Upitni Jezici

- Relaciona Algebra
- **Relacioni Račun**
  - **Relacioni Račun sa Torkama** (Tuple Relational Calculus)
  - **Relacioni Račun sa Domenima** (Domain Relational Calculus)
- SQL

Relacioni Račun sa Torkama

Neproceduralni upitni jezik, u kome svaki upit ima oblik

$$\{ t \mid P(t) \}$$

To je skup svih torki t takvih da je predikat P istinit za t

- t je promenljiva torka, a $t[A]$ označava vrednost torke t na atributu A
- $t \in r$ označava da je torka t u relaciji r
- P je ***formula predikatnog računa***

Formula Predikatnog Računa

1. Skup atributa i konstanti
2. Skup operatora poredjenja: ($<$, $\leq$, $=$, $\neq$, $>$, $\geq$)
3. Skup logičkih operacija: i ($\wedge$), ili (v), ne ($\neg$)
4. Implikacija ($\Rightarrow$):

   x $\Rightarrow$ y, ako je x istinito, tada je i y istinito

   $x \Rightarrow y \equiv \neg x \vee y$
5. Skup kvantifikatora:
   - $\exists\, t \in r\,(Q\,(t\,)) \equiv$ "postoji neka" torka t u relaciji r takva da je predikat $Q\,(t\,)$ istinit
   - $\forall t \in r\,(Q\,(t\,)) \equiv Q$ je istinito "za svaku" torku t u relaciji r

Primeri (1)

- Find the *loan_number, branch_name,* and *amount* for loans of over $1200

$$\{t \mid t \in loan \wedge t\,[amount\,] > 1200\}$$

- Find the loan number for each loan of an amount greater than $1200

$$\{t \mid \exists\, s \in \text{loan}\,(t\,[loan\_number\,] = s\,[loan\_number\,] \wedge s\,[amount\,] > 1200)\}$$

Primetimo da je relacija na šemi [*loan_number*] implicitno definisana upitom

Primeri (2)

- Find the names of all customers having a loan, an account, or both at the bank

$$\{t \mid \exists s \in \textit{borrower} \,(\, t\,[\textit{customer\_name}\,] = s\,[\textit{customer\_name}\,])$$
$$\vee \exists u \in \textit{depositor}\,(\, t\,[\textit{customer\_name}\,] = u\,[\textit{customer\_name}\,])$$

- Find the names of all customers who have a loan and an account at the bank

$$\{t \mid \exists s \in \textit{borrower} \,(\, t\,[\textit{customer\_name}\,] = s\,[\textit{customer\_name}\,])$$
$$\wedge \exists u \in \textit{depositor}\,(\, t\,[\textit{customer\_name}\,] = u\,[\textit{customer\_name}]\,)$$

Primeri (3)

- Find the names of all customers having a loan at the Perryridge branch

$$\{t \mid \exists s \in \textit{borrower}\ (t[\textit{customer\_name}] = s[\textit{customer\_name}] \wedge \exists u \in \textit{loan}\ (u[\textit{branch\_name}] = \text{“Perryridge”} \wedge u[\textit{loan\_number}] = s[\textit{loan\_number}]))\}$$

- Find the names of all customers who have a loan at the Perryridge branch, but no account at any branch of the bank

$$\{t \mid \exists s \in \textit{borrower}\ (t[\textit{customer\_name}] = s[\text{customer\_name}] \wedge \exists u \in \textit{loan}\ (u[\textit{branch\_name}] = \text{“Perryridge”} \wedge u[\textit{loan\_number}] = s[\text{loan\_}\textit{number}])) \wedge \textbf{not}\ \exists v \in \textit{depositor}\ (v[\textit{customer\_name}] = t[\textit{customer\_name}])\}$$

Primeri (4)

- Find the names of all customers having a loan from the Perryridge branch, and the cities in which they live

$$\{t \mid \exists s \in loan\ (s[branch\_name] = \text{“Perryridge”} \wedge \exists u \in borrower\ (u[loan\_number] = s[loan\_number] \wedge t[customer\_name] = u[customer\_name]) \wedge \exists v \in customer\ (u[customer\_name] = v[customer\_name] \wedge t[customer\_city] = v[customer\_city]))\}$$

