Poglavje VI

Manipulacija in definicija podatkov

- DML ukazi v SQL
- DDL: kreiranje tabel in pogledov
- DDL: podatkovni tipi in domene
- DDL: omejitve
- Shranjeni podprogrami in bazni prožilci

Pomembnejši DQL-DML ukazi

- SELECT (v osnovi DQL)
- INSERT
- UPDATE
- DELETE

Sklopi jezika SQL:
DQL: data query language ✅
DML: data manipulation language
DDL: data definition language
DCL: data control language
TCL: transaction control language

- POZOR! DML operacije lahko delate na domačih računalnikih, na FRI strežniku pa le v svoji shemi ali v shemi *sandbox* (drugod nimate pravic)!

INSERT stavek...

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

INSERT INTO TableName [(columnList)]
VALUES (dataValueList)

- Seznam columnList ni obvezen; če ga spustimo, SQL interpreter pričakuje vrednosti za vse stolpce tabele, v vrstnem redu, kot so bili kreirani.
- Pri vnosu moramo vpisati najmanj vse obvezne vrednosti (not null), razen za stolpce, pri katerih je bila ob kreiranju določena privzeta vrednost (DEFAULT).

INSERT stavek...

- Seznam dataValueList mora ustrezati seznamu columnList:
  - Število elementov v seznamih mora biti enako;
  - Vrednost, ki se nanaša na nek stolpec, mora biti v seznamu dataValueList na istem mestu, kot je stolpec v seznamu columnList;
  - Podatkovni tip vrednosti, ki se nanaša na nek stolpec, mora biti enak kot podatkovni tip stolpca.

Primeri INSERT stavkov...

Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)

- Vnos nove vrstice v tabelo rezervacija:

INSERT INTO rezervacija
- VALUES (74, 102, DATE'2011-11-20');

- Shema relacije rezervacija:

Rezervacija(jid, cid, dan)

Primeri INSERT stavkov...

Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)

- Tabela jadralec(jid, ime, rating, starost)
- Vnos nove vrstice v tabelo jadralec – vnos samo obveznih vrednosti

```
INSERT INTO jadralec
    VALUES (300, null, null, null);
```

Stolpci ime, starost in rating so neobvezni.

- ali

```
INSERT INTO jadralec(jid, ime, starost, rating)
    VALUES (300, 'Patrik', 20, 8);
```

Vnos vrednosti po specificiranem seznamu atributov.

Primeri INSERT stavkov...

- Vnos množice vrstic

```
INSERT   INTO  TableName [ (columnList) ]
         VALUES (…), (…), …, (…);
```

- Vnos množice vrstic iz ene ali več drugih tabel

```
INSERT   INTO TableName [ (columnList) ]
         SELECT columnList ... ;
```

Primeri INSERT stavkov...

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

- Predpostavimo, da imamo tabelo StariJadralec z enako shemo kot tabela jadralec. Vanjo želimo vnesti oznake, imena in starost jadralcev, starejših od 40 let.

LIKE: prenese opis - atribute, tipe, indekse

```
CREATE TABLE StariJadralec LIKE Jadralec;
INSERT INTO StariJadralec(jid, ime, starost)
        SELECT jid, ime, starost
        FROM jadralec
        WHERE starost > 40;
```

Ratinga ne prenašamo: postane NULL

UPDATE stavek...

UPDATE TableName
SET columnName1 = dataValue1
  [, columnName2 = dataValue2...]
[WHERE searchCondition]

- TableName se lahko nanaša na ime osnovne tabele ali ime pogleda.
- Sklop SET določa nazive enega ali več stolpcev ter nove vrednosti teh stolpcev.

UPDATE stavek

- WHERE sklop je neobvezen:
  - Če ga spustimo, se v imenovane stolpce vpišejo nove vrednosti za vse vrstice v tabeli;
  - Če WHERE sklop določimo, se spremembe zgodijo zgolj za vrstice, ki ustrezajo WHERE pogojem.
- Lahko souporabljamo z WITH stavkom (CTE)
  WITH sinonim AS (poizvedba)
  INSERT INTO ...

- Nove podatkovne vrednosti morajo ustrezati podatkovnemu tipu stolpca.

Primeri UPDATE stavkov

- Vsem starim jadralcem postavimo rating na 10.

```
UPDATE stariJadralec
SET rating=10;
```

- Vsem starim jadralcem zmanjšaj rating za 10%.

```
UPDATE stariJadralec
SET rating=0.9 * rating;
```

Primeri UPDATE stavkov

- Vsem starim jadralcem pripišemo originalni rating iz tabele jadralec.
- Standardni UPDATE nima FROM sklopa!

```
UPDATE stariJadralec sj
SET rating = (   SELECT rating
                 FROM jadralec
                 WHERE jid=sj.jid );
```

Povezava z drugimi tabelami je možna samo preko uporabe vgnezdenih poizvedb ali CTE (WITH stavek).

DELETE stavek

DELETE FROM TableName
[WHERE searchCondition]

- TableName se lahko nanaša na ime osnovne tabele ali ime pogleda.
- WHERE sklop ni obvezen. Če ga spustimo, zbrišemo vse vrstice v tabeli. Tabela ostane.

Primeri DELETE stavkov

- Izbriši vse rezervacije pred letom 2006.

```
DELETE FROM rezervacija
WHERE EXTRACT(YEAR FROM dan) < 2006;
```

- Opomba: **MySQL safe update mode** ne dovoli spreminjanja, torej UPDATE/DELETE brez uporabe primarnega ključa

Primeri DELETE stavkov

- Izbriši vse rezervacije zelenih čolnov pred letom 2006.

```
DELETE FROM rezervacija
WHERE EXTRACT(YEAR FROM dan) < 2006
    AND cid IN (SELECT cid
                FROM coln
                WHERE barva='zelena');
```

Povezava z drugimi tabelami je možna samo preko uporabe vgnezdenih poizvedb ali CTE (WITH stavek).

Stavek TRUNCATE TABLE

- S pomočjo stavka TRUNCATE TABLE izbrišemo vse vrstice tabele (na nivoju datotečnega sistema).

  TRUNCATE TABLE TblName

- Mnogo hitrejše kot DELETE FROM TblName.
- Tabela ne sme imeti integritetnih ali referenčnih omejitev (oziroma biti v njih uporabljena).
- Primer:

  TRUNCATE TABLE jadralec

Stavki skupine SQL DDL...

- DDL skupina ukazov zajema SQL stavke za manipulacijo s strukturo podatkovne baze.
- Tabele in omejitve:
  - Standardni SQL in uporabniško definirani podatkovni tipi
  - Integritetne omejitve
  - Kako definirati omejitve z SQL-om
  - Uporaba integritetnih omejitev v CREATE in ALTER TABLE stavkih
- Pogledi:
  - Kako kreirati in brisati poglede z SQL-om?
  - Kako SUPB izvaja operacije nad pogledi?
  - Pod kakšnimi pogoji so pogledi spremenljivi?
  - Prednosti in slabosti pogledov

Kreiranje tabele

```
CREATE TABLE TableName (
    ImeAtributa1 PodatkovniTip1,
    ImeAtributa2 PodatkovniTip2,
    ...
    Omejitve
    ...
)
```

- Imena (različna!) in tipi atributov (standardni, definirani)

Kreiranje tabele in omejitve

- Omejitve vrednosti atributov (celovitost vrednosti atributov)
- Referenčne omejitve (celovitost povezav)
- Integritetne omejitve (celovitost vrstic, splošne omejitve - poslovna pravila)

- Poimenovanje omejitev: koristno, saj jih lahko naknadno spreminjamo

Standardni podatkovni tipi v ISO SQL

Table 6.1 ISO SQL data types.

Data type	Declarations			
boolean	BOOLEAN			
character	CHAR	VARCHAR		
bit	BIT	BIT VARYING		
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION	
datetime	DATE	TIME	TIMESTAMP	
interval	INTERVAL			
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT	

Integritetne omejitve – celovitost podatkov

- Za zagotavljanje celovitosti podatkov SQL standard ponuja več vrst omejitev:
  - Obveznost podatkov
  - Omejitve domene (Domain constraints); podtipi atributov
  - Pravila za celovitost podatkov (Integrity constraints)
    - Celovitost entitet (Entity Integrity)
    - Celovitost povezav (Referential Integrity)
  - Številnost (Multiplicity)
  - Splošne omejitve (General constraints)
- Omejitve so lahko definirane v CREATE TABLE ali ALTER TABLE stavkih.

