Povpraševalni jezik
(SQL-Structured query language)

SQL omogoča uporabnikom dostop do podatkov, ki se nahajajo v (relacijski) bazi podatkov kot so: Oracle, Sybase, Informix, Microsoft SQL Server, Access,.... SQL, prav tako, pa tudi njihovo definiranje(kreiranje) in obdelovanje(procesiranej).
SQL, ki ga v tem dokumentu koristimo je ANSI verzija tega jezika, oziroma standardni SQL.

Vsebina

Ukaz SELECT

U relacijski bazi podatkov se podatki nahajajo v tabelah (vrsticah tabele). Primer: tabela DIJAKI vsebuje enotno matično številko (EMŠO), ime, priimek, naslov, in razred:

Dijaki					
EMŠO	**Ime**	**Priimek**	**Naslov**	**Kraj**	**Razred**
512687458	Janko	Kos	Cankarjeva 10	N. Gorica	3Ra
758420012	Marija	Simčič	Kojsko 5	Kojsko	4Ga
102254896	Milan	Drolc	Delpinova 17	Šempeter	2Rb
876512563	Maja	Leban	XXX divizije 10	N.Gorica	3Gb
122113311	Aleš	Delpin	Bukovica 2	Bukovica	4Ga

Če želimo izpis vseh dijakov(vseh vrstic tabele), uporabimo ukaz SELECT:

```
SELECT Ime, Priimek, Naslov, Kraj, Razred
FROM Dijaki;
```

Splošna obika ukaza SELECT je:

```
SELECT ImeStolpca, ImeStolpca, ...
FROM ImeTabele;
```

Če hočemo dobiti vse vrstice tabele, napišemo naslednji ukaz:

```
SELECT * FROM ImeTabele;
```

Vsak sistem za upravljanje podatkovne baze (SUPB-DBMS) in programi za delo z njim, imajo različne načine prijavljanja ter različne načine pisanja SQL ukazov (SQL, SQLDeveloper,....)

Poizvedovanja

Relacijski operatorji

SQL- ima šest različnih relacijskih operatorjev:

=	Enako
< ali !=	Različno
<	Manjše
>	Večje
<=	Manjše ali enako
>=	Večje ali enako

Če hočemo izpis samo tistih vrstic tabele, ki zadovoljujejo nek kriterij (pogoj), napišemo klavzulo WHERE.

Primer: Izpis vseh imen in priimekov dijakov iz 4Ga

```
SELECT Ime,Priimek
FROM DIJAKI
WHERE Razred = '4Ga';
```

Sestavljeni (kompleksni) kriteriji (pogoji)

Operator AND omogoča kombinacijo dveh ali več pogojev ter izpis vrstic tabele, če podatki, zadovoljujejo **vse** navedene pogoje.

Primer:

```
SELECT Ime,Priimek
FROM DIJAKI
WHERE Razred = '4Ra' AND Kraj='N.Gorica';
```

Operator OR omogoča kombinacijo dveh ali več pogojev ter izpis vrstic tabele, če podatki, zadovoljujejo **vsaj eden** od navedenih pogojev.

Primer:

```
SELECT Ime,Priimek
FROM DIJAKI
WHERE Razred = '4Ra' OR Razred = '4Rb';
```

Operatorja AND in OR lahko tudi kombiniramo.

Primer:

```
SELECT Ime,Priimek
FROM DIJAKI
WHERE Razred = '4Ra' AND (Kraj='N.Gorica' OR Kraj='Šempeter');
```

Operatorji IN in BETWEEN

S pomočjo operatorjev IN in BETWEEN lažje kombiniramo pogoje.

Primer: Izpis imen in priimekov učencev iz 4Ra in 4Rb razreda.

```
SELECT Ime,Priimek
FROM DIJAKI
WHERE Razred IN ( '4Ra', '4Rb');
```

ali:
Primer: Izpis identifikacijskih številk delavcev, ki imajo znesek plače med 100.000 Sit in 200.000 SIT.

```
SELECT IDDelavca
FROM PLAČE
WHERE Znesek BETWEEN 100000 AND 200000;
```

ali:
Primer: Izpis identifikacijskih številk delavcev, ki niimajo plače med 100.000 Sit in 200.000 SIT.

```
SELECT IDDelavca
FROM PLACE
WHERE Znesek NOT BETWEEN 100000 AND 200000;
```

Vrstni red izračunavanja logičkih operacij v SQL-u (operacije računamo z leve proti desni) je :

1. NOT
2. AND
3. OR

Pri tem upoštevamo matematično notacijo (oklepaje,...).