- Find the names of all customers who have an account at all branches located in Brooklyn:

$$\{t \mid \exists r \in customer\ (t[customer\_name] = r[customer\_name]) \wedge (\forall u \in branch\ (u[branch\_city] = \text{“Brooklyn”} \Rightarrow \exists s \in depositor\ (t[customer\_name] = s[customer\_name] \wedge \exists w \in account\ (w[account\_number] = s[account\_number] \wedge (w[branch\_name] = u[branch\_name])))))\}$$

Problem Beskonačne Relacije

- Upit u relacionom računu sa torkama može generisati **beskonačnu relaciju**.

  Na primer, rezultat upita $\{ t \mid \neg\, t \in r \}$ je beskonačna relacija ako je domen bilo kog atributa relacije r beskonačan

- Ograničavamo se na **sigurne upite**.

- Izraz $\{t \mid P(t)\}$ relacionog računa sa torkama je ***siguran*** ako se svaka komponenta t nalazi u nekoj od relacija, torki, ili konstanti koje se nalaze u P

  Primer: $\{ t \mid t[A] = 5 \vee \textbf{true} \}$ nije siguran upit --- definiše beskonačan skup sa vrednostima atributa koje se ne pojavljuju ni u jednoj relaciji, torki ili konstanti u P.

Relacioni Račun sa Domenima

Neproceduralni upitni jezik, u kome svaki upit ima oblik

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n) \}$$

- $x_1, x_2, \ldots, x_n$ predstavljaju promenljive domena
- P je ***formula predikatnog računa***

Primeri (1)

- Find the *loan_number, branch_name,* and *amount* for loans of over $1200

$$\{< l, b, a > \mid < l, b, a > \in loan \wedge a > 1200\}$$

- Find the names of all customers who have a loan of over $1200

$$\{< c > \mid \exists\, l, b, a\, (< c, l > \in borrower \wedge < l, b, a > \in loan \wedge a > 1200)\}$$

- Find the names of all customers who have a loan from the Perryridge branch and the loan amount:

  - $\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge \exists b\, (< l, b, a > \in loan \wedge b = \text{“Perryridge”}))\}$
  - $\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge < l, \text{“ Perryridge”}, a > \in loan)\}$

Primeri (2)

- Find the names of all customers having a loan, an account, or both at the Perryridge branch:

$$\{< c > \mid \exists\, l\, (< c, l > \in \textit{borrower} \wedge \exists\, b,a\, (< l, b, a > \in \textit{loan} \wedge b = \text{“Perryridge”})) \vee \exists\, a\, (< c, a > \in \textit{depositor} \wedge \exists\, b,n\, (< a, b, n > \in \textit{account} \wedge b = \text{“Perryridge”}))\}$$

- Find the names of all customers who have an account at all branches located in Brooklyn:

$$\{< c > \mid \exists\, s,n\, (< c, s, n > \in \text{customer}) \wedge \forall\, x,y,z\, (< x, y, z > \in \textit{branch} \wedge y = \text{“Brooklyn”}) \Rightarrow \exists\, a,b\, (< x, y, z > \in \textit{account} \wedge < c,a > \in \textit{depositor})\}$$

Problem Beskonačne Relacije

Izraz:

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

je ***siguran*** ako važi sledeće:

1. Sve vrednosti koje se pojavljuju u torkama izraza su vrednosti iz $dom(P)$ (to jest, vrednosti se pojavljuju u P ili u torki relacije u P).
2. Za svaku formulu oblika $\exists x(P_1(x))$, formula je istinita ako i samo ako postoji vrednost x u $dom(P_1)$ takvo da je $P_1(x)$ istinito.
3. Za svaku formulu oblika $\forall x(P_1(x))$, formula je istinita ako i samo ako je $P_1(x)$ istinito za sve vrednosti x iz $dom(P_1)$.

