Podatkovne baze

Delovna skripta za laboratorijske vaje

Bojan Klemenc, Lovro Šubelj, Aljaž Zrnec in Marko Bajec
Fakulteta za računalništvo in informatiko

22. april 2010

Delovna skripta za laboratorijske vaje pri predmetih OPB in PB1 na univerzitetnem študiju na Fakulteti za računalništvo in informatiko. Skripta je še v procesu nastajanja, zato določena poglavja manjkajo, obstoječa poglavja pa se lahko še spreminjajo.

Kazalo

Poglavje 1

Poizvedbe v SQL-u

SQL (Structured Query Language) - strukturirani povpraševalni jezik

1.1 Osnove SQL-a:

Oblika klasičnega SQL-stavka:

```
1 SELECT A1, A2, ... Ak
2   FROM T1, T2, ... Tn
3  WHERE P;
```

Zapis iste poizvedbe v relacijski algebri:

$$\pi_{A_1,A_2 \ldots A_k}(\sigma_P(T_1 \times T_2 \times \cdots \times T_n))$$

1.1.1 Enostavni primeri

Projekcija

Projekcija po atributih $A_1, A_2 \ldots A_k$ nad tabelo T v relacijski algebri:

$$\pi_{A_1,A_2 \ldots A_k}(T))$$

Ustrezen SQL-stavek:

```
1 SELECT A1, A2, ... An
2   FROM T;
```

Selekcija

Selekcija pod pogojem P nad tabelo T in projekcijo po atributih $A_1, A_2 \dots A_k$ v relacijski algebri:

$$\pi_{A_1,A_2 \dots A_k}(\sigma_P(T))$$

Ustrezen SQL-stavek:

```
1 SELECT A1, A2, ... An
2   FROM T
3  WHERE P;
```

Selekcija brez projekcije

Selekcija pod pogojem P nad tabelo T brez projekcije v relacijski algebri:

$$\sigma_P(T)$$

Ustrezen SQL-stavek:

```
1 SELECT *
2   FROM T
3  WHERE P;
```

Vsi stolpci iz tabele T pod pogojem P.

Stik

Stik je kartezični produkt pod določenim pogojem.
Primer: Imamo relaciji r in s; pripadajoči relacijski shemi sta R(A, B, C) in S(C, D). Napravimo stik po skupnem atributu C:

$$\pi_{A,B,D}(\sigma_{r.C=s.C}(r \times s))$$

ali

$$\pi_{A,B,D}(r \underset{r.C=s.C}{\bowtie} s)$$

Ustrezen SQL stavek:

```
1 SELECT A, B, D
2   FROM r, s
3  WHERE r.C = s.C;
```

1.1.2 Podvajanje vrednosti

SQL privzeto v rezultatih poizvedb dopušča podvajanje vrstic. Z DISTINCT izločimo vse podvojitve.

```
SELECT DISTINCT A1, A2, ... Ak
  FROM T1, T2, ... Tn
 WHERE P;
```

1.1.3 Preimenovanje in sinonimi (aliasi)

Kadar želimo preimenovati stolpce uporabimo AS "novo_ime":

```
SELECT ime AS "ime stranke"
  FROM stranka;
```

AS lahko tudi izpustimo.

Kadar se pogosto sklicujemo na imena tabel, jim lahko določimo krajše sinonime. Sinonimi so obvezni, kadar se ena tabela večkrat pojavi v istem stavku.

Namesto:

```
SELECT stranka.ime, agent.ime
  FROM stranka, narocilo, agent
 WHERE stranka.sid = narocilo.sid AND agent.aid = narocilo.aid;
```

lahko zapišemo:

```
SELECT s.ime, a.ime
  FROM stranka s, narocilo n, agent a
 WHERE s.sid = n.sid AND a.aid = n.aid;
```

1.1.4 Stik nizov (konkatenacija)

Stik nizov zapišemo z dvema navpičnima črtama ||.
Primer: Izpisati želimo imena in priimke strank kot niz oblike 'ime priimek'.

```
SELECT ime || ' ' || priimek AS "ime in priimek stranke"
  FROM stranka;
```

1.1.5 Aritmetični izrazi

Lahko napišemo tudi aritmetične izraze (z operatorji $* / + -$).
Primer: Za vsako naročilo želimo izpisati zmnožek količine in cene na kos.

```
1 SELECT kolicina * cena_na_kos
2   FROM narocilo n, izdelek i
3  WHERE n.iid = i.iid;
```

1.1.6 Uporaba spremenljivk v SQL-stavkih (Oracle)

Spremenljivko definiramo na naslednji način: &ime_spremenljivke.
Primer: Preberemo ime kraja in izpišemo stranke, ki prihajajo iz tistega kraja.

```
1 SELECT sid
2   FROM stranka
3  WHERE mesto = '&ime_kraja';
```

1.1.7 Komentarji

Enovrstični komentar: −− (dva minusa)
Večvrstični komentar: /* */

Primer:

```
1 SELECT ime                -- enovrstični komentar
2   FROM stranka            /* večvrstični
3                              komentar      */
4  WHERE mesto = 'Ljubljana';
```

1.1.8 Pisanje SQL-stavkov

- Besede v SQL-stavku niso občutljive na razliko med velikimi in malimi črkami (sami podatki pa so – nizi v enojnih navednicah).
- Stavek lahko zapišemo v eni ali več vrsticah.
- Ključnih besed ne moremo skrajšati.
- Zaradi boljše berljivosti se posamezni del stavka zapiše v svojo vrstico.
- Ključne besede lahko zaradi berljivosti pišemo z velikimi črkami.
- Ugnezdene stavke zaradi berljivosti zamaknemo desno.