Integritetne omejitve – celovitost atributov

- Obveznost podatkov

  ime VARCHAR(10) NOT NULL

- Omejitve domene (PostgreSQL/Oracle/MS SQL/MariaDB/MySQL od 8.0.16; alternative je prožilec)

  CHECK (rating >= 1 AND rating <= 10)

- Domena natančneje kot podatkovni tip določa množico dopustnih vrednosti (domena=podtip)
- Domene lahko eksplicitno definiramo; smiselno če nek podtip pogosto uporabljamo

Integritetne omejitve – celovitost atributov

```
CREATE DOMAIN DomainName [AS] dataType
[DEFAULT defaultOption]
[CONSTRAINT  ime_omejitve]
[CHECK (searchCondition)]
```

Oracle, MySQL/MariaDB:
ne podpirajo.
PostgreSQL, MS SQL Server:
podpirata

Primer:

```
CREATE DOMAIN Tspol AS CHAR
        CONSTRAINT Slovensko
                CHECK (VALUE IN ('M', 'Ž'));
```

Deklaracija atributa:

```
Spol Tspol NOT NULL
```

Integritetne omejitve – celovitost atributov

- searchCondition lahko vsebuje iskalno tabelo (lookup table):

```
CREATE DOMAIN sifraColna AS INTEGER
    CHECK (VALUE IN (SELECT cid FROM Coln));
```

- Domeno lahko ukinemo z uporabo stavka DROP DOMAIN:

```
DROP DOMAIN DomainName
    [RESTRICT | CASCADE]
```

→ Kaj naj se zgodi, če je domena trenutno v uporabi (neuspeh, kaskadno brisanje atributov)

Integritetne omejitve - celovitost vrstic

- Primarni ključ tabele mora vsebovati enolično neprazno vrednost v vsaki vrstici tabele.
- ISO standard podpira primarne ključe s sklopom PRIMARY KEY v okviru CREATE in ALTER TABLE stavkov.

  PRIMARY KEY(EMSO) ≡ NOT NULL, UNIQUE + INDEKS

- Vsaka tabela ima lahko največ en primarni ključ.
- Primarni ključ se običajno avtomatsko indeksira.

Integritetne omejitve - celovitost vrstic

- Enoličnost neosnovnih (niso del primarnega ključa) stolpcev zagotavljamo z uporabo omejitve UNIQUE

  UNIQUE(davcna_stevilka) ali
  CONSTRAINT preveriDavcno UNIQUE(davcna_stevilka)

- UNIQUE zagotavlja enoličnost, ne pa tudi obveznosti; oznaka je lahko tudi NULL!
  - Pozor: za primerjavo velja NULL $\neq$ NULL (torej lahko nastopa večkrat)!
- Alternativni ključ: UNIQUE, NOT NULL za vse atribute, ki ga sestavljajo
  - Tudi UNIQUE se ponavadi avtomatsko indeksira

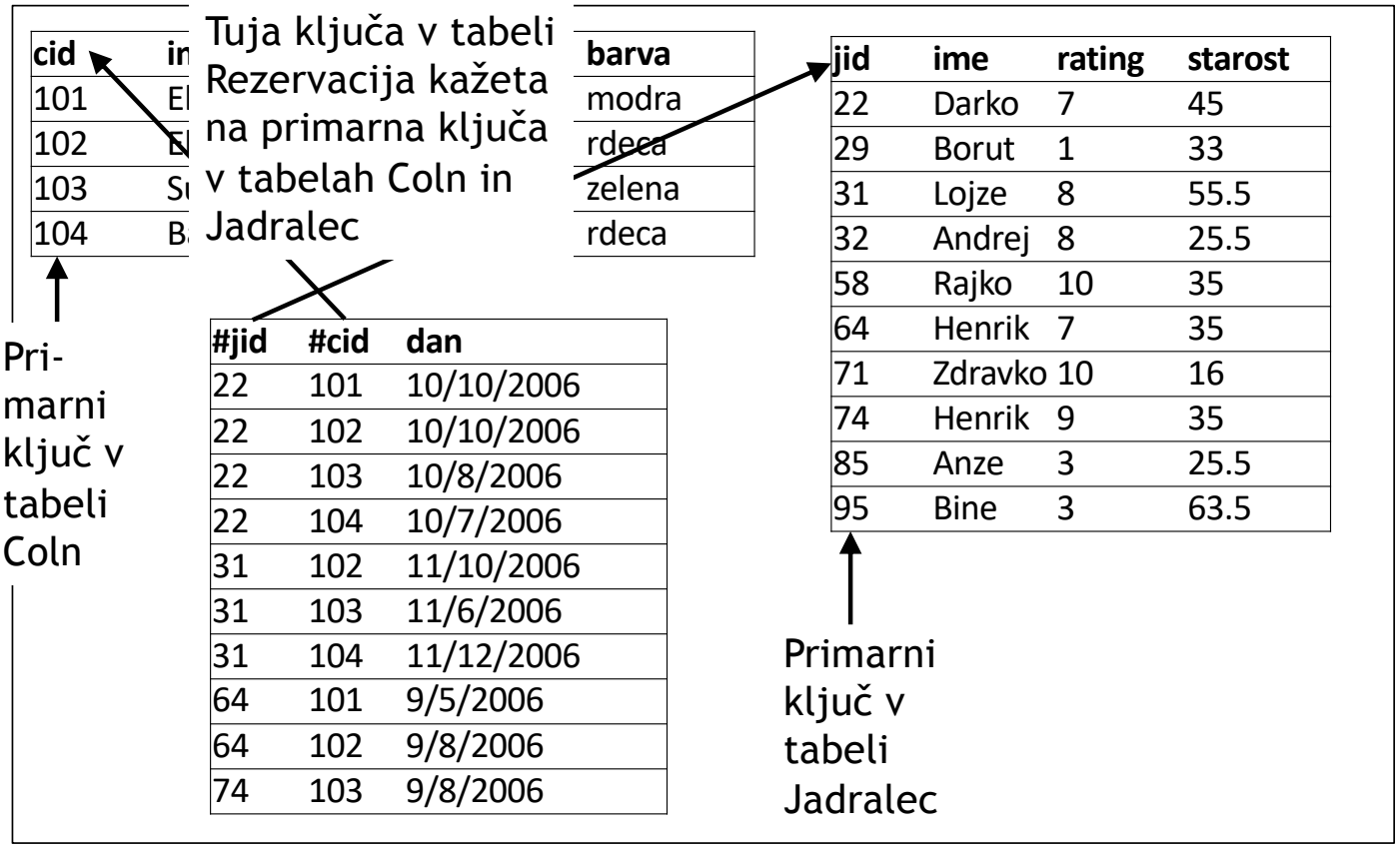
Integritetne omejitve - celovitost povezav

- Implementacija povezav med tabelami s tujimi ključi.
- FOREIGN KEY (tuji ključ) je stolpec ali množica stolpcev, ki povezujejo vsako vrstico tabele A z vrstico referenčne tabele B in se ujemajo vrednosti A.FK = B.PK (primarni ključ ali kandidat za ključ).
- Celovitost povezav zagotavlja, da če ima tuji ključ neko vrednost, potem se ta vrednost nahaja v primarnem ključu povezane tabele.
- ISO standard omogoča definicijo tujih ključev s sklopom FOREIGN KEY v CREATE in ALTER TABLE

  FOREIGN KEY(jid) REFERENCES jadralec(jid)

Primeri tujih ključev

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(#jid, #cid, dan)
```



cid	in
101	El
102	El
103	Su
104	Bi

barva
modra
rdeca
zelena
rdeca

jid	ime	rating	starost
22	Darko	7	45
29	Borut	1	33
31	Lojze	8	55.5
32	Andrej	8	25.5
58	Rajko	10	35
64	Henrik	7	35
71	Zdravko	10	16
74	Henrik	9	35
85	Anze	3	25.5
95	Bine	3	63.5

#jid	#cid	dan
22	101	10/10/2006
22	102	10/10/2006
22	103	10/8/2006
22	104	10/7/2006
31	102	11/10/2006
31	103	11/6/2006
31	104	11/12/2006
64	101	9/5/2006
64	102	9/8/2006
74	103	9/8/2006

Integritetne omejitve - celovitost povezav

- Vsak INSERT/UPDATE stavek, ki skuša kreirati FK vrednost v tabeli, ne da bi ta vrednost obstajala kot PK v povezani tabeli, je zavrnjen.
- Ob UPDATE/DELETE operacijah nad referencirano tabelo so možne naslednje akcije glede na originalno operacijo (ON UPDATE, ON DELETE):
  - CASCADE: spremeni ali briše ustrezne vrstice iz vseh tabel, ki referencirajo spremenjeno ali izbrisano vrstico
  - SET NULL: postavi FK na NULL
  - SET DEFAULT: postavi FK na privzeto vrednost (če je definirana)
  - NO ACTION, RESTRICT: zavrne update/delete operacijo

Integritetne omejitve - celovitost povezav

- Določimo z uporabo ON UPDATE, ON DELETE
  ON UPDATE SET NULL

- Primeri nad tabelo Rezervacija(jid,cid,dan):

  FOREIGN KEY (jid) REFERENCES jadralec(jid)
  ON DELETE SET NULL

  FOREIGN KEY (cid) REFERENCES coln(cid)
  ON UPDATE CASCADE

Splošne integritetne omejitve

- Splošne omejitve niso direktno vezane na definicije tabel.
- Lahko se nanašajo na več tabel istočasno (zato se uporablja poseben ukaz, ne pa CHECK ali CONSTRAINT pri vseh sodelujočih tabelah)

```
CREATE ASSERTION AssertionName
    CHECK (searchCondition)
```

Ni vedno implementirano! Potencialno zelo obremenjujoče za bazo!

Splošne integritetne omejitve – primer

- Na dan lahko rezerviramo samo en čoln.