Operator LIKE

Če želimo izpis dijakov, ki imajo priimek, ki se začne z "M", napišemo sledeči ukaz:

```
SELECT IDdijaka
FROM DIJAKI
WHERE Priimek LIKE 'M%';
```

ali tiste, ki jim se priimek konča z črko "A"

```
SELECT IDdijaka
FROM DIJAKI
WHERE Priimek LIKE '%A';
```

ali tiste, ki imajo v priimeku črko "A"

```
SELECT IDdijaka
FROM DIJAKI
WHERE Priimek LIKE '%A';
```

Relacije med tabelami (povezovanje tabel)

V tem poglavju obravnavamo "*notranje povezovanje tabel",* ki ga tudi največkrat uporabljamo v praksi.
Pravilno načrtovanje baze podatkov pomeni, da vsaka tabela vsebuje samo podatke o enem entitetu, dodatne informacije(podatke) pa dobimo iz drugih tabel tako, da tabele povežemo.

Primer:

Učenci

ID	Priimek	Ime
01	Kos	Branko
02	Leban	Aleš
15	Lasič	Pavle
21	Jakovčič	Nataša
50	Filipčič	Simon

Predmeti

ID	Naziv
50	MAT
02	ANJ
21	APJ
15	POS

KončnaOcene

Idučenca	IDpredmeta	Ocena
01	50	4
02	15	3
15	02	3
21	50	4
50	01	5
01	21	1
02	21	2

Ključi

V relacijskih bazah podatkov, podatke hranimo v tabelah(vrsticah tabel).

Primarni ključ je podatek ili množica(kombinacija) podatkov, v tem primeru ga imenujemo sestavljeni ključ, ki enolično določa ostale podatke v vrstici tabele.

Tuji ključ je podatek, ki je primarni ključ v drugi tabeli. V terminologiji relacijskih baz podatkov, povezavo med primarnim in tujim ključem imenujemo ***referenčna integriteta***.

Poleg le-teh obstaja še **unique ključ**, ki prav tako enolično določa ostale podatke v vrstici tabele, ni pa primarni ključ.

Relacije med tabelami (povezovanje tabel)

Praktičen pomen ključev
Ključi(primarni in tuji) omogočajo povezovanje tabel in izbiro(kombinacijo) podatkov, ki se nahajajo v različnih tabelah. Tako ni potrebe po podvajanju podatkov (redundanca podatkov), podatkovna baza pa je konsistetna. Konsistenca podatkov pomeni, da se vsak podatek nahaja v podatkovni bazi samo enkrat, tako niso možne različne vrednosti istega podatka. To je tudi ena izmed največjih prednosti relacijskih podatkovnih baz.

Primer: Seznam priimkov in imen učencev ter njihovih ocen iz predmeta APJ

```
SELECT priimek.ucenci,ime.ucenci,ocena.ocene
FROM Ucenci,Predmeti,Ocene
WHERE ucenci.id = ocene.iducenca AND
      predmeti.id = ocene.idpredmeta AND
      predmeti.naziv='APJ';
```

Pozor: Tabele povezujemo tako, da zapišemo pogoj ucenci.id = ocene.iducenca ter predmeti.id =ocene.idpredmeti. Poveuzujemo torej primarni ključ iz ene tabele z odgovarjajočim tujim ključem iz druge tabele (le-ti podatki morajo imeti isti podatkovni tip in dolžino). Temu načinu povezovanja pravimo povezovanje z enačajem.
Dvoumnosti v zapisovanju podatkov se izognemo tako, da pred imenom podatka zapišemo še ime tabele.