1.2 Urejanje

Za razvrščanje vrstic po vrstnem redu (abecednem redu) se uporablja ORDER BY. Privzeto se razvršča naraščajoče (ASC).

Razvrščanje po padajočem vrstnem redu (DESC):

```
1   SELECT ime_stolpca...
2     FROM ime_tabele
3    WHERE pogoj
4 ORDER BY ime_stolpca DESC;
```

Razvrščamo lahko tudi po več stolpcih:

```
1 ORDER BY ime_stolpca1, ime_stolpca2 ...
```

1.3 Primerjanje

Primerjalni operatorji so podani v preglednici 1.1.

operator	pomen
=	je enako
! = ali <>	ni enako, različno
>	večje
>=	večje ali enako
<	manjše
<=	manjše ali enako
IN	vsebovan v množici ($\in$)
BETWEEN . . . AND . . .	med dvema vrednostima, vključno z mejnima vrednostima
LIKE	niz ustreza vzorcu
IS NULL	neznano, nedoločeno
NOT	negacija (NOT IN, NOT BETWEEN, NOT LIKE, NOT NULL)
AND	in
OR	ali

Preglednica 1.1: Primerjalni operatorji in njihov pomen.

1.3.1 IN

Izraža vsebovanost v določeni množici (množica je lahko tudi rezultat poizvedbe).

Primer: Iščemo imena strank, ki prihajajo iz Ljubljane, Trsta ali Celovca.

```
1 SELECT ime
2   FROM stranka
3  WHERE mesto IN ('Ljubljana', 'Trst', 'Celovec');
```

1.3.2 LIKE

Iskanje z vzorci:
% predstavlja nič ali več znakov
_ predstavlja en znak

Primer: Iščemo vsa imena strank, ki se začnejo na M in so dolga vsaj 4 črke.

```
1 SELECT ime
2   FROM stranka
3  WHERE ime LIKE 'M___%';
```

1.3.3 IS NULL

Praznih vrednosti ne moremo enačiti, zato uporabimo NOT NULL.
Primer: Iščemo imena zaposlencev, ki nimajo vzdevka.

```
1 SELECT ime
2   FROM zaposlenci
3  WHERE vzdevek IS NULL;
```

1.3.4 Vrstni red operacij

1. aritmetični operatorji
2. stik nizov (konkatenacija)
3. primerjalni operatorji
4. IS [NOT] NULL, LIKE [NOT] IN
5. [NOT] BETWEEN
6. NOT
7. AND
8. OR

Sprememba vrstnega reda je možna z uporabo oklepajev.

1.4 Stiki

Sintaksa za kartezični produkt in različne vrste stikov je podana v preglednici 1.2.

sintaksa	pomen
CROSS JOIN	kartezični produkt ($\times$)
NATURAL JOIN	naravni stik ($\bowtie$)
JOIN USING (ime_stolpca)	stik po določenem istoimenskem stolpcu (oz. po stolpcih)
JOIN ON (pogoj)	stik pod pogojem ($\bowtie_P$)
LEFT OUTER JOIN	levi zunanji stik ($\bowtie$)
RIGHT OUTER JOIN	desni zunanji stik ($\bowtie$)
FULL OUTER JOIN	popolni zunanji stik ($\bowtie$)

Preglednica 1.2: Vrste stikov in njihova sintaksa v SQL-u.

1.4.1 Kartezični produkt

Kartezični produkt lahko zapišemo z naštevanjem tabel v FROM delu stavka.
Primer:

```
1 SELECT *
2   FROM r, s;
```

Lahko pa uporabimo ključno besedo CROSS JOIN.

```
1 SELECT *
2   FROM r CROSS JOIN s;
```

1.4.2 Stik

Želimo poiskati imena vseh strank, ki so že kadarkoli kaj kupile.

```
1 SELECT ime
2   FROM stranka s, narocilo n
3  WHERE s.sid = n.sid;
```

Pogoj za stik navedemo v WHERE delu stavka.

JOIN ON

Poizvedbo lahko z uporabo JOIN ON napišemo tudi drugače:

```
1 SELECT ime
2   FROM stranka s JOIN narocilo n ON (s.sid = n.sid);
```

V ON lahko opredelimo različne pogoje za povezovanje tabel.

NATURAL JOIN

Ker gre za naravni stik, lahko uporabimo tudi NATURAL JOIN:

```
1 SELECT ime
2   FROM stranka NATURAL JOIN narocilo;
```

JOIN USING

Lahko pa napišemo po katerem istoimenskem atributu naj se vrši stik (JOIN USING):

```
1 SELECT ime
2   FROM stranka JOIN narocilo USING (sid);
```

Uporabno predvsem, če bi imeli več istoimenskih stolpcev v tabelah, po katerih ne želimo stika. Pri USING ime povezovalnega stolpca pred seboj ne sme imeti imena tabele.

1.5 Množice

Operatorji za delo z množicami:

- UNION - vse vrstice iz vseh množic (brez podvojenih vrstic)
- INTERSECT - presek množic
- MINUS - vrstice, ki so samo v prvi množici
- UNION ALL - vse vrstice iz vseh množic (s podvojenimi vrsticami)

Sintaksa za uporabo zgoraj naštetih operatorjev:

```
1 SELECT ...
2 {operator}
3 SELECT ...;
```

Pomembna je združljivost stolpcev. Število stolpcev mora biti enako, istoležni stolpci morajo biti istega tipa.