Upitni Jezici

- Relaciona Algebra
- Relacioni Račun
- **SQL**
  - **Definicija Podataka**
  - **Osnovna Struktura Upita**
  - **Skupovne Operacije**
  - **Agregatne Funkcije**
  - **Null Vrednosti**
  - **Ugnježđeni Upiti**
  - **Kompleksni Upiti**
  - **Pogledi**
  - **Modifikacije Baze**
  - **Spojene Relacije****

Istorijat

- IBM Sequel jezik razvijen kao deo projekta System R u IBM San Jose Research Laboratory
- Preimenovan u Structured Query Language (SQL)
- ANSI i ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (usaglašen sa Y2K)
  - SQL:2003
- Komercijalni sistemi nude najveći deo ili sve mogućnosti standarda SQL-92, plus različite mogućnosti iz kasnijih standarda plus specijalne mogućnosti.

Definicija Podataka (Data Definition)

Specificiraju se relacije i informacije o relacijama:

- Šeme relacija.
- Domeni vrednosti atributa.
- Ograničenja Integriteta
- Skupovi indeksa relacija.
- Informacije o Sigurnosti i Zaštiti relacija.
- Fizička struktura relacija na disku.

Komanda create table

- Definicija relacije se vrši komandom **create table**:

  create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - r je ime relacije
  - A_i je ime atributa u šemi relacije r
  - D_i je tip podatka vrednosti u domenu atributa A_i

  Primer:

  create table *branch*
  (*branch_name* char(15) **not null,**
  branch_city char(30),
  assets integer)

Ograničenja Integriteta u create table

- **not null**
- **primary key** (A_1, ..., A_n)

Primer: Deklaracija *branch_name* za primarni ključ relacije *branch* i obezbedjenje da vrednosti *assets* budu ne-negativne.

```
create table branch
        (branch_name    char(15),
         branch_city    char(30),
         assets         integer,
         primary key (branch_name))
```

Deklaracija **primary key** automatski obezbedjuje **not null** u standardu SQL-92 i novijim, u SQL-89 je potrebno eksplicitno zadati

Komande drop i alter table

- Komanda **drop table** briše iz baze sve informacije o zadatoj relaciji.
- Komanda **alter table** se koristi za dodavanje atributa u postojeću relaciju:

  alter table *r* **add** *A D*

  gde je *A* ime atributa koji se dodaje relaciji *r* a *D* domen atributa *A.*
  - Sve vrednosti novog atributa u relaciji su *null* .
- Komanda **alter table** se može koristiti i za eliminisanje atributa iz relacije:

  alter table *r* **drop** *A*

gde je *A* ime atributa iz relacije *r*

Osnovna Struktura Upita

- SQL se bazira na skupovnim i relacionim operacijama sa određenim modifikacijama i unapređenjima
- Tipični SQL upit je oblika:

  select A_1, A_2, ..., A_n
  from r_1, r_2, ..., r_m
  where P

  gde su: A_i atributi, r_i relacije, a P predikat.
- Ovaj upit je ekvivalentan sledećem izrazu relacione algebre:

$$\Pi_{A_1,A_2,\ldots,A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

- Rezultat SQL upita je relacija.

Komanda select (1)

- Komandom **select** zadajemo listu atributa u rezultatu upita
  - Odgovara operaciji projekcije u relacionoj algebri
- Primer: Naći imena svih ekspozitura u relaciji *loan*:

  select *branch_name*
  from *loan*
- U relacionoj algebri, odgovarajući upit je:

  $$\Pi_{branch\_name}(loan)$$
- Napomena: SQL imena su case insensitive

Komanda select (2)

- SQL dozvoljava duplikate u relacijama kao i u rezultatu upita.
- Za eliminaciju duplikata, koristi se ključna reč **distinct** posle select.
- Naći imena svih ekspozitura u relaciji *loan*, i eliminisati duplikate

```
select distinct branch_name
from loan
```

- Ključna reč **all** specificira da duplikate ne treba uklanjati.

```
select all branch_name
from loan
```

Komanda select (3)

- Zvezdica * u komandi select označava "sve atribute"

  select *
  from *loan*

- Komanda **select** može sadržati aritmetičke izraze koji uključuju operacije, +, –, *, /, na konstantama ili atributima torki.
- Upit:

  select *loan_number, branch_name, amount* * 100
  from *loan*

  daje relaciju koja se od relacije *loan* razlikuje po vrednosti atributa *amount* koji je pomnožen sa 100.