```
CREATE ASSERTION PrevecRezervacijColna
   CHECK (NOT EXISTS (SELECT cid, dan
                      FROM rezervacija
                      GROUP BY jid, dan
                      HAVING COUNT(*) > 1))
```

- Če ASSERTION ni implementiran, lahko splošne omejitve še vedno preverjamo z baznimi prožilci npr. pri MariaDB, MySQL, PostgreSQL.

Kreiranje podatkovnih objektov...

- SQL DDL omogoča kreiranje, spreminjanje in brisanje podatkovnih objektov, kot so: shema, domena, tabela, pogled in indeks.

- Glavni SQL DDL stavki so:

CREATE SCHEMA	DROP SCHEMA
CREATE/ALTER DOMAIN	DROP DOMAIN
CREATE/ALTER TABLE	DROP TABLE
CREATE VIEW	DROP VIEW
CREATE INDEX	DROP INDEX

Kreiranje podatkovnih objektov...

- Tabele in drugi podatkovni objekti obstajajo v nekem SUPB okolju (instanci).
- Vsako SUPB okolje vsebuje enega ali več katalogov, vsak katalog pa množico shem.
- Shema je poimenovana zbirka povezanih podatkovnih objektov.
- Objekti v shemi so lahko tabele, pogledi, domene, trditve, dodelitve, pretvorbe in znakovni nizi. Vsi objekti imajo istega lastnika.
- Primeri shem na pb.fri.uni-lj.si: vaje, sandbox, vsaka vaša shema

Kreiranje/brisanje/uporaba sheme

- Kreiranje sheme
  - CREATE SCHEMA Name [specifikacije…] – kreiranje prazne sheme
  - CREATE SCHEMA AUTHORIZATION ownerName [create statements] (PostgreSQL – kreiranje prazne sheme s specifikacijo lastnika)
- Brisanje sheme
  - DROP SCHEMA Name [RESTRICT | CASCADE]
    RESTRICT (privzeto): shema mora biti prazna, sicer brisanje ni možno.
    CASCADE: kaskadno se brišejo vsi objekti, povezani s shemo. Če katerokoli brisanje ne uspe, se zavrne celotna operacija.
- Uporaba sheme:
  - SET SCHEMA 'Name' (PostgreSQL) ali USE Name (MariaDB, MySQL)
  - SELECT ime_sheme.ime_objekta (eksplicitno, od koderkoli)

Kreiranje tabele...

```
CREATE TABLE TableName (
    {colName dataType [NOT NULL] [UNIQUE]
        [DEFAULT defaultOption]
        [CHECK searchCondition] [,...]}
    {[FOREIGN KEY (listOfFKColumns)
        REFERENCES ParentTableName [(listOfCKColumns)],
        [ON UPDATE referentialAction]
        [ON DELETE referentialAction ]] [,…]}
    {[CHECK (searchCondition)] [,…] }
    [PRIMARY KEY (listOfColumns),]
    {[UNIQUE (listOfColumns),] […,]}
    {CONSTRAINT ime ...}
)
```

Primer kreiranja tabele...

Najprej kreiramo domene

```
CREATE DOMAIN sifraColna AS INTEGER
   CHECK (VALUE IN (SELECT cid FROM coln));

CREATE DOMAIN rezervDan AS DATE
   CHECK(VALUE BETWEEN DATE'1.1.1995' AND DATE'1.1.2200');

CREATE DOMAIN sifraJadralca AS INTEGER
   CHECK(VALUE BETWEEN 100 AND 999);

CREATE DOMAIN spolJadralca AS CHAR
   CHECK (VALUE IN ('M', 'Ž'));
```

Primer kreiranja tabele...

potem kreiramo tabelo

```
CREATE TABLE Rezervacija (
    jid             sifraJadralca    NOT NULL,
    cid             sifraColna       NOT NULL,
    dan             rezervDan        NOT NULL DEFAULT date(),    -- date(): MySQL/MariaDB
    CONSTRAINT PrevecRezervacijColna                             -- current_date: PostgreSQL
        CHECK (NOT EXISTS (   SELECT *                           -- Gnezdena poizvedba
                              FROM rezervacija                   -- v omejitvi ni vedno podprta
                                   GROUP BY jid, dan             -- (npr. v PostgreSQL)
                                   HAVING COUNT(*) > 1)),
    PRIMARY KEY (jid, cid, dan),
    FOREIGN KEY (jid) REFERENCES jadralec(jid)
        ON DELETE SET NULL ON UPDATE CASCADE,
    FOREIGN KEY (cid) REFERENCES coln(cid)
        ON DELETE SET NULL ON UPDATE CASCADE,
...);
```

ALTER TABLE stavek...

- S stavkom ALTER TABLE lahko:
  - Dodajamo ali ukinjamo stolpce v tabeli;
  - Dodajamo ali ukinemo omejitve tabele;
  - Za stolpce v tabeli določamo ali ukinjamo privzete vrednosti;
  - Spreminjamo podatkovne tipe stolpcev v tabeli;

Primeri ALTER TABLE stavkov...

- Spremeni tabelo rezervacija tako, da ukineš privzeto vrednost stolpca dan.

```
ALTER TABLE rezervacija
        ALTER dan DROP DEFAULT;
```

Primeri ALTER TABLE stavkov...

- Spremeni tabelo rezervacija tako, da ukineš omejitev, da noben čoln ne sme imeti več kot eno rezervacijo na isti dan. V tabelo jadralec dodaj stolpec Spol.

```
ALTER TABLE rezervacija
    DROP CONSTRAINT PrevecRezervacijColna;

ALTER TABLE jadralec
    ADD spol spolJadralca NOT NULL DEFAULT  'M';
```

Stavek DROP TABLE

- S pomočjo stavka DROP TABLE ukinemo tabelo. Obenem se zbrišejo vsi zapisi tabele.

  DROP TABLE TblName [RESTRICT | CASCADE]

- Restrict: Ukaz se ne izvede, če obstajajo objekti, ki so vezani na tabelo, ki jo brišemo.
- Cascade: kaskadno se brišejo vsi vezani objekti.
- Primer:

  DROP TABLE jadralec RESTRICT;

Pogledi v SQL

- Pogled je navidezna relacija (tabela), ki ne obstaja v relacijski bazi, temveč se dinamično kreira takrat, ko jo kdo potrebuje.
- Vsebina pogleda je definirana kot zaporedje operacij nad eno ali več osnovnimi relacijami.
- Pogledi so dinamični → spremembe nad osnovnimi relacijami, katerih atributi so zajeti tudi v pogledu, so v pogledu takoj vidne.
- Preko pogledov lahko podatke (v zelo omejenem obsegu) tudi dodajamo/brišemo/spreminjamo
- SQL sintaksa:

  CREATE VIEW

Pogledi (v smislu relacijskega podatkovnega modela)

- Osnovna relacija (base relation)
  - Poimenovana relacija, ki ustreza nekemu realnemu konceptu.
  - Njene n-terice so fizično shranjene v podatkovni bazi v obliki tabele.

- Pogled (view)
  - Poimenovan rezultat ene ali več operacij nad osnovnimi relacijami z namenom pridobitve nove relacije.
  - Služi za definicijo zunanjih shem (uporabniških pogledov na podatke)

Namen uporabe pogledov

- Predstavljajo uporaben mehanizem za zagotavljanje varnosti, saj zakrivajo posamezne dele podatkovne baze pred določenimi uporabniki.

- Uporabnikom dajejo možnost, da do podatkov dostopajo na prilagojen način → isti podatki so lahko s strani različnih uporabnikov v istem času vidni na različne načine.

- Poenostavljajo kompleksne operacije nad osnovnimi relacijami (podobno kot CTE).

Primeri zbirnega pogleda

cid	ime	dolzina	barva
101	Elan	34	modra
102	Elan	34	rdeca
103	Sun Odyssey	37	zelena
104	Bavaria	50	rdeca

#jid	#cid	dan
22	101	10/10/2006
22	102	10/10/2006
22	103	10/8/2006
22	104	10/7/2006
31	102	11/10/2006
31	103	11/6/2006
31	104	11/12/2006
64	101	9/5/2006
64	102	9/8/2006
74	103	9/8/2006

jid	ime	rating	starost
22	Darko	7	45
29	Borut	1	33
31	Lojze	8	55.5
32	Andrej	8	25.5
58	Rajko	10	35
64	Henrik	7	35
71	Zdravko	10	16
74	Henrik	9	35
85	Anze	3	25.5
95	Bine	3	63.5

```
SELECT j.jid, j.ime, COUNT(*)
FROM jadralec j JOIN rezervacija r USING(jid)
GROUP BY j.jid, j.ime;
```

Stavek CREATE VIEW...

CREATE VIEW ViewName [(newColumnName [,...])]

AS subselect

[WITH [CASCADED | LOCAL] CHECK OPTION]

- Vsakemu stolpcu pogleda lahko dodelimo novo ime.
- Če določimo imena stolpcev, potem morajo stolpci SELECT stavka ustrezati stolpcem pogleda.
- Če imena stolpcev ne določimo, se uporabijo imena stolpcev iz SELECT stavka.
- WITH CHECK OPTION: nadzira spremembe pogleda

Stavek CREATE VIEW...

- WITH CHECK OPTION: zagotavlja, da če vrstica ne izpolnjuje WHERE pogoja, ni dodana v osnovno tabelo, nad katero je pogled osnovan.
- Potrebujemo pravice SELECT nad vsemi tabelami, uporabljenimi v SELECT stavku ter pravice USAGE nad vsemi domenami, ki jih uporabljajo stolpci SELECT stavka.
- Pogled ukinemo z ukazom DROP VIEW

  DROP VIEW ViewName [RESTRICT | CASCADE]

Primer horizontalnega pogleda

- Kreiraj pogled DarkoveRezervacije, tako, da bo jadralec Darko videl samo svoje rezervacije.