Še dodatni operatorji:

- IN, NOT IN (tabela): pripadnost $\in$ in $\notin$ (glej 1.3.1)
- ALL, ANY: kvantifikatorja $\forall$ in $\exists$
- EXISTS, NOT EXISTS (tabela): nepraznost in praznost

1.5.1 ANY in ALL

Prav tako kot IN, tudi ALL in ANY uporabljamo pri večvrstičnih podpoizvedbah. To pomeni, da primerjajo določeno vrednost z vsemi vrednostmi iz podpoizvedbe. Podpoizvedba lahko vrača le podatke iz enega stolpca.

Primeri za ANY:

`...WHERE x < ANY (SELECT y...);`	$\exists y : x < y$	manj kot maksimum
`...WHERE x = ANY (SELECT y...);`	$x \in \{y \mid \ldots\}$	vsebovanost (IN)
`...WHERE x <> ANY (SELECT y...);`	$\exists y : x \neq y$	

Primeri za ALL:

`...WHERE x < ALL (SELECT y...);`	$\forall y : x < y$	manj kot minimum
`...WHERE x > ALL (SELECT y...);`	$\forall y : x > y$	več kot maksimum
`...WHERE x <> ALL (SELECT y...);`	$\forall y : x \neq y$	element ni v množici (NOT IN)

1.5.2 EXISTS in NOT EXISTS

Operator EXISTS testira obstoj vrstic v rezultatu ugnezdene poizvedbe. Če se v rezultatu ugnezdene poizvedbe nahaja vsaj ena vrstica, je logična vrednost operatorja EXIST "true", v nasprotnem primeru pa je "false". Takoj ko EXIST v ugnezdeni poizvedbi najde vrstico, se izvajanje ugnezdene poizvedbe prekine. Število stolpcev v podpoizvedbi ni pomembno.

Primer: Iščemo stranke, ki še niso nikoli nič naročile.

```
1 SELECT ime
2   FROM stranka s
3  WHERE NOT EXISTS (SELECT *
4                      FROM narocilo
5                     WHERE sid = s.sid);
```

Ali je izbira vseh stolpcev sploh potrebna?

Z množicami operatorjem NOT EXIST lahko na enostaven način izvedemo operacijo deljenja.

1.6 Skupinjenje in operacije nad skupinami

1.6.1 Operacije nad skupinami

Operacije nad skupinami za posamezno skupino izračunajo en rezultat. Tudi množica vseh vrstic je skupina. V preglednici 3.1 so podane funkcije, ki jih lahko uporabimo.

funkcija	pomen
AVG	povprečje
COUNT	prešteje število vrednosti v skupini
MAX	največja vrednost; deluje tudi nad datumi in nizi
MIN	najmanjša vrednost; deluje tudi nad datumi in nizi
SUM	vsota
STDDEV	standardni odklon
VARIANCE	varianca

Preglednica 1.3: Funkcije, ki jih lahko uporabimo nad posamezno skupino.

Nobena operacija ne upošteva praznih elementov (NULL); izjema je COUNT(*).
Primer: Koliko izdelkov smo prodali?

```
1 SELECT COUNT(*)
2   FROM narocilo;
```

Primer: Koliko različnih vrst izdelkov smo prodali?

```
1 SELECT COUNT(DISTINCT iid)
2   FROM narocilo;
```

1.6.2 GROUP BY

Skupinjenje vršimo z GROUP BY.
Primer: Za posamezno smer želimo izpisati vsoto plačanih šolnin.

```
1   SELECT smer, SUM(solnina_placana)
2     FROM student
3 GROUP BY smer;
```

Stolpca, po katerem se izvaja skupinjenje, ni potrebno navesti v SELECT delu stavka.

1.6.3 HAVING

Za omejevanje skupin uporabimo HAVING.
Primer: Za posamezno smer želimo izpisati vsoto plačanih šolnin, vendar le če je vsota pri posamezni smeri večja od 2500.

```
1   SELECT smer, SUM (solnina_placana)
2     FROM student
3 GROUP BY smer
4   HAVING SUM (solnina_placana) > 2500;
```

Vrstni red je pomemben: GROUP BY, HAVING, ORDER BY.

Razširjeni SELECT stavek torej zgleda takole:

```
1   SELECT [DISTINCT] ime_stolpca ...
2     FROM ime_table ...
3    WHERE pogoj ...
4 GROUP BY ime_stolpca ...
5   HAVING pogoj ...
6 ORDER BY ime_stolpca ...;
```

Poglavje 2

SQL-ukazi

SQL-ukaze delimo v naslednje skupine:

- DDL (Data Definition Language) - jezik za opredeljevanje podatkov (glej 2.1)
- DCL (Data Control Language) - jezik za nadzor nad podatki (glej 2.2)
- DML (Data Manipulation Language) - jezik za rokovanje s podatki (glej 2.3)
- TCL (Transaction Control Language) - jezik za nadzor transakcij (glej 2.4)

2.1 DDL

Jezik za opredeljevanje podatkov zajema med drugim naslednje ukaze:

- CREATE (ustvarjanje tabele)
- ALTER (razširjanje tabele s stolpci, spreminjanje stolpca)
- DROP (brisanje tabele, stolpca)

2.1.1 CREATE

Ustvarjanje tabele:

```
1 CREATE TABLE ime_tabele
2 (ime_stolpca1 podatkovniTip()    NULL|NOT NULL,
3 ...
4  ime_stolpcaN podatkovniTip()    NULL|NOT NULL);
```

Namesto TABLE lahko uporabimo: USER, VIEW,...
Več o podatkovnih tipih v 3.1.