Komanda where (1)

- Komanda **where** specificira uslove koje rezultat mora zadovoljiti
  - Odgovara predikatu selekcije u relacionoj algebri.
- Naći sve brojeve kredita za kredite podignute u ekspozituri Perryridge sa iznosima većim od $1200.

```
select loan_number
from loan
where branch_name = 'Perryridge´and
                    amount > 1200
```

- Mogu se koristiti logičke operacije **and**, **or**, i **not**.
- Mogu se koristiti aritmetički izrazi.

Komanda where (2)

- SQL dozvoljava korišćenje operatora **between**

- Primer: Naći brojeve kredita sa iznosima između \$90,000 i \$100,000 (to jest, $\geq$ \$90,000 i $\leq$ \$100,000)

```
select loan_number
from loan
where amount between 90000 and 100000
```

Komanda from

- Komandom **from** se zadaju relacije na koje se upit odnosi
  - Odgovara operaciji Dekartovog proizvoda u relacionoj algebri.
- Naći Dekartov proizvod relacija *borrower* i *loan*

select *
from *borrower, loan*

- Naći ime, broj i iznos kredita za sve klijente koji imaju kredit u ekspozituri Perryridge.

select *customer_name, borrower.loan_number, amount*
from *borrower, loan*
where *borrower.loan_number = loan.loan_number* **and**
branch_name = 'Perryridge'

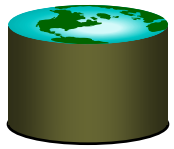
Operacija preimenovanja

- SQL omogućuje preimenovanje relacija i atributa korišćenjem komande **as**:

  staro-ime **as** *novo-ime*

- Naći ime, broj i iznos kredita za sve klijente; preimenovati atribut *loan_number* u *loan_id.*

```
select customer_name, borrower.loan_number as loan_id,
       amount
from borrower, loan
where borrower.loan_number = loan.loan_number
```



Promenljive Torki

- Promenljive torki su definisane korišćenjem komande **as** u komandi **from**.
- Naći imena klijenata i brojeve njihovih kredita koje imaju u nekoj ekspozituri.

```
select customer_name, T.loan_number, S.amount
from borrower as T, loan as S
where  T.loan_number = S.loan_number
```

- Naći imena svih ekspozitura koje imaju veći prihod nego neka ekspozitura u Brooklyn-u.

```
select distinct T.branch_name
from branch as T, branch as S
where T.assets > S.assets and S.branch_city = 'Brooklyn'
```

Sortiranje Prikazivanja Torki

- Prikazati abecednim redom imena svih klijenata koji imaju kredit u ekspozituri Perryridge

```
select distinct customer_name
from    borrower, loan
where borrower.loan_number=loan.loan_number
            and branch_name = 'Perryridge'
order by customer_name
```

- Može se specificirati **desc** za opadajući ili **asc** za rastući redosled, za svaki atribut; rastući redosled je default.
  - Primer: **order by** *customer_name* **desc**

Skupovne Operacije

- Skupovne operacije **union**, **intersect**, i **except** odgovaraju operacijama relacione algebre $\cup$, $\cap$, $-$.
- Svaka od ovih operacija automatski eliminiše duplikate; ako hoćemo da zadržimo duplikate koristimo odgovarajuće multiset verzije **union all**, **intersect all** i **except all**.

  Pretpostavimo da se torka pojavljuje *m* puta u *r* i *n* puta u *s*, tada se pojavljuje:
  - *m* + *n* puta u *r* **union all** *s*
  - min(*m*,*n*) puta u *r* **intersect all** *s*
  - max(0, *m* – *n*) puta u *r* **except all** *s*

Skupovne Operacije (Primeri)

- Find all customers who have a loan, an account, or both:

  (select *customer_name* **from** *depositor*)
  union
  (select *customer_name* **from** *borrower)*

- Find all customers who have both a loan and an account.

  (**select** *customer_name* **from** *depositor*)
  intersect
  (select *customer_name* **from** *borrower)*

- Find all customers who have an account but no loan.