```
CREATE VIEW DarkoveRezervacije
AS  SELECT r.*
      FROM rezervacija r, jadralec j
      WHERE r.jid = j.jid AND
                  j.ime='Darko';
```

```
+-----+-----+------------+
| jid | cid | dan        |
+-----+-----+------------+
|  22 | 101 | 2006-10-10 |
|  22 | 102 | 2006-10-10 |
|  22 | 103 | 2006-10-08 |
|  22 | 104 | 2006-10-07 |
+-----+-----+------------+
```

```
SELECT * FROM DarkoveRezervacije;
```

Primer vertikalnega pogleda

- Izdelaj pogled vseh šifer in imen čolnov, ki jih je rezerviral jadralec Lojze.

```
+-----+-------------+
| cid | ime         |
+-----+-------------+
| 102 | Elan        |
| 103 | Sun Odyssey |
| 104 | Bavaria     |
+-----+-------------+
```

```
CREATE VIEW LojzetoviColni
AS  SELECT c.cid, c.ime
      FROM jadralec j, rezervacija r, coln c
      WHERE j.jid=r.jid AND r.cid=c.cid AND
            j.ime='Lojze';
```

Primer pogleda z grupiranjem

- Ustvari pogled, kjer imamo za vsak čoln zapisano šifro, ime in število rezervacij.

```
CREATE VIEW ColnRez (cid, ime, stRez)
AS (
    SELECT c.cid, c.ime, COUNT(*) AS Cnt
    FROM coln c, rezervacija r
    WHERE c.cid = r.cid
    GROUP BY c.cid, c.ime
);
```

```
+-----+------------+-----+
| cid | ime        | Cnt |
+-----+------------+-----+
| 101 | Elan       |   2 |
| 102 | Elan       |   3 |
| 103 | Sun Odyssey |   3 |
| 104 | Bavaria    |   2 |
+-----+------------+-----+
```

Izvedba pogleda...

- Imamo naslednji SELECT stavek

```
SELECT cid, ime, stRez
FROM ColnRez
WHERE cid = 102
ORDER BY ime;
```

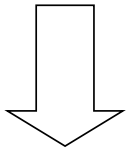
View ColnRez

```
CREATE VIEW ColnRez (cid, ime, stRez)
AS (
  SELECT c.cid, c.ime, COUNT(*) AS Cnt
  FROM coln c, rezervacija r
  WHERE c.cid = r.cid
  GROUP BY c.cid, c.ime);
```

Izvedba pogleda...

(I) Imena stolpcev pogleda iz SELECT stavka so prevedena v imena SELECT stavka, ki definira pogled:

SELECT cid, ime, stRez



SELECT c.cid, c.ime, COUNT(*) AS Cnt

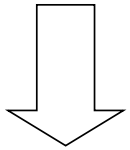
View CoInRez

```
CREATE VIEW CoInRez (cid, ime, stRez)
AS (
  SELECT c.cid, c.ime, COUNT(*) AS Cnt
  ... );
```

Izvedba pogleda...

(II) Imena iz FROM sklopa pogleda so zamenjana z imeni FROM sklopa SELECT stavka, ki definira pogled:

FROM ColnRez



FROM coln c, rezervacija r

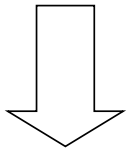
View ColnRez

```
CREATE VIEW ColnRez (cid, ime, stRez)
AS (
  SELECT c.cid, c.ime, COUNT(*) AS Cnt
  FROM coln c, rezervacija r
  ... );
```

Izvedba pogleda...

(III) WHERE sklop SELECT stavka se konjunktivno združi z WHERE sklopom iz SELECT stavka, ki definira pogled:

WHERE c.cid = 102



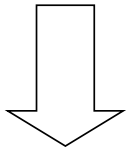
WHERE c.cid = 102 **AND**
c.cid = r.cid

View ColnRez

```
CREATE VIEW ColnRez (cid, ime, stRez)
AS (
  SELECT c.cid, c.ime, COUNT(*) AS Cnt
  FROM coln c, rezervacija r
  WHERE c.cid = r.cid AND c.cid=102
  GROUP BY c.cid, c.ime);
```

Izvedba pogleda...

(IV) GROUP BY in HAVING sklop se preneseta iz SELECT stavka, ki definira pogled:



GROUP BY c.cid, c.ime

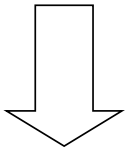
View CoInRez

```
CREATE VIEW CoInRez (cid, ime, stRez)
AS (
  SELECT c.cid, c.ime, COUNT(*) AS Cnt
  FROM coln c, rezervacija r
  WHERE c.cid = r.cid AND c.cid=102
  GROUP BY c.cid, c.ime);
```

Izvedba pogleda...

(V) ORDER BY se kopira iz SELECT stavka. Imena stolpcev se zamenjajo z imeni stolpcev iz SELECT stavka, ki definira pogled

ORDER BY ime



ORDER BY c.ime

View CoInRez

```
CREATE VIEW CoInRez (cid, ime, stRez)
AS (
  SELECT c.cid, c.ime, COUNT(*) AS Cnt
  FROM coln c, rezervacija r
  WHERE c.cid = r.cid AND c.cid=102
  GROUP BY c.cid, c.ime
  ORDER BY c.ime
);
```

Izvedba pogleda...

Po prevedbi in preimenovanju atributov dobimo naslednji SELECT stavek:

```
SELECT c.cid AS cid, c.ime AS ime, COUNT(*) AS stRez
FROM coln c, rezervacija r
WHERE c.cid = 102 AND
        c.cid = r.cid
GROUP BY c.cid, c.ime
ORDER BY c.ime;
```

```
+-----+------+-------+
| cid | ime  | stRez |
+-----+------+-------+
| 102 | Elan |     3 |
+-----+------+-------+
```

Omejitve pogledov...

- Preko pogledov lahko tudi spreminjamo podatke.
- Pri kreiranju in uporabi pogledov veljajo določene omejitve, predvsem v zvezi s spreminanjem
- Omejitve so v veliki meri odvisne od konkretnega SUPB, ki ga uporabljamo in so tipično precej obsežne in zapletene (primer: MySQL)

MySQL: Updatable and Insertable Views

19.5.3. Updatable and Insertable Views

Some views are updatable. That is, you can use them in statements such as UPDATE, DELETE, or INSERT to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable. To be more specific, a view is not updatable if it contains any of the following:

- Aggregate functions (SUM(), MIN(), MAX(), COUNT(), and so forth)
- DISTINCT
- GROUP BY
- HAVING
- UNION or UNION ALL
- Subquery in the select list
- Certain joins (see additional join discussion later in this section)
- Nonupdatable view in the FROM clause
- A subquery in the WHERE clause that refers to a table in the FROM clause
- Refers only to literal values (in this case, there is no underlying table to update)
- Uses ALGORITHM = TEMPTABLE (use of a temporary table always makes a view nonupdatable)
- Multiple references to any column of a base table.

With respect to insertability (being updatable with INSERT statements), an updatable view is insertable if it also satisfies these additional requirements for the view columns:

- There must be no duplicate view column names.
- The view must contain all columns in the base table that do not have a default value.
- The view columns must be simple column references and not derived columns. A derived column is one that is not a simple column reference but is derived from an expression. These are examples of derived columns:
- 3.14159 col1 + 3 UPPER(col2) col3 / col4 (*subquery*)

A view that has a mix of simple column references and derived columns is not insertable, but it can be updatable if you update only those columns that are not derived. Consider this view:

- CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;

This view is not insertable because col2 is derived from an expression. But it is updatable if the update does not try to update col2. This update is permissible:

- UPDATE v SET col1 = 0;This update is not permissible because it attempts to update a derived column:
- UPDATE v SET col2 = 0;It is sometimes possible for a multiple-table view to be updatable, assuming that it can be processed with the MERGE algorithm. For this to work, the view must use an inner join (not an outer join or a UNION). Also, only a single table in the view definition can be updated, so the SET clause must name only columns from one of the tables in the view. Views that use UNION ALL are not permitted even though they might be theoretically updatable, because the implementation uses temporary tables to process them.
- For a multiple-table updatable view, INSERT can work if it inserts into a single table. DELETE is not supported.
- INSERT DELAYED is not supported for views.
- If a table contains an AUTO_INCREMENT column, inserting into an insertable view on the table that does not include the AUTO_INCREMENT column does not change the value of LAST_INSERT_ID(), because the side effects of inserting default values into columns not part of the view should not be visible.
- The WITH CHECK OPTION clause can be given for an updatable view to prevent inserts or updates to rows except those for which the WHERE clause in the *select_statement* is true.
- In a WITH CHECK OPTION clause for an updatable view, the LOCAL and CASCADED keywords determine the scope of check testing when the view is defined in terms of another view. The LOCAL keyword restricts the CHECK OPTION only to the view being defined. CASCADED causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is CASCADED. Consider the definitions for the following table and set of views:
- mysql> **CREATE TABLE t1 (a INT);** mysql> **CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2** -> **WITH CHECK OPTION;** mysql> **CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0** -> **WITH LOCAL CHECK OPTION;** mysql> **CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0** -> **WITH CASCADED CHECK OPTION;** Here the v2 and v3 views are defined in terms of another view, v1. v2 has a LOCAL check option, so inserts are tested only against the v2 check. v3 has a CASCADED check option, so inserts are tested not only against its own check, but against those of underlying views. The following statements illustrate these differences:

mysql> **INSERT INTO v2 VALUES (2);** Query OK, 1 row affected (0.00 sec) mysql> **INSERT INTO v3 VALUES (2);** ERROR 1369 (HY000): CHECK OPTION failed 'test.v3' MySQL sets a flag, called the view updatability flag, at CREATE VIEW time. The flag is set to YES (true) if UPDATE and DELETE (and similar operations) are legal for the view. Otherwise, the flag is set to NO (false). The IS_UPDATABLE column in the INFORMATION_SCHEMA.VIEWS table displays the status of this flag. It means that the server always knows whether a view is updatable. If the view is not updatable, statements such UPDATE, DELETE, and INSERT are illegal and will be rejected. (Note that even if a view is updatable, it might not be possible to insert into it, as described elsewhere in this section.)

- The updatability of views may be affected by the value of the updatable_views_with_limit system variable. See Section 5.1.4, "Server System Variables".

Spreminjanje vsebine pogledov...

- Vse spremembe nad osnovnimi relacijami so takoj vidne tudi v pogledih nad temi relacijami.

- Želeli bi tudi obratno: če spremenimo podatke v pogledu, se morajo spremembe poznati tudi v osnovnih relacijah, na katere se te spremembe nanašajo.

Spreminjanje vsebine pogledov...

- Žal ni povsem transparentno, kajti v pogledih niso možne vse spremembe. V splošnem veljajo naslednje omejitve:
  - Nad pogledom so možne spremembe, če pogled zajema eno samo osnovno relacijo ter vključuje atribute, ki so del kandidata za ključ relacije.
  - Če pogled zajema več relacij, spremembe običajno niso možne.
  - Če je pogled pridobljen kot povzetek (agregacija) več vrstic, spremembe niso možne.

Spreminjanje in omejitve pogledov...

- Poglede lahko tudi spreminjamo (INSERT, UPDATE, DELETE), vendar spreminjanje vsebine ni vedno možno
- V resnici se spremenijo originalne tabele, zato tu veljajo mnoge omejitve