2.1.2 ALTER

Dodajanje stolpca:

```
1 ALTER TABLE ime_tabele
2 ADD (ime_stolpca podatkovniTip());
```

Če tabela že vsebuje vrstice, potem dodani stolpec privzeto dovoljuje NULL vrednosti.

Spreminjanje stolpca:

```
1 ALTER TABLE ime_tabele
2 MODIFY (ime_stolpca podatkovniTip());
```

2.1.3 DROP

Brisanje stolpca:

```
1 ALTER TABLE ime_tabele
2 DROP COLUMN ime_stolpca;
```

- Brišemo lahko posamezen stolpec (največ enega naenkrat).
- Stolpec lahko vsebuje podatke ali pa ne.
- V tabeli mora po brisanju ostati vsaj en stolpec.
- Ko se stolpec zbriše, ga ni več mogoče obnoviti.

Brisanje celotne tabele:

```
1 DROP TABLE ime_tabele;
```

Namesto TABLE lahko uporabimo tudi USER ali VIEW.

Koristno je tudi:

```
1 CREATE TABLE ime_tabele AS SELECT.....
```

Integritetne omejitve se ne prenesejo na novo tabelo. Če v ugnezdeni poizvedbi atribut množimo z neko vrednostjo, je potrebno uporabljati aliase na stolpce.

2.2 DCL

- GRANT (podelitev pravice)
- REVOKE (odvzem pravice)

2.2.1 GRANT in REVOKE

Primer za GRANT:

```
1 GRANT ime_vloge TO uporabnik;
```

Primer za REVOKE:

```
1 REVOKE privilegiji... ON
2         ime_objekta FROM uporabnik;
```

2.2.2 O pravicah

Varnost je zagotovljena:

- z uporabniškimi imeni in gesli (kontrola dostopa do ORACLA)
- z vlogami (nadzor dostopa na nivoju PB)
- s privilegiji (za nadzor dostopa na nivoju tabel)
- s sinonimi (za nadzor dostopa na nivoju tabel)

Uporabniška imena

DBA (Database Administrator) ustvari nove uporabnike z SQL-ukazom:

```
1 CREATE USER uporabnik
2 IDENTIFIED BY geslo;
```

Briše pa jih z:

```
1 DROP USER uporabnik;
```

Geslo se zamenja z:

```
1 ALTER USER uporabnik
2 IDENTIFIED BY novo_geslo;
```

Vloge

Oracle ob namestitvi baze zagotavlja 3 privzete vloge:

- DBA
- RESOURCE
- CONNECT

Vloga CONNECT dovoljuje uporabniku:

- povezavo s podatkovno bazo
- manipulacijo s podatki v tabelah na katerih ima pravico

Vloga RESOURCE dovoljuje uporabniku:

- vse, kar dovoljuje CONNECT
- ustvarjanje objektov v PB
- dodeljevanje pravic drugim uporabnikom nad svojimi objekti

Vloga DBA dovoljuje uporabniku:

- vse, kar dovoljuje RESOURCE
- ustvarjanje PB
- zaganjanje in zaustavljanje baze
- ustvarjanje uporabnikov
- izdelovanje varnostnih kopije in obnavljanje celotne ali dela PB

Za dodeljevanje vlog uporabimo stavek:

```
1 GRANT ime_vloge TO uporabnik;
```

Za odvzemanje vlog pa:

```
1 REVOKE ime_vloge FROM uporabnik;
```

2.3 DML

Delo nad obstoječimi tabelami.

- SELECT (poizvedba) – glej poglavje 1
- INSERT (vnos, najmanjša enota pri vnosu je ena vrstica)
- UPDATE (posodabljanje, najmanjša enota je stolpec v vrstici)
- DELETE (brisanje, najmanj kar lahko brišemo je ena vrstica)

2.3.1 INSERT

Vstavljanje vrstic v tabelo:

```
1 INSERT INTO ime_tabele (atribut1, atribut2, atribut3)
2     VALUES ('vrednost1', 'vrednost2', 'vrednost3');
```

- Števila se ne vpisuje v navednicah.
- S takšno sintakso je možno vnesti samo eno vrstico naenkrat.
- Možno je dodajati tudi vrstice z NULL vrednostmi.
- Tabelo lahko polnimo tudi na podlagi vrednosti iz druge tabele.

2.3.2 UPDATE

Posodabljanje vrstic v tabeli:

```
1 UPDATE ime_tabele
2    SET ime_stolpca = izraz | NULL
3  WHERE pogoj;
```

Pogoj pove, katere vrstice naj se posodobijo. Če ga ne navedemo, se posodobijo vse vrstice.

2.3.3 DELETE

Brisanje vrstic iz tabele:

```
1 DELETE FROM ime_tabele
2       WHERE pogoj;
```

Pogoj pove, katere vrstice naj se izbrišejo. Če ga ne navedemo, se pobrišejo vse vrstice. Če iz tabele pobrišemo vse vrstice, ta še vedno obstaja v PB.