  (**select** *customer_name* **from** *depositor*)
  except
  (select *customer_name* **from** *borrower)*

Agregatne Funkcije

- Agregatne funkcije se izvršavaju na skupu vrednosti atributa relacije, i daju sledeće rezultate:

avg:	srednju vrednost
min:	minimalnu vrednost
max:	maximalnu vrednost
sum:	sumu vrednosti
count:	broj vrednosti

Agregatne Funkcije (Primeri)

- Find the average account balance at the Perryridge branch.

```
select avg (balance)
        from account
        where branch_name = 'Perryridge'
```

- Find the number of tuples in the *customer* relation.

```
select count (*)
        from customer
```

- Find the number of depositors in the bank.

```
select count (distinct customer_name)
        from depositor
```

Agregatne Funkcije – Group By

- Naći broj ulagača za svaku ekspozituru.

select *branch_name,* **count (distinct** *customer_name)*
from *depositor, account*
where *depositor.account_number =account.account_number*
group by *branch_name*

Atributi u komandi **select** izvan agregatne funkcije moraju se nalaziti u listi **group by**

Agregatne Funkcije – Having

- Naći imena svih ekspozitura u kojima je srednje stanje na računima veće od $1,200.

select *branch_name,* **avg** (*balance*)
from *account*
group by *branch_name*
having avg (*balance*) > 1200

Predikati u komandi **having** se primenjuju posle formiranja grupa dok se predikati u komandi **where** primenjuju pre formiranja grupa

Null Vrednosti (1)

- Torke mogu imati ***null*** vrednosti, za neke od atributa
- ***null*** označava nepoznatu vrednost ili da ta vrednost ne postoji.
- Predikat **is null** se koristi za proveru null vrednosti.
  - Primer: Naći sve brojeve kredita sa null vrednostima za iznos kredita.

    select *loan_number*
    from *loan*
    where *amount* **is null**
- Rezultat bilo kojeg aritmetičkog izraza sa *null* je *null*
  - Primer: 5 + *null* daje null
- Agregatne funkcije ignorišu null vrednosti

Null Vrednosti (2)

- Poređenje sa *null* daje *unknown*
  - Primer*: 5 < null ili null <> null ili null = null*
- Tro-vrednosna logika koristi istinosnu vrednost *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: *(true* **and** *unknown) = unknown,*
    (false **and** *unknown) = false,*
    (unknown **and** *unknown) = unknown*
  - NOT*:* *(***not** *unknown) = unknown*
  - "*P* **is unknown"** je true ako je predikat *P unknown*
- Rezultat predikata komande **where** se tretira kao *false* ako je *unknown*

Null Vrednosti (3)

- Naći ukupnu sumu svih iznosa kredita

  select sum (*amount*)
  from *loan*

  – Izraz ignoriše null iznose kredita
  – Rezultat je *null* ako nema ne-null iznosa

- Sve agregatne operacije izuzev **count(*)** ignorišu torke sa null vrednostima na agregiranim atributima.

Ugnježđeni Upiti

- SQL omogućuje ugnježđavanje upita.
- Ugnježđeni upit je **select-from-where** izraz zadat unutar drugog izraza.
- Ugnježđeni upiti se obično koriste za testiranje elemenata skupa, poređenje, i kardinalnost skupa.

Ugnježdeni Upiti (Primer)

- Find all customers who have both an account and a loan at the bank.

select distinct *customer_name*
from *borrower*
where *customer_name* **in** (**select** *customer_name*
from *depositor*)

- Find all customers who have a loan at the bank but do not have an account at the bank

select distinct *customer_name*
from *borrower*
where *customer_name* **not in** (**select** *customer_name*
from *depositor*)

Provera Postojanja Dupliciranih Torki

- Konstrukcijom **unique** se vrši provera postojanja dupliciranih torki u rezultatu.
- Naći sve klijente koji imaju ne više od jednog računa u ekspozituri Perryridge.

```
select T.customer_name
from depositor as T
where unique (
    select R.customer_name
    from account, depositor as R
    where T.customer_name = R.customer_name and
          R.account_number = account.account_number and
          account.branch_name = 'Perryridge' )
```

Primer

- Naći sve klijente koji imaju najmanje dva računa u ekspozituri Perryridge.

```
select T.customer_name
from depositor as T
where not unique (
    select R.customer_name
    from account, depositor as R
    where T.customer_name = R.customer_name and
          R.account_number = account.account_number and
          account.branch_name = 'Perryridge' )
```