```
INSERT INTO ColnRez
    VALUES (105, 'Titanik', 1316);
```

- V osnovno tabelo rezervacija bi morali dodati 1316 rezervaciji, v tabelo coln pa nov vnos, ki se nanaša na čoln Titanik. Nimamo dovolj podatkov!

```
CREATE VIEW ColnRez
            (cid, ime, stRez)
AS (
  SELECT c.cid, c.ime,
            COUNT(*) AS Cnt
  FROM coln c, rezervacija r
  WHERE c.cid = r.cid
  GROUP BY c.cid, c.ime
);
```

Omejitve pogledov...

- ISO standard določa, da je pogled možno spreminjati samo, če veljajo naslednji pogoji:
  - Opcija DISTINCT v definiciji pogleda ni uporabljena;
  - Vsak element v SELECT seznamu stavka, ki definira pogled, se nanaša na stolpec ene izmed osnovnih tabel; noben stolpec se ne pojavi več kot enkrat;
  - FROM sklop v pogledu se nanaša samo na eno tabelo ali pogled, pri čemer pogled ne sme temeljiti na stiku, uniji, preseku ali razliki;
  - V definiciji pogleda ni vgnezdenih poizvedb;
  - Sklopa GROUP BY in HAVING v pogledu nista uporabljena;
- V večini primerov se spreminjanje opravlja nad projekcijo atributov neke tabele.

Spreminjanje pogleda in WITH CHECK OPTION

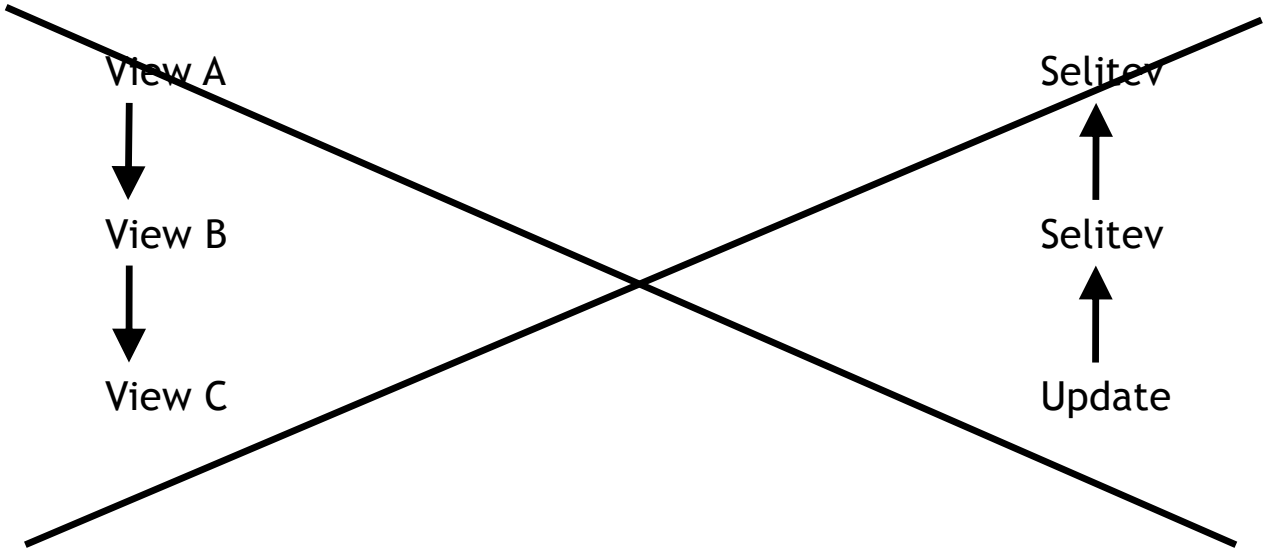
- Vrstice v pogledu obstajajo, ker izpolnjujejo WHERE pogoj SELECT stavka, ki pogled definira.
- Če se vrstica spremeni in ne zadošča več pogoju, izgine iz pogleda.
- Nove vrstice se v pogledu pojavijo le, če zadoščajo WHERE pogoju.
- Vrstice, ki vstopijo ali zapustijo pogled, imenujemo selitvene vrstice (migrating rows).
- WITH CHECK OPTION prepoveduje selitev vrstic v ali iz pogleda kot posledico vstavljanja/spreminjanja v pogledu.

Uporaba WITH CHECK OPTION...

- Brez uporabe WITH CHECK OPTION:
  - v pogled je dovoljeno vstavljati vrstice, ki ne zadoščajo kriteriju pogleda (takoj po vstavljanju ga zapustijo)
- Z uporabo WITH CHECK OPTION:
  - LOCAL: vrstice za vstavljanje morajo zadoščati pogoju definicije pogleda, v katerega vstavljamo. Pogojev pogledov, iz katerih je izpeljan trenutni pogled, se ne preverja.
  - CASCADED (privzeto): vrstice za vstavljanje morajo zadoščati tako pogoju definicije pogleda, v katerega vstavljamo, kot pogojem vseh pogledov, iz katerih je izpeljan.

Uporaba WITH CHECK OPTION...

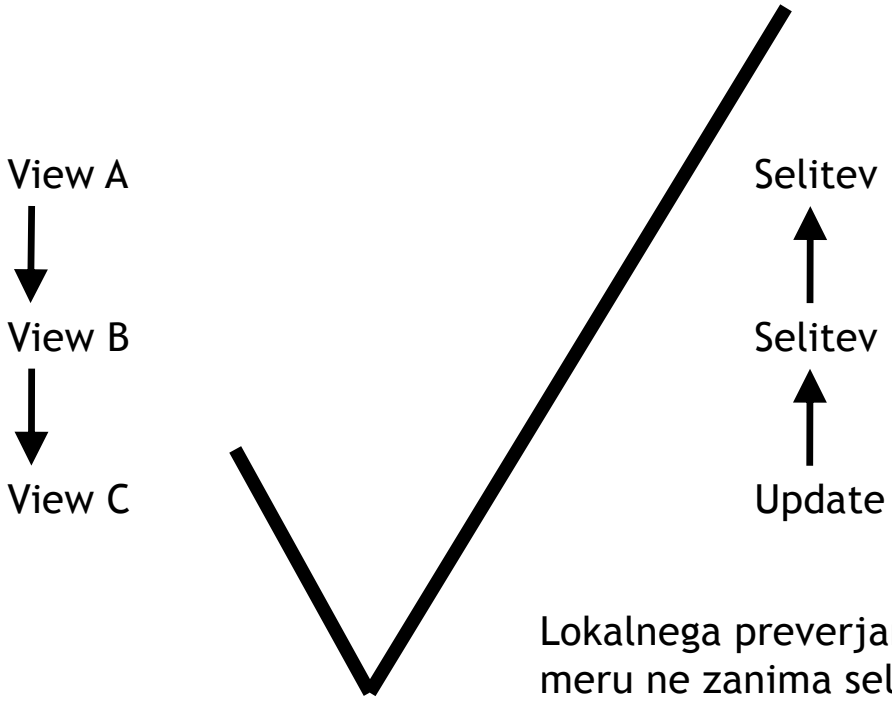
- WITH CASCADED CHECK OPTION (privzeto)



Selitev pomeni, da je kršen pogoj definicije relevantnega pogleda. Kaskadno preverjanje v tem primeru zazna selitev vstavljene vrstice iz pogledov A in B.

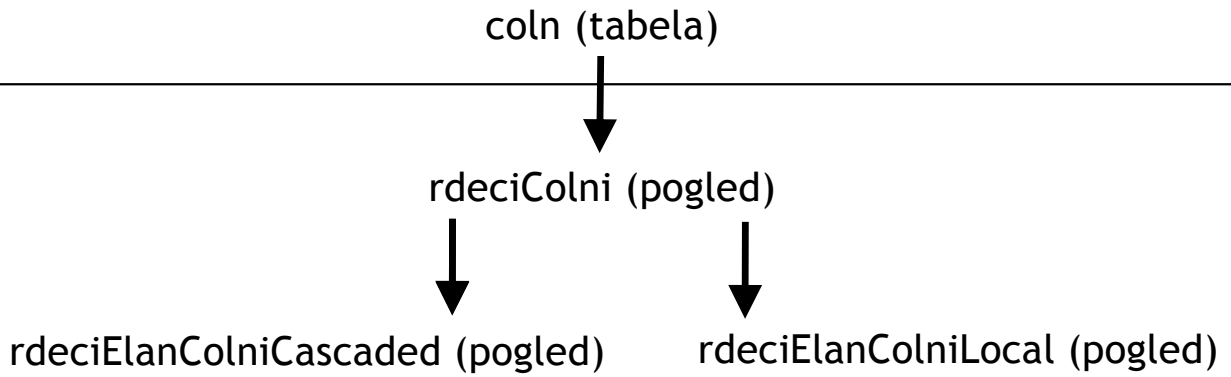
Uporaba WITH CHECK OPTION...

- WITH LOCAL CHECK OPTION



Lokalnega preverjanja v tem primeru ne zanima selitev vstavljene vrstice iz pogledov A in B.

Primer



- V pogledu rdeciColniCascaded:
  - Ne moremo spremeniti barve čolna, ker bi s tem povzročili selitev vrstice iz pogleda
  - Ne moremo vnesti novega čolna, ki ne bi bil rdeče barve, v pogled.
- V pogledu rdeciColniLocal:
  - lahko počnemo oboje

Prednosti in slabosti pogledov

- PREDNOSTI
  - Podatkovna neodvisnost
  - Ažurnost
  - Večja varnost
  - Manjša kompleksnost
  - Udobnost
  - Prilagodljivost
  - Podatkovna celovitost
- SLABOSTI:
  - Omejitve spreminjanja
  - Omejitve strukture
  - Slabša učinkovitost

Materializirani pogledi

- Postopek za izvedbo pogleda je lahko počasen, še posebej pri pogostem dostopanju do pogleda.
- Materializirani pogled ob prvi uporabi shrani pogled kot začasno tabelo za kasnejšo rabo.
- Ob kasnejši rabi večja učinkovitost.
- Problem vzdrževati ažurno stanje, če se osnovne tabele pogosto spreminjajo.
- PostgreSQL, Oracle (MariaDB in MySQL ne podpirata):
  CREATE MATERIALIZED VIEW …

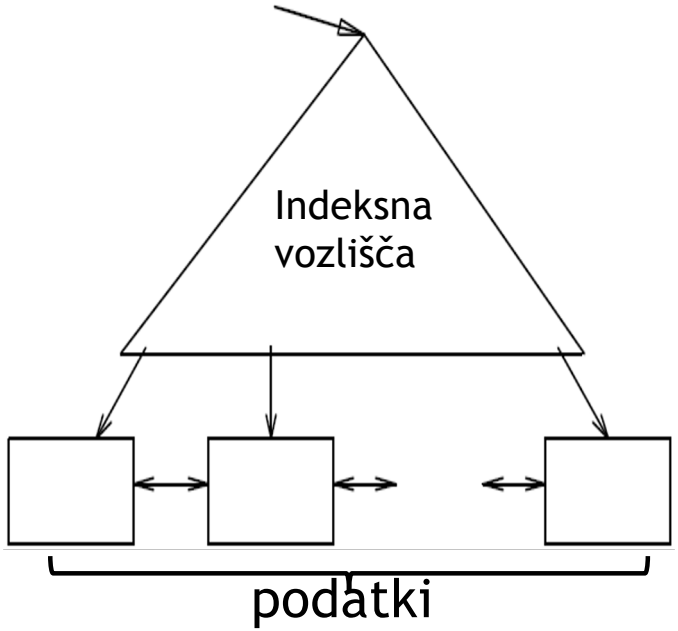
Vzdrževanje materializiranega pogleda

- Zagotavljanje ažurnosti v materializiranem pogledu
  - Avtomatsko: prenašajo se samo tiste spremembe, ki so potrebne, da je pogled ažuren (Microsoft SQL Server indexed views, Oracle z REFRESH FAST ON COMMIT).
    - Zahteve po dodatnih podatkovnih strukturah, REFRESH LOG (Oracle)
  - Na zahtevo (Oracle, PostgreSQL, nestandardno): REFRESH MATERIALIZED VIEW
  - Periodično (Oracle), npr. vsako uro