2.4 TCL

TCL (Transaction Control Language): S temi ukazi nadziramo spremembe narejene z ukazi DML. Najbolj pomembna ukaza sta:

- COMMIT (uveljavi transakcijo): podatki v PB se uveljavijo in postanejo vidni drugim uporabnikom
- ROLLBACK (razveljavi transakcije od zadnje uveljavitve naprej): uporabnik razveljavi operacije

Samodejno uveljavljanje se lahko vklopi z ukazom `AUTOCOMMIT ON` in izklopi z `AUTOCOMMIT OFF` (privzeto).

Poglavje 3

Podatkovni tipi, omejitve in objekti

V tem poglavju bomo obravnavali lastnosti in funkcije, pri katerih se posamezne podatkovne baze do neke mere med seboj razlikujejo, zato se bomo precej bolj kot v prejšnjih dveh poglavjih naslonili samo na en SUPB (v našem primeru Oracle).

3.1 Podatkovni tipi

Podatkovni tipi v Oracle-u se lahko razdelijo v naslednje skupine:

skupine	**podatkovni tipi**

znakovni tipi	`CHAR[(dolžina)]`
	`VARCHAR2(dolžina)`
	`...`

številski tipi	`NUMBER(p,s)` `...`

datumski tip	`DATE`
	`TIMESTAMP`
	`...`

tipi LOB (Large Object)	`BLOB`
	`CLOB`
	`BFILE`
	`...`

tipi RAW	`RAW` `LONG RAW`

ostali tipi	`XMLType`
	`ROWID`
	`...`

Preglednica 3.1: Podatkovni tipi v Oracle-u.

3.2 Omejitve

Poznamo več vrst omejitev (angl. constraint) za zagotavljanje celovitosti podatkov:

- Obveznost podatkov: NULL/NOT NULL
- Omejitve domene (angl. Domain constraints): DOMAIN
- Pravila za celovitost podatkov (angl. Integrity constraints)
  - Celovitost entitet (angl. Entity Integrity): PRIMARY KEY
  - Celovitost povezav (angl. Referential Integrity): FOREIGN KEY
- Številnost (angl. Multiplicity)
- Splošne omejitve (angl. General constraints)

O omejitvah:

- Omejitev je potrebno poimenovati, drugače ga sistem poimenuje z imenom SYS_Cn (n je število).
- Omejitev se ustvari
  - istočasno z ustvarjanjem tabele ali
  - po tem, ko tabela že obstaja.
- Omejitev se opredeli na nivoju tabele ali na nivoju stolpca.

3.2.1 NULL

Določa, da stolpec ne more vsebovati NULL-vrednosti. Definiran samo na nivoju stolpca.

```
1 CREATE TABLE zaposlenci (
2                 id NUMBER(6),
3           priimek VARCHAR(25) NOT NULL,
4             placa NUMBER(8,2),
5 datum_zaposlitve DATE CONSTRAINT zap_datum_zap_nn NOT NULL);
```

3.2.2 UNIQUE

Definiran na nivoju stolpca ali tabele. Določa, da morajo biti vrednosti v stolpcu unikatne (se ne smejo ponovljati). Implicitno se definira še indeks nad stolpcem. Omejitev UNIQUE lahko določimo ob ustvarjanju table:

```
1 CREATE TABLE zaposlenci (
2                id NUMBER(6),
3           priimek VARCHAR(25) NOT NULL,
4          e_naslov VARCHAR2(25) UNIQUE,
5             placa NUMBER(8,2),
6 datum_zaposlitve DATE NOT NULL);
```

ali naknadno:

```
1 ALTER TABLE zaposlenci ADD
2 CONSTRAINT zap_e_naslov_uk UNIQUE (e_naslov);
```

3.2.3 PRIMARY KEY

Definiran na nivoju stolpca ali tabele. Določa vsako vrstico v tabeli.

```
1 CREATE TABLE oddelki (
2        oid NUMBER(4),
3      naziv VARCHAR2(20) NOT NULL,
4   lokacija NUMBER(4)
5 CONSTRAINT oddelki_oid_pk PRIMARY KEY (oid));
```

ali

```
1 ALTER TABLE oid
2 ADD CONSTRAINT oddelek_oid_pk PRIMARY KEY (oid);
```

Nad primarnim ključem se samodejno upošteva omejitev UNIQUE.

3.2.4 FOREIGN KEY

Definiran na nivoju stolpca ali tabele. Vzpostavi relacijo na referencirano tabelo. Opredelitev tujih ključev med ustvarjanjem tabele:

```
1 CREATE TABLE zaposlenci(
2                id NUMBER(6),
3           priimek VARCHAR(25) NOT NULL,
4          e_naslov VARCHAR2(25),
5             placa NUMBER(8,2),
6 datum_zaposlitve DATE NOT NULL,
7               oid NUMBER(4),
8 CONSTRAINT zap_oddelek_fk  FOREIGN KEY (oid) REFERENCES oddelki(oid),
```

```
9 CONSTRAINT zap_e_naslov_uk UNIQUE(e_naslov));
```

ali naknadno:

```
1 ALTER TABLE zaposlenci
2 ADD CONSTRAINT zap_oddelek_fk FOREIGN KEY (oid) REFERENCES oddelki(oid)
3 ON UPDATE CASCADE ON DELETE CASCADE;
```

3.2.5 CHECK

CHECK opredeljuje pogoj, ki mora biti resničen.

```
1 CREATE TABLE zaposlenci (
2 ...
3 placa NUMBER(8,2)
4 CONSTRAINT zap_placa_min CHECK (placa > 0),
5 ...
```

3.2.6 Odstranjevanje in onemogočanje omejitev

Odstranjevanje omejitve:

```
1 ALTER TABLE ime_tabele
2 DROP CONSTRAINT ime_omejitve;
```

Omogočanje omejitve:

```
1 ALTER TABLE ime_tabele
2 DISABLE CONSTRAINT ime_omejitve;
```

Omogočanje omejitve:

```
1 ALTER TABLE ime_tabele
2 ENABLE CONSTRAINT ime_omejitve;
```

3.3 Pogledi

Pogled predstavlja logično tabelo, ki temelji na tabeli ali drugem pogledu. Pogled sam po sebi ne vsebuje nobenega podatka. Lahko bi rekli, da pogled predstavlja okno, skozi katerega uporabnik vidi podatke v eni ali več tabelah. V bazi je pogled shranjen kot SQL-stavek.

