subquery

- Može se koristiti rezultat iz single-row subquery za pravljenje nove vrednosti za updejtovanu kolonu

```
UPDATE copy_employees
SET salary = (SELECT salary
              FROM copy_employees
             WHERE employee_id = 100)
WHERE employee_id = 101;
```

- This example changes the salary of one employee (id = 101) to the same salary as another employee (id = 100).

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
100	Steven	King	24000
101	Neena	Kochhar	24000

```
UPDATE copy_employees
SET salary = (SELECT salary
              FROM copy_employees
             WHERE employee_id = 100)
WHERE employee_id = 101;
```

- As usual, the subquery executes first and retrieves the salary for employee id = 100.
- This salary value is then used to update the salary for employee id = 101.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
100	Steven	King	24000
101	Neena	Kochhar	24000

Updating Two Columns with Two Subquery Statements

- Za updejtovanje nekoliko kolona u jednom UPDATE iskazu, moguće je pisati više single-row podupita, jedan za svaku kolonu
- U sledećem primeru UPDATE iskaz menja platu i job id jednog zaposlenog (id = 206) na istu vrednost kao drugi zaposleni (id = 205)

```
UPDATE copy_employees
SET salary = (SELECT salary
              FROM copy_employees
              WHERE employee_id = 205),
    job_id = (SELECT job_id
              FROM copy_employees
              WHERE employee_id = 205)
WHERE employee_id = 206;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	JOB_ID
205	Shelley	Higgins	12000	AC_MGR
206	William	Gietz	12000	AC_MGR

Updating Rows Based On Another Table

- Kao što se i očekuje, podupit može vratiti informaciju iz jedne tabele koja se onda koristi za updejt druge tabele
- U ovom primeru, kopija tabele employees je kreirana
- Onda se podaci iz originalne tabele employees vrate, kopiraju i koriste za popunjavanje copy_employees tabele

```
UPDATE  copy_employees
SET salary  = (SELECT salary
               FROM employees
               WHERE employee_id = 205)
WHERE employee_id  = 202;
```

Updating Rows Using Correlated Subquery

- Podupiti ili su samostalni ili su u korelaciji
- Kod korelacionih podupita, updejtuje se red u tabeli na osnovu select iz iste tabele
- U sledećem primeru, kopija imena kolone sektora je kreirana u tabeli employees
- Zatim se vratili podaci iz originalne tabele departments, kopirani i korišćeni za popunu kopije kolone u tabeli employees

```
ALTER TABLE  copy_employees
ADD (department_name varchar2(30) NOT NULL);
```

```
UPDATE  copy_employees e
SET e.department_name  = (SELECT d.department_name
                          FROM departments d
                          WHERE e.department_id = d.department_id);
```

DELETE

- Iskaz DELETE se koristi za odstranjivanje postojećih redova u tabeli
- Iskaz zahteva dve vrednosti: ime tabele i uslov koji identifikuje kolone koje će se obrisati
- Primer koristi kopiju tabele employee za brisanje jednog reda – zaposlenog sa ID brojem 303

```
DELETE from copy_employees
WHERE employee_id = 303;
```

- Šta misliš da će biti obrisano ako WHERE iskaz je eliminisan u DELETE iskazu ?
- Svi redovi u tabeli su obrisani ako se izostavi WHERE iskaz

Subquery DELETE

- Podupiti se takođe mogu koristiti u DELETE iskazima
- Primer pokazuje obrisane redove iz tabele employees za sve zaposlene koji rade u Shipping sektoru; možda je ovaj sektor preimnovan ili zatvoren

```
DELETE FROM copy_employees
WHERE  department_id =
   (SELECT department_id
    FROM   departments
    WHERE  department_name = 'Shipping');
```

5 row(s) deleted

- Sledeći primer briše redove svih zaposlenih koji rade za menadžera koji radi sa manje od dva zaposlena

```
DELETE FROM copy_employees e
WHERE  e.manager_id IN
        (SELECT d.manager_id
         FROM employees d
         HAVING count (d.department_id) < 2
         GROUP BY d.manager_id);
```

Integrity Constraint Errors

- Ograničenja integriteta obezbeđuju da podatak odgovara potrebnim setovima pravila
- Ograničenja se automatski proveravaju kada god DML iskaz, koji može se suprotstaviti pravilima, se izvrši
- Ako se bilo koje pravilo prekrši, tabela se ne updejtuje i greška se vraća
- Sledeći primer krši NOT NULL ograničenje pošto last_name ima not null ograničenje i id=999 ne postoji, tako da podupit vraća null rezultat