Ukaz DISTINCT in odpravljanje podvajanja podatkov

Če želimo izpis ID-ja,imena in priimka učencev, ki imajo vsaj eno ocene, podatke iščemo v tabelah Učenci in Ocvene, kjer vsak učenec ima lahko več zapisov. V konkretnem primeru pa nas ne zanima koliko ocen učenec ima temveč samo ali ocene sploh ime. To pomeni, da z SQL ukazom moramo odpraviti (eliminirati) podvojene zapise oz. izpisati moramo učenca samo enkrat. V tem primeru uporabljamo klavzulo **DISTINCT**.

```
SELECT DISTINCT Iducenca, Ime, Priimek
FROM Ucenci, Ocene
WHERE ucenci.id = ocene.iducenca;
```

Če želimo podatke tudi urediti, uporabljamo klavzulo **ORDER BY**

```
SELECT DISTINCT Iducenca, Ime, Priimek
FROM Ucenci, Ocene
WHERE ucenci.id = ocene.iducenca
ORDER BY Priimek, Ime;
```

Psevdonimi in IN podvprašanja

```
SELECT Dijaki.Priimek Prii, Ocene.Ocena Oc
FROM Dijaki DIJ, Ocene OCN
WHERE Dij.Id = Ocn.IdDijaka
      AND Oc IN
          (SELECT Oc
           FROM Ocene);
```

Pri tem je potrebno omeniti:

1. "Prii" in "Oc", v prvi vrstici, predstavljajo naslov
2. DIJ in OCN so psevdonimi; to so nova imena za tabele navedene v klavzuli FROM
3. AND v klavzuli WHERE povzroča izvrševanje IN podvprašanja ("= ANY" oz. "= SOME" sta dva ekvivalentna zapisa za IN)

Logična imena in sinonimi

Priporočljiva je uporaba logičnih imen oz. sinonimov, ki nam omogočajo povezavo na različne objekte podatkovne baze: bazo podatkov(instanco), tabelo(tables) ali vpigled(views). Tako se izognemo direktnim povezavam na konkretno bazo podatkov, tabelo ali vpogled oz. omogočamo administratorjem baze podatkov in posredno tudi uporabnikom, bolj fleksibilno delo.

Primer:

```
CREATE Synonym TAB$Ucenci For Ucenci;

CREATE View VIE$Ucenci For VieUcenci;
```

Pomožni SQL ukazi

Agregatne funkcije

SQL pozna pet pomembnih *agregatnih funkcij*: SUM, AVG, MAX, MIN i COUNT. Imenujemo jih agregatne, ker dajejo sumarne rezultate nekoga vprašanja, in ne seznam vseh vrstic.

- SUM () vsota vrednosti stolpca tabele
- AVG () povprečna vrednost stolpca tabele
- MAX () maksimalna vrednost v stolpcu tabele
- MIN () minimalna vrednost v stolpcu tabele
- COUNT(*) število vrstic, ki zadovoljujejo pogoj

Primeri:

```
SELECT SUM(ZnesekPlace), AVG(ZnesekPlace)
FROM PLACE


SELECT MIN(ZnesekPlaca)
FROM PLACE
WHERE delovnoMesto = 'vodja';


SELECT COUNT(*)
FROM PLACE
WHERE delovnoMesto = 'delavec';
```

Kreiranje tabel

Tabelo pred uporabo moramo kreirati. To naredimo na naslednji način:

```
CREATE TABLE Dijaki
(ID INTEGER             NOT NULL,
 Priimek CHAR(40)       NOT NULL,
 Ime CHAR(30)           ...
);
```

Tako določimo ime tabeli in podatke, ki jih le-ta vsebuje, njihov podatkovni tip, dolžino itn.

Podatkovni tipi:

- VarChar2(x), Char(x) – znakovni tip
- Integer – celoštevilčni podatkovni tip
- Decimal(x, y) – decimalna števila
- Date – datumski tip
- Logical – logični podatkovni tip (TRUE ali FALSE).

NOT NULL pomeni, da podatek mora imeti neko vrednost. V nasprotnem primeru je vrednost NULL dovoljena.

Spreminjanje strukture tabele (ALTER TABLE)

Pogosto se pojavi potreba po novem podatku tako, da moramo spremeniti strukturo tabele (vzrok: nove zahteve uporabnika ali zunanje (zakonske) spremembe). To naredimo na naslednji način:

```
ALTER TABLE Dijaki ADD (NASLOV CHAR(30));
```

Dodajanje podatkov v tabelo (INSERT)

Če hočemo dodati podatke o entiteti v tabelo (vrstico tabele) napišemo naslednji ukaz:

```
INSERT INTO Dijaki VALUES (21, 'KOS', 'JURE', 'Cankarjeva 10');
```

Podatke lahko dodamo tudi na naslednji način:

```
INSERT INTO Dijaki (ID, Priimek, Ime)
VALUES (21, 'KOS', 'JURE');
```

Kot vidimo, nismo vnesli podatka Naslov oz. podatek Naslov ima vrednost NULL.