Razlogi za uporabo pogledov:

- omejitev dostopa do podatkov (s pogledom se prikažejo le izbrani stolpci in izbrane vrstice)
- skrivanje kompleksnosti podatkov (pogled se obnaša kot ena tabela, čeprav so podatki v njem iz več tabel)
- zagotavljanje podatkovne neodvisnosti in
- za predstavitev različnih pogledov na podatke (pogled predstavlja način za preimenovanje stolpcev, ne da bi dejansko spremenili definicijo osnovne tabele).

3.3.1 Ustvarjanje pogleda

```
1 CREATE VIEW ime_pogleda (alias_1, alias_2,...alias_n) AS
2 SELECT A1, A2,...An
3   FROM T1, T2,...Tn
4  WHERE P1, P2,...Pn;
```

3.3.2 Posodabljanje pogleda

```
1 CREATE OR REPLACE VIEW ime_pogleda (id_zaposlenca, "ime in priimek",
      placa, st_oddelka) AS
2 SELECT id, ime || ' ' || priimek, placa, oid
3   FROM zaposlenci
4  WHERE oid=80;
```

Pogled se nadomesti z novim.

3.3.3 Brisanje pogleda

```
1 DROP VIEW ime_pogleda;
```

3.3.4 Materializirani pogledi

Tabela, ki jo dobimo kot rezultat poizvedbe (znotraj pogleda), se shrani.

```
1 CREATE MATERIALIZED VIEW ...
```

3.4 Sekvence

Sekvenca je objekt v podatkovni bazi, ki si ga lahko deli več uporabnikov. Samodejno generira unikatna cela števila. Tipično se uporablja za generiranje vrednosti primarnega ključa in tako nadomešča programsko kodo.

3.4.1 Ustvarjanje sekvence

```
1 CREATE SEQUENCE oddelki_oid_seq
2     INCREMENT BY 10
3       START WITH 120
4         MAXVALUE 9999
5          NOCACHE
6          NOCYCLE;
```

Uporaba opcije CYCLE v sekvenci ni priporočljiva v primeru generiranja primarnih ključev, razen če se uporablja zanesljiv mehanizem, ki zagotavlja, da se stare vrstice brišejo hitreje, kot pa sekvenca ponovno generira že obstoječe vrednosti.

3.4.2 Uporaba sekvence

```
1 INSERT INTO oddelki(oid, naziv, lokacija)
2       VALUES (oddelki_oid_seq.NEXTVAL,'Podpora', 2500);
```

3.4.3 Posodabljanje sekvence

```
1 ALTER SEQUENCE oddelki_oid_seq
2 INCREMENT BY 20
3      MAXVALUE 9999
4       NOCACHE
5       NOCYCLE;
```

3.5 Indeksi

Indeks je podatkovna struktura, ki SUPB-ju omogoča hitrejše lociranje zapisov v tabeli.
Osnovna sintaksa za ustvarjanje indeksa:

```
1 CREATE [UNIQUE] INDEX ime_indeksa
2     ON ime_tabele (ime_stolpca1 [ASC|DESC],
3                    ime_stolpca2 [ASC|DESC], ... );
```

Brisanje indeksa:

```
1 DROP INDEX ime_indeksa ON ime_tabele;
```

Primer ustvarjanja indeksa po imenu oddelka v tabeli oddelkov:

```
1 CREATE INDEX po_naziv ON oddelki(naziv);
```

Primer ustvarjanja indeksa po imenu oddelka in lokaciji skupaj:

```
1 CREATE INDEX po_naziv_lokacija ON oddelki(naziv, lokacija);
```

Ko so indeksi ustvarjeni, se uporabljajo samodejno. SUPB med načrtovanjem poizvedbe izbere, katere indekse bo uporabil. Izrecno uporabo oziroma neuporabo indeksov dosežemo z t.i. “namigi”.

3.6 Namigi

Namigi so dodatna navodila prevajalniku, ki spremenijo načrt izvedbe podane SQL-poizvedbe.
V Oracle-u se namigi zapišejo v bločni komentar:

```
1 /*+ IME_NAMIGA(parametri) */
```

Nekaj namigov za indeksiranje:

- INDEX(ime_tabele ime_indeksa) – uporabi indeks nad tabelo
- INDEX_ASC(ime_tabele ime_indeksa) – uporabi naraščajoči indeks
- INDEX_DESC(ime_tabele ime_indeksa) – uporabi padajoči indeks
- NO_INDEX(ime_tabele ime_indeksa) – ne uporabi indeksa
- FULL(ime_tabele) – ne uporabi nobenega indeksa

Primer izrecne uporabe indeksa po imenu oddelka in lokaciji skupaj:

```
1 SELECT /*+ INDEX(o po_naziv_lokacija) */ *
2 FROM oddelki o
3 ORDER BY naziv, lokacija;
```

Brez uporabe indeksov:

```
1 SELECT /*+ FULL(oddelki) */  naziv, lokacija
2 FROM oddelki
3 ORDER BY naziv, lokacija;
```

Poglavje 4

SQL in proceduralno programiranje

4.1 PL/SQL

PL/SQL (angl. Procedural Language/Structured Query Language) je proceduralna (postopkovna) razširitev SQL-a.