```
UPDATE copy_employees
SET last_name = (SELECT last_name
                 FROM copy_employees
                 WHERE employee_id = 999)
WHERE employee_id = 101;
```

```
ORA-01407: cannot update
("US_A009EMEA815_PLSQL_T01"."COPY_EMPLOYEES"."LAST_NAME") to NULL
```

- Kada će PK – FK ograničenje biti proveravano ?
- Tabela EMPLOYEES ima FK ograničenje na department_id koje referencira department_id od DEPARTMENTS tabele
- Ovo obezbeđuje da svaki zaposleni pripada validnom sektoru
- U DEPARTMENTS tabeli, department_id 10 i 20 postoje ali 15 ne postoji
- Koji od iskaza vraća grešku ?

```
1.      UPDATE employees SET department_id = 15
        WHERE employee_id = 100;

2.      DELETE FROM departments WHERE department_id = 10;

3.      UPDATE employees SET department_id = 10
        WHERE     department_id = 20;
```

- Upit 1 će vratiti grešku pošto department_id ne postoji
- Upit 2 će vratiti grešku ako postoje zaposleni sa department_id 10
- Upit 3 će uspešno pretpostaviti da department_id 10 je u tabeli departments
- Pri modifikaciji kopija tabele (npr copy_employees) mogu se videti greške not null ograničenja, ali nećeš videti bilo koje greške PK-FK ograničenja
- Razlog za to je što CREATE TABLE...AS (SELECT...) iskaz koji se koristi za kreiranje kopije tabele, kopira i redove i not null ograničenja, ali ne kopira PK-FK ograničenja
- Zato, nijedno PK-FK ograničenje ne postoji na nilo kojoj kopiranoj tabeli

FOR UPDATE Clause in a SELECT Statement

- Kada SELECT iskaz se napravi za db tabelu, nikakav locks su izdati u db na redove vraćene od strane upita koji se izdaje
- Najviše vremena ovo je kako se želi da se db ponaša – držanje broja locks sveden na minimum
- Ipak ponekad, želimo da budemo sigurni da niko drugi ne može updejtovati ili brisati zapise koje vraća tvoj upit dok ti radiš na ovim zapisima
- Ovo je kada FOR UPDATE iskaz se koristi
- Čim se izvrši upit, db će automatski izdati ekskluzivan row-level locks na sve redove vraćene od strane tvoje SELECT iskaza, koji će biti zadržan dok ti ne izdaš COMMIT ili ROLLBACK komandu
- On-line/hosted instanca APEX-a će autocommit a row-level lock se neće desiti
- Ako se koristi FOR UPDATE izraz u SELECT iskazu sa više tabela u njoj, svi redovi iz svih tabela će biti zaključana

```
SELECT e.employee_id, e.salary, d.department_name
FROM employees e JOIN departments d  USING (department_id)
WHERE job_id = 'ST_CLERK'  AND location_id = 1500
  FOR UPDATE
ORDER BY e.employee_id;
```

- Ova četiri reda su sada zatvorena od strane korisnika koji startuju SELECT iskaz sve dok korisnik ne izda COMMIT ili ROLLBACK

EMPLOYEE_ID	SALARY	DEPARTMENT_NAME
141	3500	Shipping
142	3100	Shipping
143	2600	Shipping
144	2500	Shipping

12-3 DEFAULT Values, MERGE, and Multi-Table Inserts

- Data Werehouse je kolekcija podataka dizajnirana za podržavanje donošenje odluka u biznis-menadžmentu. Data werehouse sadrži širok spektar podataka, kao što su datumi prodaje, podaci o mušterijama, payroll, knjigovodstvo, lični podaci, i sve to daje koherentnu sliku biznis uslova u jednom momentu
- Sada se radi o efikasnim metodama za updejt i unos podataka korišćenjem sekvence uslovnih INSERT i UPDATE komandi u jednom atomskom iskazu
- Takođe se vidi kako se dobija podatak iz jednog podupita i INSERT za redove vraćene u više od jedne ciljne tabele

DEFAULT

- Svaka kolona u tabeli može imati difolt vrednost specificiranu za nju
- U slučaju da novi red je unet a nikakva vrednost nije dodeljena za kolonu, difolt vrednost će biti dodeljena umesto null vrednosti
- Korišćenje difolt vrednosti omogućava kontrolu gde i kada će difolt vrednost biti primenjena
- Difolt vrednost može biti literal vrednost, izraz ili SQL funkcija kao što su SYSDATE i USER, ali vrednost ne može biti ime druge kolone
- Difolt vrednost mora odgovarati tipu podataka kolone
- DEFAULT može biti specificiran za kolonu kada tabela je kreirana ili izmenjena
- Primer ispod pokazuje defoult vrednost koja je specificirana za hire_date kolonu u trenu kada je tabela kreirana:

```
CREATE TABLE my_employees (
       hire_date      DATE DEFAULT SYSDATE,
       first_name     VARCHAR2(15),
       last_name      VARCHAR2(15));
```

- Kada su redovi dodati u ovu tabelu, SYSDATE će biti dodeljen na bilo koji red koji eksplicitno ne specificira hire_date vrednost

Explicit DEFAULT with INSERT

- Eksplicitni difolt se može koristiti u INSERT i UPDATE iskazima
- INSERT primer koristi my_employees tabelu i prikazuje eksplicitno korišćenje DEFAULT:

```
INSERT INTO my_employees
   (hire_date, first_name, last_name)
VALUES
   (DEFAULT, 'Angelina','Wright');
```

- Implicitno korišćenje DEFAULT

```
INSERT INTO my_employees
   (first_name, last_name)
VALUES
   ('Angelina','Wright');
```

- Ako je difolt vrednost specificirana za hire_date kolonu kada je tabela kreirana, Oracle dodeljuje difolt vrednost koloni. Ako ne postoji specificirana difolt vrednost, Oracle dodeljuje null vrednost

Explicit DEFAULT with UPDATE

- Eksplicitni difolt mogu biti korišćeni u INSERT i UPDATE iskazima
- UPDATE primer korišćenjem my_employees tabele pokazuje eksplicitno korišćenje DEFAULT

```
UPDATE my_employees
SET hire_date = DEFAULT
WHERE last_name = 'Wright';
```

- Ako difolt vrednost je specificirana za hire_date kolonu, kolona je dodeljena defoult vrednost
- Ipak, ako nijedna difolt vrednost nije specificirana kada je kolona kreirana, null vrednost je dodeljena

MERGE

- Korišćenje MERGE iskaza izvršava dva zadatka istovremeno. MERGE će INSERT i UPDATE simultano. Ako vrednost ne postoji, nova se ubacuje
- Ako vrednost postoji ali treba da se promeni, MERGE će je updejtovati
- Za izvršenje ovih tipova promena na db tabelama, mora postojati INSERT i UPDATE privilegija na ciljnoj tabeli i SELECT privilegija na izvornoj tabeli
- Alijasi se mogu koristiti sa MERGE iskazom

MERGE sintaksa

- Red po red se čita iz izvorne tabele i upoređuje sa redovima u ciljnoj tabeli korišćenjem uslova upoređivanja
- Ako upređivani red postoji u ciljnoj tabeli, izvorni red se koristi za updejt jedne ili više kolona u odgovarajućem ciljnom redu
- Ako upoređen red ne postoji, vrednosti iz izvornog reda se koriste za ubacivanje novog reda u ciljnu tabelu

```
MERGE INTO destination-table USING source-table
  ON matching-condition
WHEN MATCHED THEN UPDATE
     SET ……
WHEN NOT MATCHED THEN INSERT
     VALUES (……);
```

- Primer koristi EMPLOYEES tabelu (alijas e) kao izvor podataka za insertovanje i updejt kolona u kopiju tabele sa imenom COPY_EMP (alijas c)

```
MERGE INTO copy_emp c  USING employees e
  ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN UPDATE
     SET
         c.last_name    = e.last_name,
         c.department_id   = e.department_id
WHEN NOT MATCHED THEN INSERT
     VALUES   (e.employee_id, e.last_name, e.department_id);
```

```
create table copy_emp as (Select * from employees where department_id > 100);

select * from copy_emp;

update copy_emp
set last_name = 'Cotton'
where employee_id = 205;

select * from copy_emp;

MERGE INTO copy_emp c  USING employees e
  ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN UPDATE
    SET
       c.last_name   = e.last_name,
       c.department_id   = e.department_id
WHEN NOT MATCHED THEN INSERT
    VALUES   (e.employee_id, e.first_name, e.last_name, e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id, e.department_id, e.bonus);

select * from copy_emp;