  - Zahtevno, vprašljiva podpora različnih SUPB.
- Alternativa: običajna tabela, kjer pa je potrebno programsko zagotavljanje ažurnosti (npr. z baznimi prožilci – triggerji)

Indeksi in SQL

- Indeks je pomožna podatkovna struktura, namenjena hitrejšim operacijam (iskanje,dodajanje, brisanje, spreminjanje) nad tabelo.
- Indeks nudi urejen pogled na tabelo (po določenem atributu ali množici atributov).
- Datotečne organizacije (in podpora za indekse):
  - Metoda indeksiranega zaporednega dostopa (Indexed Sequential Access Method - ISAM)
  - B+ drevesno indeksiranje (najpogostejše)
  - razpršene tabele (hash): dobro za nespremenljive tabele
  - bitni in stični indeksi: podatkovna skladišča
  - gručni (cluster) indeksi: v indeksu hranijo cele vrstice tabele
- Mnogi SUPB ne podpirajo vseh naštetih možnosti.

Drevesna zgradba B/B+ indeksa

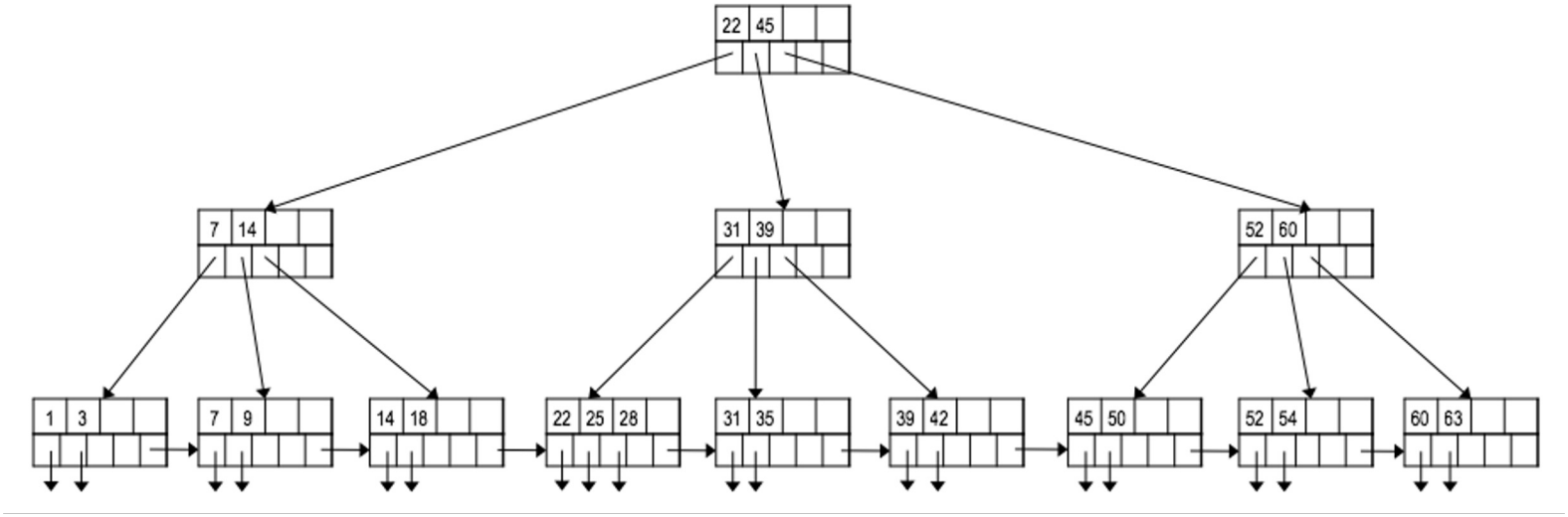


- Vsako indeksno vozlišče (razen eventuelno korena) vsebuje $d \leq m \leq 2d$ indeksnih zapisov potomcev).
- Vsako vozlišče ima kapaciteto $2d$ in je torej vedno vsaj pol polno.
- Alternativa: fiksna velikost vozlišča (blok)
- Parametru d pravimo red indeksa
- Globina indeksa g : razdalja (število indeksnih nivojev) od korena do listov (nivoja z listi ne štejemo):
  max g = $O(\log_d(N))$

- Indeks bistveno pospeši dostop do vrstic tabele
- Zaporedno preiskovanje n vrstic: s $t_1=O(n)$. Indeks: $t_2=O(\log_d n)$
- Pri n=1 milijon in d=10 je $t_1=O(100000)$, t_2=O(6)
- Primerjava: 300 ur proti nekaj sekundam

Primer B/B+ indeksa

- Red drevesa: 2
- Vrednosti (urejene): 1, 3, 7, 9, 14, 18, 22, 25, 28, 31, 35, 39, 42, 45, 50, 52, 54, 60, 63
- (a) poišči 35 (b) poišči 55



Izbira atributov za indeksiranje

- Primarni indeks: indeks po primarnem ključu.
- Sekundarni indeks: indeks po atributu, ki ni primarni ključ (lahko jih imamo več).
- Ali lahko z dodatnimi sekundarnimi indeksi povečamo učinkovitost sistema?
- Uporaba vseh možnih indeksov ni smiselna (za N atributov 2^N možnih indeksov).
- Možen pristop:
  - Zapise uredimo (samo po enem kriteriju!)
  - Zapise pustimo neurejene in izdelamo toliko sekundarnih indeksov, kolikor je potrebno.

Izbira atributov za indeksiranje

- Sekundarni indeksi so način, kako omogočiti učinkovito iskanje s pomočjo dodatnih (iskalnih) ključev.
- Pri določanju sekundarnih indeksov tehtamo:
  - Povečanje učinkovitosti (predvsem pri iskanju po PB)
  - Dodatno delo (in poraba prostora), ki ga mora sistem opravljati za vzdrževanje indeksov. To vključuje:
    - Dodajanje zapisa v vsak sekundarni indeks, kadarkoli dodamo nek zapis v osnovno relacijo
    - Spreminjanje sekundarnega indeksa vsakokrat, ko se osnovna relacija spremeni
    - Povečanje porabe prostora v sekundarnem pomnilniku
    - Povečanje časovnega obsega za optimizacijo poizvedb zaradi preverjanja vseh sekundarnih indeksov.

Izbira atributov za indeksiranje

- Smernice za uporabo sekundarnih indeksov:
  - Ne indeksiraj majhnih relacij.
  - Če nimamo primarnega indeksa ali tabela (kot datoteka) ni urejena po primarnem ključu, potem kreiraj indeks na osnovi primarnega ključa.
  - Če je tuji ključ pogosto v uporabi, dodaj sekundarni indeks na tuji ključ.
  - Sekundarne indekse dodaj atributom, ki nastopajo v pogojih za selekcijo ali stik, ORDER BY, GROUP BY ali v drugih operacijah, ki vključujejo sortiranje (npr. UNION ali DISTINCT).

Izbira atributov za indeksiranje

- Smernice za uporabo sekundarnih indeksov:
  - Dodaj sekundarni indeks atributom, po katerih se izvajajo iskanja ali filtriranja
  - Izogibaj se indeksiranju atributov, ki se pogosto spreminjajo.
  - Izogibaj se indeksiranju atributov v relacijah, nad katerimi se bodo pogosto izvajale poizvedbe, ki bodo vključevale večji del zapisov.
  - Izogibaj se indeksiranju atributov, ki so predstavljeni z daljšimi nizi znakov (manj učinkovito, omejitve dolžine).
  - Napredno: uporaba polnotekstovnih (FULL TEXT) indeksov
    - Učinkovito iskanje po besedilih v naravnem jeziku in z vzorci

SQL: kreiranje in brisanje indeksov

- Kreiranje indeksov:

```
CREATE [UNIQUE] INDEX ime_indeksa
ON ime_tabele (ime_atributa1 [ASC|DESC],
               ime_atributa2 [ASC|DESC],
               ... );
```

- Indeks se gradi po kombinaciji vrednosti atributov; za vsako kombinacijo atributov potrebujemo svoj indeks
- Možna specifikacija tipa indeksa (npr. BTREE)
- Brisanje nepotrebnih indeksov:

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

SQL: uporaba indeksiranja

- Indeksiraj čolne po barvi!

```
CREATE INDEX po_barvi
ON coln(barva);
```

- Indeksiraj jadralce po šifrah in ratingih skupaj!

```
CREATE INDEX po_jid_rating
ON jadralec(jid DESC, rating DESC);
```

- Upraba indeksa je običajno avtomatska (SUPB izbere najprimernejšega), lahko pa tudi ročna: sami navedemo, katerega naj uporabi (namigi - nestandardno).

SQL: nadzor indeksiranja

- Filtrirani (delni) indeksi: z logičnim pogojem povemo, katere vrstice naj se indeksirajo (npr. manjši del tabele, ki pa je pogosto uporabljan).
- Izbor tip indeksa (privzeto BTREE):
  - USING tip_indeksa
  - MySQL, PostgreSQL: USING BTREE ali HASH (omejeni tipi tabel)
  - PostgreSQL: tudi GIST-geometrični, SPGIST-prostorski, GIN-za večvrednostne ali strukturirane atribute

```
CREATE INDEX po_barvi_hash_filter
ON coln USING HASH(barva)
WHERE cid IN (101, 102);
```

Shranjeni podprogrami v SQL

- Shranjeni podprogrami: procedure in funkcije, ki jih pogosto potrebujemo
- Poimenovani SQL bloki, ki jih lahko kličemo s parametri
- Lahko spreminjajo podatke ali vračajo rezultate
- Funkcija: vrne natanko eno vrednost kot rezultat
- Procedura: vrača vrednost v izhodnih argumentih
- Omogočajo modularno in razširljivo pisanje programov
- Žal so implementacije pogosto sistemsko odvisne.

Shranjeni podprogrami v SQL

- Parametri (predvsem v procedurah)
  - vhodni (IN), izhodni(OUT)
  - vhodno-izhodni (IN OUT)
- Pogosto potrebna uporaba postopkovnih dodatkov (spremenljivke, kurzorji, ...)
  - ISO/ANSI: SQL/PSM (Persistant Stored Modules).
  - PostgreSQL: PL/pgSQL, Oracle: PL/SQL, Microsoft: T-SQL
  - MySQL, IBM DB2: najbližja standardu SQL/PSM
- Deklaracija in uporaba podprogramov

```
CREATE PROCEDURE
      Test (a IN VARCHAR(10)) AS … ;

CALL ali EXECUTE Test('abcd');

DROP PROCEDURE Test;
```

Primer izračuna shranjenega atributa

- V tabelo jadralec dodamo število rezervacij za vsakega jadralca.

```
ALTER TABLE jadralec
      ADD stRez  INTEGER DEFAULT  0 NOT NULL;
```

- Kako (in kdaj) izračunamo vrednost tega atributa?

```
UPDATE jadralec j
       SET stRez =
              (SELECT COUNT(*)
               FROM rezervacija r
               WHERE r.jid = j.jid);
```

Kdaj je vse to zares potrebno izračunati?

Primer procedure (Oracle)

- Inicializiraj število rezervacij na poljubno vrednost (parameter).

```
CREATE PROCEDURE JADR_REZ_INIT
( INIT IN INTEGER DEFAULT 0 ) AS
BEGIN
   UPDATE jadralec j
     SET stRez = INIT;
END;

CALL JADR_REZ_INIT(0);
```

Primer procedure (MariaDB/MySQL)

- Inicializiraj število rezervacij na poljubno vrednost (parameter).

```
DELIMITER //
CREATE PROCEDURE JADR_REZ_INIT
(IN INIT INTEGER)
BEGIN
   UPDATE jadralec j
     SET stRez = INIT;
END//
DELIMITER ;
CALL JADR_REZ_INIT(0);
```

Primer procedure (PostgreSQL)

- Pozna le funkcije; procedura je funkcija, ki vrača tip **VOID**
- Klic procedure (funkcije) v stavku SELECT

```
CREATE FUNCTION JADR_REZ_INIT
( INIT IN INTEGER DEFAULT 0 ) RETURNS VOID AS
$telo$
BEGIN
   UPDATE jadralec j
   SET stRez = INIT;
END;
$telo$ LANGUAGE plpgsql;
```

```
-- Klic:

SELECT JADR_REZ_INIT(0);
```

Primer procedure (Oracle)

- Izračunaj dejansko število rezervacij.

```
CREATE PROCEDURE JADR_REZ AS
BEGIN
  UPDATE jadralec j
    SET stRez =
          (SELECT COUNT(*)
           FROM rezervacija r
           WHERE r.jid = j.jid);
END;

CALL JADR_REZ();
```

Primer procedure (MariaDB/MySQL)

- Izračunaj dejansko število rezervacij.