- Programi v PL/SQL so sestavljeni iz:
  - procedur,
  - funkcij in
  - neimenovanih (“anonimnih”) blokov.
- Vsak izmed naštetih je sestavljen iz osnovnih PL/SQL enot – blokov.
- Bloki so lahko tudi gnezdeni.

Splošna sestava PL/SQL-bloka:

```
1 [DECLARE
2 -- deklaracije]                    --neobvezno
3 BEGIN
4 -- izvršljivi stavki (proceduralna koda)
5 [EXCEPTION
6 -- obravnava izjem]                --neobvezno
7 END;
```

4.1.1 Deklaracije

Deklariramo:

- spremenljivke
- konstante
- izjeme
- objekte
- kazalce na vrstice (kurzorje)
- ...

Primeri deklaracij:

```
1 imeZaposlenca VARCHAR2(20);
2 placa NUMBER(6, 2) NOT NULL := 600;
3 NAJVECJE_STEVILO_STRANK CONSTATNT NUMBER := 100;
```

Skrajšan seznam podatkovnih tipov, ki jih lahko uporabimo, se nahaja v razdelku 3.1.

V DECLARE lahko spremenljivkam priredimo tudi začetne vrednosti, v nasprotnem primeru ima privzeto vrednost NULL.

Primeri deklaracij tipa spremenljivke s pomočjo tipa druge spremenljivke:

```
1 zaposlenec zaposlenci.priimek%TYPE;
2 zaposlenec2 zaposlenec%TYPE;
3 zaposlenecZapis zaposlenci%ROWTYPE;
```

4.1.2 Prirejanje

- normalno prirejanje :=
- kot rezultat stavkov SELECT ali FETCH

Primer prirejanja:

```
1 zaposlenec := 'Korošec';
```

4.1.3 Pogoji

Stavek IF ima naslednjo obliko:

```
IF pogoj1 THEN stavki1
[ELSIF pogoj2 THEN stavki2]
...
[ELSE stavki]
END IF;
```

Primer uporabe IF:

```
IF (pozicija = 'ravnatelj') THEN
   placa := placa * 1.2;
ELSIF (pozicija = 'podravnatelj') THEN
   placa := placa * 1.1;
ELSE
   placa := placa * 1.05;
END IF;
```

Lahko uporabimo tudi CASE:

```
CASE pozicija
    WHEN 'ravnatelj' THEN placa := placa * 1.2;
    WHEN 'podravnatelj' THEN placa := placa * 1.1;
    ELSE placa := placa * 1.05;
END CASE;
```

4.1.4 Zanke

Za zapis zanke uporabimo rezervirano besedo LOOP. Osnovna sintaksa je:

```
[ime_zanke:]
LOOP
    SQL-stavki...
    EXIT [ime_zanke] [WHEN pogoj]
END LOOP [ime_zanke];
```

Lahko uporabimo tudi WHILE:

```
WHILE pogoj LOOP
    SQL-stavki
END LOOP [ime_zanke];
```

Sintaksa za FOR:

```
FOR indeksnaSpremenljivka
 IN spodnjaMeja .. zgornjaMeja LOOP
    SQL-stavki
END LOOP [ime_zanke];
```

4.1.5 Izjeme

- vnaprej določene izjeme (npr. NO_DATA_FOUND, TOO_MANY_ROWS, ZERO_DIVIDE, INVALID_CURSOR . . .)
- izjeme, ki jih določi uporabnik (RAISE ime_izjeme)

Sintaksa za opredelitev in obravnavanje izjeme:

```
DECLARE
    ime_izjeme EXCEPTION;
    PRAGMA
    EXCEPTION_INIT (ime_izjeme, stevilka_napake);
BEGIN
    ...
EXCEPTION
  WHEN ime_izjeme THEN
     obravnavanje izjeme...
  WHEN ime_vnaprej_dolcene_izjeme THEN
     obravnavanje izjeme...
END;
```

4.1.6 Kazalci na vrstice

Kadar dobimo kot rezultat poizvedbe poljubno število vrstic (lahko tudi 0 ali 1), moramo za dostop do posamezne vrstice uporabiti kazalec na vrstico. Kazalec na vrstico je potrebno najprej deklarirati in "odpreti" (na koncu pa "zapreti"). Podatke iz posamezne vrstice dobimo s FETCH.