drop table copy_emp;
```

- EMPLOYEES redovi 100 i 103 imaju odgovarajuće redove u COPY_EMP i tako su COPY_EMP redovi updejtovani
- EMPLOYEE 142 nema odgovarajući red, tako da je insertovan u COPY_EMP

EMPLOYEES (source table)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
103	Hunold	60
142	Davies	50

EMPLOYEES (source table)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
103	Hunold	60
142	Davies	50

COPY_EMP before the MERGE is executed

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	Smith	40
103	Chang	30

COPY_EMP after the MERGE has executed

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
103	Hunold	60
142	Davies	50

Multi-Table Inserts

- Multio-table inserts se koriste kada isti izvorni podatak treba uneti u više od jedne ciljne tabele
- Ova funkcionalnost je korisna kada se radi u data warehouse okruženju gde je često pomerati podatak iz OS u data warehouse za analitički izveštaj i analizu
- Kreiranje i upravljanje DW je jedan način upravljanja ponekad veoma visokog broja redova unetih u OS tokom normalnog radnog dana

- Koliko redova podataka telefon provajder mora kreirati dnevno za sve vaše pozive ili teks poruke napravljene na svim uređajima kojima imamo pristup ?
- Ovi redovi možda moraju biti uneti u više od jedne tabele u DW, pa ako možemo samo SELECT njih jednom a zatim ih ponoviti, to će poboljšati performanse
- Multi-table inserts može biti bezuslovan ili uslovan. U jednom bezuslovnom multi-table insert, Oracle će uneti sve redove vraćene od strane podupita u sve tabele unete iskaze proinađene u iskazu
- U uslovnom multi-table inster može se specificirati ili ALL ili FIRST
- Specifikacija ALL:
    - ako specificiramo ALL, difolt vrednost, db evaluira svaki WHEN iskaz bez obzira na reztultat evaluacije bilo koje drugačije WHEN iskaza
    - Za svaki WHEN iskaz čiji uslov evaluira na true, db izvršava odgovarajući INTO iskaz listu
- Specificiranje FIRST:
    - ako se specificira FIRST, db evaluira svaki WHEN iskaz po redosledu u kojem se i pojavljuje u iskazu
    - Za prvi WHEN iskaz koji evaluira na true, db izvršava odgovarajući INTO iskaz i preskače subsequent WHEN iskaze za dati red
- Specificirati ELSE iskaz:
    - za dati red, ako nema WHEN iskaz da evaluira na true, db izvršava INTO iskaz listu dodeljenu sa ELSE iskazom
    - Ako ti nisi specificirao else iskaz, onds db ne uzima nijednu akciju za taj red

Multi-Table Inserts Syntax

- Multi-table insert iskaz sintaksa:

```
INSERT ALL INTO clause VALUES clause SUBQUERY
```

- Multi-table insert statement example is as follows:

```
INSERT ALL
   INTO my_employees
      VALUES (hire_date, first_name, last_name)
   INTO copy_my_employees
      VALUES (hire_date, first_name, last_name)
SELECT hire_date, first_name, last_name
FROM employees;
```

This multi-table insert, will retrieve the rows from the SELECT statement, and INSERT the rows into both the my_employees and copy_my_employees table.

INSERT ALL requires a SELECT statement in order to run. We can, however, use the dummy table DUAL to perform multiple INSERT statements into a single table – for example:

```
INSERT ALL
  INTO copy_employees
    (employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary)
  VALUES
    (304,'James','Parrot', 'jparrot', '912-699-4656', '15/Jul/2015', 'IT_PROG',4200)
  INTO copy_employees
    (employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary)
  VALUES
    (305,'Rebecca','Graham', 'rgraham', '480-678-4259', '15/Jun/2015', 'IT_PROG',4200)
SELECT * FROM dual;
```

Multi-Table Inserts Conditional

```
INSERT ALL
   WHEN call_ format IN ('tlk','txt','pic') THEN
   INTO all_calls
      VALUES (caller_id, call_timestamp, call_duration, call_format)
   WHEN call_ format IN ('tlk','txt') THEN
   INTO police_record_calls
      VALUES (caller_id, call_timestamp, recipient_caller)
   WHEN call_duration < 50 AND call_type = 'tlk' THEN
   INTO short_calls
      VALUES (caller_id, call_timestamp, call_duration)
   WHEN call_duration > = 50 AND call_type = 'tlk' THEN
   INTO long_calls
      VALUES (caller_id, call_timestamp, call_duration)
SELECT  caller_id, call_timestamp, call_duration, call_format,
        recipient_caller
FROM calls
WHERE TRUNC(call_timestamp ) = TRUNC(SYSDATE);
```

- Multi-table insert koristi WHEN uslov. Ako red iz SELECT iskaza odgovara kriterijumu WHEN uslova, red se dodaje na tabelu u INTO izraz
- Prethodni primer se koristi za demonstraciju stvarnog korišćenja za multi-table inserts, ali ako tabele ne postoje ovaj upit neće raditi na APEX