```
DELIMITER //
CREATE PROCEDURE JADR_REZ()
BEGIN
  UPDATE jadralec j
    SET stRez =
          (SELECT COUNT(*)
           FROM rezervacija r
           WHERE r.jid = j.jid);
END//

CALL JADR_REZ();
```

Primer procedure (PostgreSQL)

- Izračunaj dejansko število rezervacij.

```
CREATE FUNCTION JADR_REZ()
    RETURNS VOID AS
$telo_funkcije$
BEGIN
    UPDATE jadralec j
        SET stRez =         (SELECT COUNT(*)
                             FROM rezervacija r
                             WHERE r.jid = j.jid);
END;
$telo_funkcije$ LANGUAGE plpgsql;
```

```
-- Klic:

SELECT JADR_REZ();
```

Primer procedure in funkcije (Oracle)

- Izračunaj dejansko število rezervacij.

```
CREATE FUNCTION JADR_REZ_FUNC
( JJID IN INTEGER) RETURN INTEGER AS
x INTEGER;  -- Lokalna spremenljivka
BEGIN
  SELECT COUNT(*) INTO x
  FROM rezervacija r
  WHERE r.jid = jjid;
  RETURN x;
END;
```

```
CREATE PROCEDURE JADR_REZ AS
BEGIN
  UPDATE jadralec j
     SET stRez = JADR_REZ_FUNC(j.jid);
END;
```

Primer procedure in funkcije (MariaDB/MySQL)

- Izračunaj dejansko število rezervacij.

```
DELIMITER //
CREATE FUNCTION JADR_REZ_FUNC
( JJID INTEGER) RETURNS INTEGER
BEGIN
  DECLARE x INTEGER;            -- Lokalna spremenljivka
  SELECT COUNT(*) INTO x
     FROM rezervacija r
     WHERE r.jid = jjid;
  RETURN x;
END//
```

```
CREATE PROCEDURE JADR_REZ()
BEGIN
  UPDATE jadralec j
     SET stRez = JADR_REZ_FUNC(j.jid);
END//
```

Primer procedure in funkcije (PostgreSQL)

```
CREATE FUNCTION JADR_REZ_FUNC
    (JJID IN INTEGER)  RETURNS INTEGER AS
$tf1$
DECLARE
  x INTEGER;  -- Lokalna spremenljivka
BEGIN
  SELECT COUNT(*) INTO x
  FROM rezervacija r
  WHERE r.jid = jjid;
  RETURN x;
END;
$tf1$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION JADR_REZ()
          RETURNS VOID AS
$tf2$
BEGIN
  UPDATE jadralec j
     SET stRez = JADR_REZ_FUNC(j.jid);
END;
$tf2$ LANGUAGE plpgsql
```

Bazni prožilci (triggerji)

- Prožilec: sestavljen SQL stavek, podobne oblike kot shranjena procedura, vendar nima argumentov
- Izvede se avtomatsko kot stranski produkt spremembe neke poimenovane tabele
- Ne kličemo ga ročno, ampak ga sproži prožilni dogodek
- Uporaba:
  - preverjanje pravilnosti vnosev in integritetnih omejitev (tudi denormalizacija)
  - opozarjanje na potrebne uporabniške akcije ob spremembah
  - vzdrževanje seznamov sprememb v PB
- Stavčni in vrstični prožilci.

Sintaksa prožilcev dogodkov nad tabelami

- ISO standard:

```
CREATE TRIGGER
BEFORE | AFTER dogodek ON tabela
[REFERENCING sinonimi za stare ali nove vrednosti]
[FOR EACH ROW]
[WHEN (pogoj)]    -- pogoj za vrstico (kot WHERE)
BEGIN
   -- telo prožilca
END;
```

- Dogodki: INSERT, UPDATE, DELETE

Stavčni prožilci

- Stavčni prožilec se izvede le enkrat na stavek, ki spremeni tabelo
- Oracle: stavčni prožilci so privzeti.
- MariaDB/MySQL: ne podpirata stavčnih prožilcev (samo vrstične).
- PostgreSQL: stavčni prožilci so privzeti.

Primer stavčnega prožilca (Oracle)

```
CREATE TRIGGER IzracunajSteviloRezervacij
AFTER INSERT OR UPDATE OR DELETE ON rezervacija
-- za vsak stavek (ne dela v MySQL)
BEGIN            -- PL/SQL blok
   UPDATE jadralec
      SET stRez =
            ( SELECT COUNT(*)
              FROM rezervacija
              WHERE rezervacija.jid = jadralec.jid)
END;
```

Primer stavčnega prožilca (PostgreSQL)

- V PostgreSQL prožilci nimajo definiranega telesa, ampak lahko le kličejo vnaprej definirane *prožilne funkcije*.
  - Prožilne funkcije nimajo argumentov in vračajo tip TRIGGER
  - V prožilnih funkcijah se ob klicu ustvarijo prožilne spremenljivke (NEW, OLD, ...)

```
CREATE FUNCTION JADR_REZ_TRIG()
   RETURNS TRIGGER AS ...

CREATE TRIGGER IzracunajSteviloRezervacij
AFTER INSERT OR UPDATE OR DELETE ON rezervacija
FOR EACH STATEMENT        --  Privzeto
EXECUTE PROCEDURE  JADR_REZ_TRIG();
```

Vrstični prožilci

- Vrstični prožilec se izvede za vsako spremenjeno vrstico
- Odvisno od vrste dogodka lahko referenciramo
  - stare vrednosti pred spremembo (OLD): DELETE, UPDATE
  - nove vrednosti po spremembi (NEW): INSERT, UPDATE
  - Oracle: v WHEN sklopu OLD in NEW uporabljamo normalno, znotraj BEGIN/END pa z dvopičjem :OLD, :NEW
  - Oracle: z REFERENCING sklopom lahko OLD in NEW preimenujemo
- Prednost vrstičnih prožilcev: izvedemo telo prožilcev samo za vrstice, ki so se zares spremenile
- Nerodno: pogosto moramo za vsako vrsto dogodka napisati svoj prožilec (zelo podoben ostalim).

Primer vrstičnega prožilca (INSERT)

```
CREATE TRIGGER IzracunajSteviloRezervacij_I
AFTER INSERT ON rezervacija
REFERENCING NEW AS nova            -- Alias za NEW
FOR EACH ROW    -- za vsako novo vrstico
BEGIN   -- PL/SQL blok
   UPDATE jadralec
   SET stRez = stRez +1
   WHERE jadralec.jid = :nova.jid;
END;
```

Primer vrstičnega prožilca (DELETE)

```
CREATE TRIGGER IzracunajSteviloRezervacij_D
AFTER DELETE ON rezervacija
REFERENCING OLD AS stara              -- Alias za OLD
FOR EACH ROW     -- za vsako novo vrstico
BEGIN   -- PL/SQL blok
    UPDATE jadralec
    SET stRez = stRez -1
    WHERE jadralec.jid = :stara.jid;
END;
```

Primer vrstičnega prožilca (UPDATE)

```
CREATE TRIGGER IzracunajSteviloRezervacij_U
AFTER UPDATE ON rezervacija
REFERENCING OLD AS stara NEW AS nova
FOR EACH ROW    -- za vsako novo vrstico
WHEN (stara.jid != nova.jid)
BEGIN   -- PL/SQL blok
    UPDATE jadralec
    SET stRez = stRez +1
    WHERE jadralec.jid = :nova.jid;
    UPDATE jadralec
    SET stRez = stRez -1
    WHERE jadralec.jid = :stara.jid;
END;
```

MySQL/MariaDB shranjene procedure in prožilci

- Spremenimo ločilo za konec stavka (namesto podopičja):
  npr. DELIMITER //
- Razlike pri parametrih: IN, OUT, INOUT pred imenom
  npr. (IN INIT INTEGER) samo za procedure, funkcije imajo le IN argumente, ni privzetih vrednosti
- Deklaracija lokalnih spremenljivk znotraj BEGIN/END:
  npr. DECLARE x INTEGER;
- Ne uporablja AS, RETURNS namesto RETURN
- Samo en dogodek na prožilec (nima OR)
- Ni stavčnih prožilcev, ni aliasov za OLD in NEW
- Ne uporabljamo dvopičja: OLD namesto :OLD
- Ne pozna WHEN sklopa (lahko pa uporabimo proceduralni IF/END IF sklop)

MySQL: primer vrstičnega prožilca (INSERT)

```
DELIMITER //
CREATE TRIGGER IzracunajSteviloRezervacij_I
AFTER INSERT ON rezervacija
FOR EACH ROW    -- za vsako novo vrstico
BEGIN
   UPDATE jadralec
   SET stRez = stRez +1
   WHERE jadralec.jid = NEW.jid;
END//
```

MySQL: primer vrstičnega prožilca (DELETE)

```
DELIMITER //
CREATE TRIGGER IzracunajSteviloRezervacij_D
AFTER DELETE ON rezervacija
FOR EACH ROW    -- za vsako novo vrstico
BEGIN
   UPDATE jadralec
   SET stRez = stRez -1
   WHERE jadralec.jid = OLD.jid;
END//
```