Primer uporabe poizvedbe, ki vrne en rezultat:

```
1  DECLARE
2      stevilo    NUMBER;
3      stKosov    narocilo.kosov%TYPE := 5;
4  BEGIN
5      SELECT    COUNT ( * )
6        INTO    stevilo
7        FROM    narocilo
8       WHERE    narocilo.kosov > stKosov;
9       dbms_output.put_line('število naročil z več kot ' || stKosov || '
           kosi: ' || stevilo);
10 END;
```

Primer uporabe poizvedbe, ki vrne več rezultatov:

```
1  DECLARE
2      stKosov    narocilo.kosov%TYPE := 5;
3      vrstica    narocilo%ROWTYPE;
4
5      CURSOR narociloKazalec
6      IS
7          SELECT    *
8            INTO    vrstica
9            FROM    narocilo
10          WHERE    narocilo.kosov > stKosov;
11 BEGIN
12     OPEN narociloKazalec;
13     LOOP
14         FETCH narociloKazalec INTO vrstica;
15         EXIT WHEN narociloKazalec%NOTFOUND;
16         DBMS_OUTPUT.put_line('Zastopnik '|| vrstica.aid ||' je prodal '||
               vrstica.kosov ||' kosov.');
17     END LOOP;
18
19     IF narociloKazalec%ISOPEN
20     THEN
21         CLOSE narociloKazalec;
22     END IF;
23 END;
```

Parametrizacija

Definicijo kazalca lahko spremenimo tako, da sprejme parametre. Potem lahko kazalec odpremo z različnimi parametri. Primer:

```
1  DECLARE
2      ...
3     CURSOR narociloKazalec (stKosov narocilo.kosov%TYPE)
4     IS
5        SELECT    *
6          INTO    vrstica
7          FROM    narocilo
8         WHERE    narocilo.kosov > stKosov;
9  BEGIN
10    OPEN narociloKazalec(5);
11    ...
12    OPEN narociloKazalec(10);
13    ...
```

Spreminjanje podatkov preko kazalcev

Če želimo zagotoviti, da se vrstice med definicijo, odpiranjem in branjem podatkov ne spreminjajo, dodamo na konec definicije kazalca FOR UPDATE [NO WAIT].

4.1.7 Procedure in funkcije

- procedura ne vrača vrednosti
- funkcija vrača eno vrednost

Definicija procedure:

```
1  CREATE OR REPLACE PROCEDURE ime_procedure [(ime_parametra1 tip_parametra1
       , ...)] AS ...
```

Proceduro ali funkcijo potem pokličemo z:

```
1  EXECUTE ime_procedure[(parameter1, ...)];
```

4.2 Prožilci

Prožilec predstavlja blok (PL/SQL) kode, ki se izvede ob določenem dogodku v bazi. Dogodki, ki lahko povzročijo proženje:

- INSERT, UPDATE, DELETE
- (Oracle) CREATE, ALTER, DROP
- (Oracle) prijava/odjava uporabnika, zagon/zaustavitev sistema
- (Oracle) določene napake

Poenostavljena sintaksa za ustvarjanje prožilca:

```
1 CREATE TRIGGER ime_prozilca
2         BEFORE | AFTER | INSTEADOF
3         INSERT | DELETE | UPDATE [OF seznam_stolpcev]
4         ON ime_tabele
5         [REFERENCING {OLD | NEW} AS {staro_ime | novo_ime}]
6         [FOR EACH {ROW | STATEMENT}]
7         [WHEN pogoj]
8         koda, ki naj se izvede
```

Primer uporabe prožilca:
Cena izdelka se neprestano spreminja. Zaradi tega je pomembno, da se beleži zgodovina cen.
Izdelamo lahko prožilec, ki bo posodobil tabelo s podatki o zgodovini cen vsakič, ko se bo cena izdelka v osnovni tabeli izdelkov spremenila.

```
1  CREATE TABLE izdelki_zgodovina_cen
2  (
3     iid              NUMBER (5),
4     naziv            VARCHAR2 (32),
5     dobavitelj       VARCHAR2 (32),
6     cena_na_enoto    NUMBER (7, 2)
7  );
8
9  CREATE TABLE izdelki
10 (
11    iid              NUMBER (5),
12    naziv            VARCHAR2 (32),
13    dobavitelj       VARCHAR2 (32),
14    cena_na_enoto    NUMBER (7, 2)
15 );
16
17 CREATE OR REPLACE TRIGGER prozilec_zgodovina_cen
18    BEFORE UPDATE OF cena_na_enoto
19    ON izdelki
```

```
20      FOR EACH ROW
21  BEGIN
22      INSERT INTO izdelki_zgodovina_cen
23        VALUES    (:old.iid,
24                   :old.naziv,
25                   :old.dobavitelj,
26                   :old.cena_na_kos);
27  END;
```

Dodatek A

Relacije

NAROCILO

Glej A.1.

Relacija	Relacijska shema
stranka	*STRANKA*(*sid*, *ime*, *priimek*, *mesto*, *popust*)
agent	*AGENT*(*aid*, *ime*, *priimek*, *mesto*, *provizija*)
izdelek	*IZDELEK*(*iid*, *naziv*, *dobavitelj*, *zaloga*, *cena_na_kos*)
narocilo	*NAROCILO*(*nid*, #*sid*#*aid*, #*iid*, *datum*, *kosov*)

Preglednica A.1: Relacije *NAROCILO*.

STUDENTI

Glej A.2.

Relacija	Relacijska shema
student	*STUDENT*(*vpisna*, *ime*, *priimek*, *smer*, *solnina_placana*)

Preglednica A.2: Relacije *STUDENTI*.

ZAPOSLENCI

Glej A.3.

Relacija	Relacijska shema
zaposlenci	*ZAPOSLENCI*(*id*, *ime*, *priimek*, *vzdevek*, *e_naslov*, *placa*, *datum_zaposlitve*, #*oid*)
oddelki	*ODDELKI*(*oid*, *naziv*, *lokacija*)

Preglednica A.3: Relacije *ZAPOSLENCI*.