Brisanje podatkov tabele (DELETE)

Podatek o entiteti (vrstico tabele) brišemo na naslednji način:

```
DELETE FROM Dijaki
WHERE Id = 21;
```

Kjer je Id primarni ključ.

Pozor: Če napišemo ukaz

```
DELETE FROM Dijaki
WHERE Priimek = 'KOS';
```

Lahko zbrišemo tudi podatke, ki jih nismo želeli (vse dijake, ki imajo priimek KOS)

Ažuriranje podatkov (UPDATE)

Če hočemo ažurirati stolpec Dijaki.Naslov, ki ima vrednost Null zapišemo naslednji ukaz:

```
UPDATE Dijaki SET Naslov='Gradnikova 10' WHERE Id = 21;
```

Vpogledi (VIEWS)

V SQL-u, lahko definiramo lastne vpoglede.
Primer:

```
CREATE VIEW VpogledOcene AS SELECT Ocene FROM Ocene;
```

Indeksi

Indeksi omogočajo sistemu za upravljanje baz podatakov (SUPB-DBMS), hitrejši dostop do podatkov. Sistem kreira interno strukturo podatkov, tabelo indeksov(indeks), ki omogoča hitrejši dostop do podatkov v osnovni tabeli(s pomočjo kazalcev)

```
Primer:

CREATE INDEX ImeInPriimek_IDX ON Dijaki (Priimek, Ime);
```

Če hočemo indeks zbrisati uporabimo ukaz DROP:

```
DROP INDEX ImeInPriimek_IDX;
```

Z ukazom DROP lahko zbrišemo tudi tabelo iz baze podatkov (pozor: tako zbrišemo tudi vse podatke iz tabele).

Klavzule GROUP BY i HAVING

Operator GROUP BY omogoča povezovanje neke agregatne funkcije z vrsticami tabele (še posebej se to nanaša na ukaz(funkcijo) COUNT, ki šteje vrstice tabele v vsaki grupi).

Primer: Izpis največje(maksimalne) ocene, ki jo ima dijak

```
SELECT IDdijaka, MAX(OCENA)
FROM Ocene
GROUP BY IDdijaka;
```

Ali

Primer: Izpis največje(maksimalne) ocene, ki je večja od 3

```
SELECT IDdijaka, MAX(OCENA)
FROM Ocene
GROUP BY IDdijaka
HAVING OCENA > 3;
```

Še o podvprašanji

Primer: Izpis dijakov, ki imajo oceno za 1 večjo od povprečne ocene

```
SELECT IDUcenca
FROM Ocene
WHERE OCENA >
      (SELECT AVG(OCENA) + 1
      FROM Ocene);
```

Če hočemo odpraviti podvajanja, dodamo še ukaz DISTINCT (Idcenca).

Primer: Izpis priimkov učencev, ki imajo vsaj eno oceno:

```
SELECT PRIIMEK
FROM Ucenci
WHERE ID IN
      (SELECT DISTINCT IDUcenca
      FROM Ocene);
```

ali

```
SELECT PRIIMEK
FROM Ucenci
WHERE ID =
      (SELECT DISTINCT IDUcenca
      FROM Ocene);
```

Klavzuli EXISTS in ALL

Klavzula EXISTS koristi podvprašanje kot pogoj, ki je točen(TRUE), če je rezultat vsaj ena vrstica tabele oz. napačen(FALSE) v nasprotnem primeru.

Primer:

```
SELECT ImeUcenca, PriimekUcenca
FROM Ucenci
WHERE EXISTS
      (SELECT *
      FROM Ocene
      WHERE Predmet='APJ');
```

Klavzula ALL

```
SELECT IDUcenca, Ocena
FROM Ocene
WHERE OCENA >= ALL
      (SELECT OCENA
      FROM Ocene);
```

Rezultat je iducenca in ocena in sicer učenca, ki ima najvišjo(e) oceno(e).

Klavzula UNION in zunanje združevanje

V nekaterih primerih, če hočemo skupen izpis rezultatov več vprašanj, koristimo klavzulo UNION.

Primer:

```
SELECT ID
FROM Ucenci
UNION
SELECT IDUcenca
FROM Ocene;
```

Klavzula UNION avtomatično eliminira podvajanje podatakov.

Embedded SQL

```
Embedded SQL omoguča programerjem, da se povežejo z bazo podatkov in
koristijo SQL ukaze.
```