Ugnježdeni Upiti u from

- SQL dozvoljava ugnježdavanje i unutar komande **from**
- Find the average account balance of those branches where the average account balance is greater than $1200.

```
select branch_name, avg_balance
from (select branch_name, avg (balance)
        from account
        group by branch_name )
        as branch_avg ( branch_name, avg_balance)
where avg_balance > 1200
```

Ne koristimo komandu **having**, već sračunavamo privremenu relaciju (pogled) *branch_avg* u komandi **from**, tako da atribut *branch_avg* možemo koristiti direktno u komandi **where.**

Pogledi (1)

- Potrebni su nam podskupovi konceptualnog modela.
- Npr. Hoćemo da omogućimo uvid u broj kredita klijenta, ali ne i u iznos:

  (**select** *customer_name, loan_number*
  from *borrower, loan*
  where *borrower.loan_number = loan.loan_number*)
- **Pogled (view)** obezbeđuje mehanizam za pravljenje podskupova konceptualnog modela.
- Neka relacija koja nije deo konceptualnog modela ali je korisnik može videti kao "virtuelnu relaciju" naziva se **pogled** (**view**).

Pogledi (2)

- Pogled se definiše korišćenjem naredbe **create view** koja ima sledeću formu

  create view *v* **as** < query expression >

  gde je *v* ime pogleda, a <query expression> neki legalni SQL izraz.

- Pošto je pogled definisan, ime pogleda se može koristiti za referisanje virtuelne relacije koju pogled generiše.

- Definisanje pogleda nije isto što i kreiranje nove relacije kao rezultata nekog izraza u upitu
  - Pogled nam omogućuje eliminisanje izraza u upitu.

Pogledi (Primer)

- Pogled koji sadrži ekspoziture i njihove klijente

```
create view all_customer as
        (select branch_name, customer_name
         from depositor, account
        where depositor.account_number =
                account.account_number )
        union
        (select branch_name, customer_name
        from borrower, loan
        where borrower.loan_number = loan.loan_number )
```

- Naći sve klijente ekspoziture Perryridge

```
select customer_name
        from all_customer
        where branch_name = 'Perryridge'
```

Definisanje Pogleda Drugim Pogledima

- Za definisanje pogleda se mogu koristiti drugi pogledi
- Za pogled v_1 se kaže da *direktno zavisi* (*depend directly)* od pogleda v_2 ako se v_2 koristi u definiciji pogleda v_1
- Za pogled v_1 se kaže da *zavisi* od pogleda v_2 ako *direktno zavisi* od v_2 ili postoji putanja zavisnosti od v_1 ka v_2
- Za pogled v se kaže da je *rekurzivan* ako zavisi sam od sebe.

Ekspanzija Pogleda

- Način određivanja značenja pogleda definisanog posredstvom drugih pogleda.
- Neka je pogled v_1 definisan izrazom e_1 koji koristi poglede.
- Ekspanzija pogleda se vrši po sledećem algoritmu:

  repeat
  Nađi neki pogled v_i u e_1
  Zameni pogled v_i sa definicionim izrazom za v_i
  until nema više pogleda u e_1

- Ako definicije pogleda nisu rekurzivne, petlja nije beskonačna

Modifikacije Baze - Brisanje

- Obrisati sve račune u ekspozituri Perryridge

  delete from *account*
  where *branch_name* = 'Perryridge'

- Obrisati sve račune u svakoj ekspozituri u gradu 'Needham'.

  delete from *account*
  where *branch_name* **in** (**select** *branch_name*
  from *branch*
  where *branch_city* = 'Needham')

Brisanje - Problem

- Obrisati sve račune čija su stanja ispod srednjeg stanja računa u banci.

delete from *account*
where *balance* < (**select avg** (*balance*)
from *account*)

  - Problem: ako brišemo torke iz računa, srednje stanje se menja
  - Rešenje koje se koristi u SQL:
    1. Prvo, sračunati **avg** stanje i naći sve torke za brisanje
    2. Zatim, obrisati sve nađene torke (bez ponovnog sračunavanja **avg** i pretraživanja torki)

Modifikacije Baze - Dodavanje Torki(1)

- Dodavanje nove torke u relaciju *account*

  insert into *account*
  values (‘A-9732’, ‘Perryridge’,1200)

  ili ekvivalentno

  insert into *account* (*branch_name, balance, account_number*)
  values (‘Perryridge’, 1200, ‘A-9732’)

- Dodavanje nove torke u relaciju *account* sa null *stanjem*

  insert into *account*
  values (‘A-777’,‘Perryridge’, *null*)

Modifikacije Baze - Dodavanje Torki(2)

- Poklon od $200 na štedni račun za sve klijente sa kreditom u ekspozituri Perryridge, pri čemu se broj kredita koristi kao broj novog štednog računa.

```
insert into account
   select loan_number, branch_name,  200
   from loan
   where branch_name = 'Perryridge'
insert into depositor
   select customer_name, loan_number
   from loan, borrower
   where branch_name =  'Perryridge'
         and loan.account_number = borrower.account_number
```

- Struktura **select from where** se realizuje pre nego što se izvrši dodavanje torke u relaciju; u suprotnom bi upiti kao što je sledeći izazivali probleme:

```
insert into table1 select * from table1
```

Modifikacije Baze - Ažuriranje Torki

- Dodati kamatu od 6% za stanja na računima preko $10,000, a 5% za ostale.
  - Ažuriranje se vrši sa komandom **update**:

update *account*
set *balance* = *balance* $*$ 1.06
where *balance* > 10000

update *account*
set *balance* = *balance* $*$ 1.05
where *balance* $\leq$ 10000

  - Redosled je važan
  - Bolje je koristiti **case** strukturu

Ažuriranje Torki - Case

- Isti upit: Dodati kamatu od 6% za stanja na računima preko $10,000, a 5% za ostale.

```
update account
 set balance =  case
                  when balance <= 10000 then
                                          balance *1.05
                  else   balance * 1.06
                end
```

Ažuriranje Posredstvom Pogleda (1)

- Kreiranje pogleda sa svim podacima o kreditu izuzev iznosa kredita:

  create view *branch_loan* **as**
  select *branch_name, loan_number*
  from *loan*

- Dodavanje nove torke u pogled *branch_loan*

  insert into *branch_loan*
  values (‘Perryridge’, ‘L-307’)

  Ovo dodavanje mora biti realizovano dodavanjem torke

  (‘L-307’, ‘Perryridge’, *null*)

  u relaciju *loan* na kojoj je pogled definisan.

Ažuriranje Posredstvom Pogleda (2)

- Neka ažuriranja posredstvom pogleda se ne mogu realizovati na relacijama na kojima je pogled definisan
  - **create view** *v* **as**
    select *branch_name* **from** *account*

    insert into *v* **values (**'*L-99*', 'Downtown', '*23*'**)**

- Neka nisu jedinstvena
  - **insert into** *all_customer* **values** *(*'Perryridge', 'John'**)**
    - Može za loan ili account, i treba kreirati novi loan/account broj!
- Većina SQL implementacija dozvoljava ažuriranje samo na pogledima bez agregacija definisanim na jednoj relaciji

Spojene Relacije** (Joined Relations**)

- **Operacija Join** spaja dve relacije i vraća kao rezultat novu relaciju.
- Tipično se koristi u komandi **from**
- **Join uslov** – definiše koje torke u dve relacije se poklapaju, i koji atributi se pojavljuju u rezultatu spajanja.
- **Join tip** – definiše kako se tretiraju torke koje se ne poklapaju (na osnovu join uslova).

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

Spojene Relacije - Primeri (1)

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

- *loan* **inner join** *borrower* **on** *loan.loan_number = borrower.loan_number*

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

- *loan* **left outer join** *borrower* **on** *loan.loan_number = borrower.loan_number*

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	*null*	*null*

Spojene Relacije - Primeri (2)

- *loan* **natural inner join** *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- *loan* **natural right outer join** *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Spojene Relacije - Primeri (3)

- *loan* **full outer join** *borrower* **using** (*loan_number*)

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

- Find all customers who have either an account or a loan (but not both) at the bank.

select *customer_name*
from (*depositor* **natural full outer join** *borrower*)
where *account_number* **is null or** *loan_number* **is null**