

AKTIVNE BAZE PODATAKA

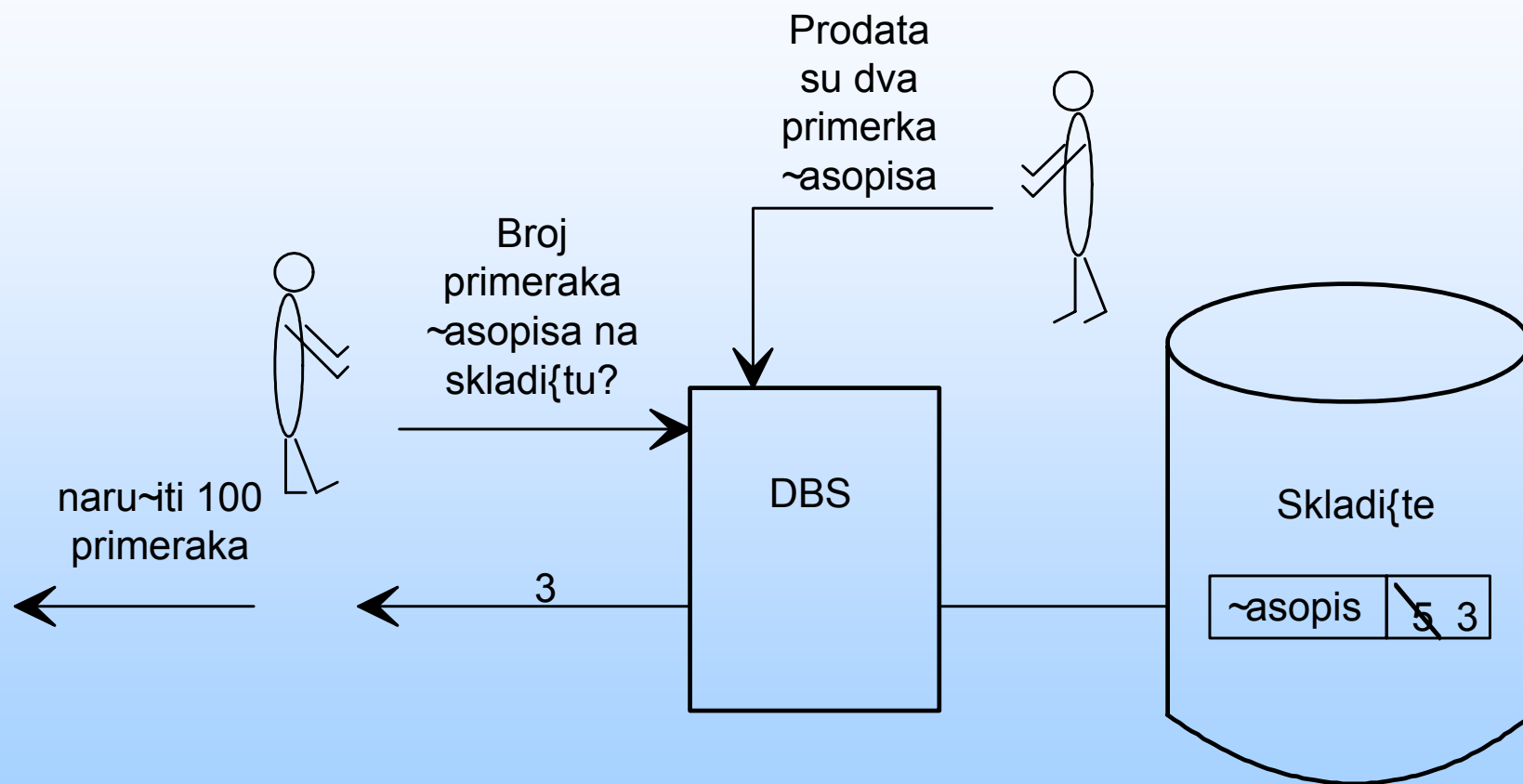


Aktivne baze podataka

Aktivni sistemi baza podataka proširuju funkcionalnost tradicionalnih baza podataka uvođenjem sistema pravila koji se koristi kao opšti, unificirani mehanizam za:

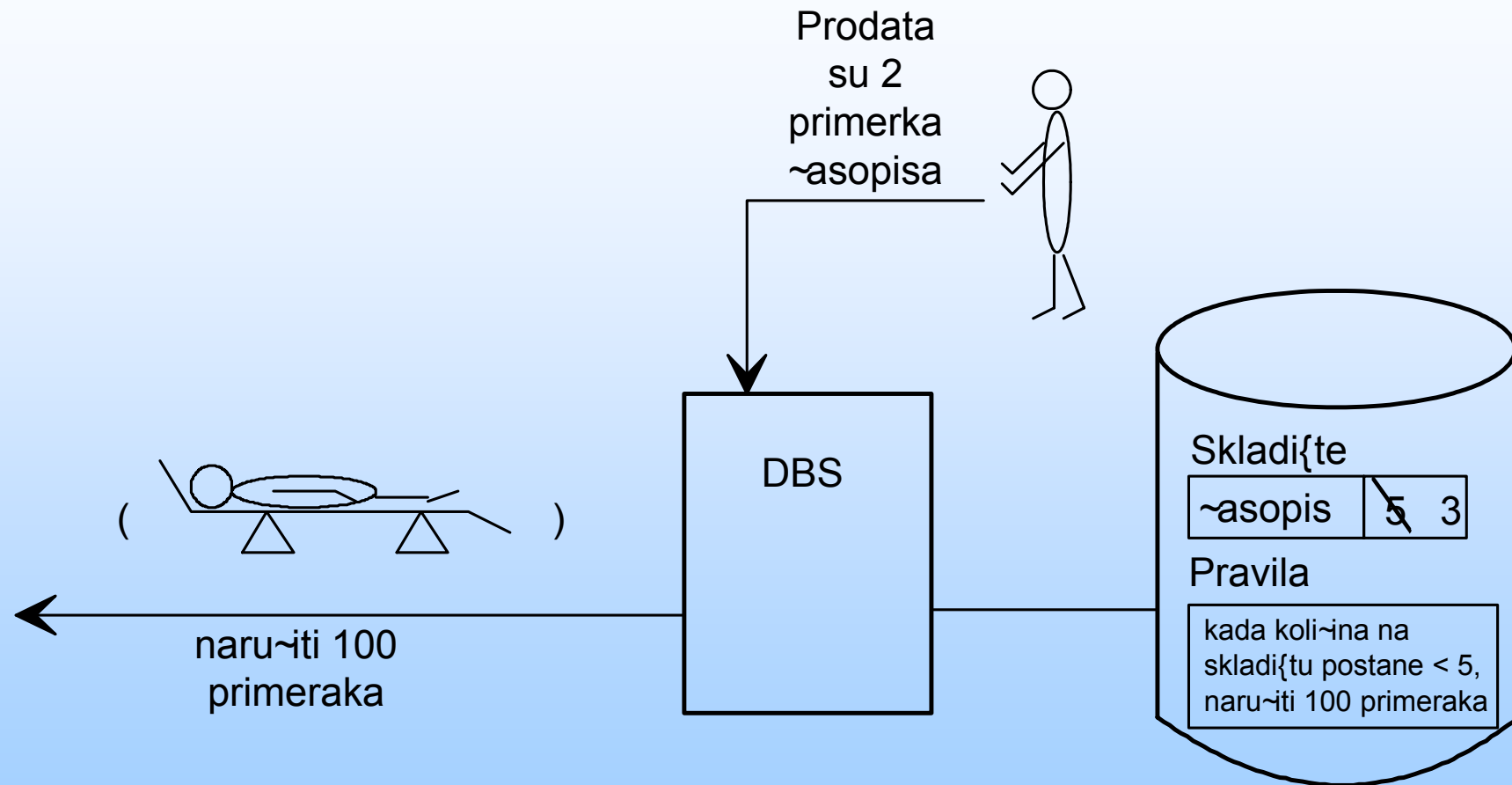
- kontrolu integriteta baze podataka,
- kontrolu prava pristupa,
- održavanje pogleda,
- generisanje izvedenih podataka,
- prikupljanje statističkih podataka,
- praćenje funkcionisanja baze i vođenje žurnala,
- ...

Konvencionalni SUBP



Pasivni sistem baze podataka

Aktivni SUBP



Aktivni sistem baze podataka



Aktivni SUBP

U osnovi aktivnih sistema nalazi se opšta ideja:

- prepoznati prethodno opisane **situacije** (događaje i uslove) u bazi podataka, aplikacijama i okruženju, i
- pokrenuti definisane **reakcije** (operacije nad bazom podataka ili programe) po nastanku situacija.

Pored logičke šeme i činjenica, aktivni sistemi omogućuju obuhvatanje i dodatne dinamike sistema iskazane kroz složena pravila integriteta.

Specifikacija pravila

Svako pravilo definiše situaciju i akcije koje se preduzimaju sa nastankom situacije. Pravila u aktivnim bazama podataka su poznata pod imenom **ECA** (**E**vent-**C**ondition-**A**ction) ***produkciona pravila*** i specifikuju se na sledeći način:

ON dogadjaj

IF uslov

"situacija"

DO akcija

"(re)akcija"



Model znanja: semantika ECA-pravila

Da bi se pravilo moglo formirati prethodno moraju biti definisani događaji. Generalno govoreći, moguće je razlikovati **primitivne** i **složene** događaje. Primitivni događaji su:

- **Ažuriranje podataka**
- **Prikaz podataka**
- **Vreme**
- **Aplikativno definisan događaj**

Akcije se pokreću neposredno nakon (AFTER) ili neposredno pre (BEFORE) detektovanja događaja.



Model znanja: semantika ECA-pravila

Složeni događaji formiraju se kombinovanjem primitivnih i prethodno definisanih složenih događaja. Korisni operatori za kombinovanje događaja mogu biti:

- **Logički operatori.** Događaji mogu biti kombinovani korišćenjem logičkih operatora AND, OR, NOT, itd.
- **Sekvenca.** Pravilo može biti pokrenuto kada se dva ili više događaja pojave u određenom, definisanom redosledu.
- **Vremenska kompozicija.** Pravilo može biti pokrenuto kombinovanjem vremenskih i događaja koji nisu vremenski, na primer "*5 sekundi nakon događaja D1*" ili "*svaki sat nakon prvog pojavljivanja događaja D2*".

Neki aktivni sistemi dozvoljavaju parametrizaciju događaja.



Model znanja: semantika ECA-pravila

Uslov može biti predikat nad stanjem baze podataka, upit nad bazom podataka ili izvršavanje određene aplikativne procedure. U poslednja dva slučaja uslov je zadovoljen ako upit, odnosno aplikativna procedura vraća neke podatke.

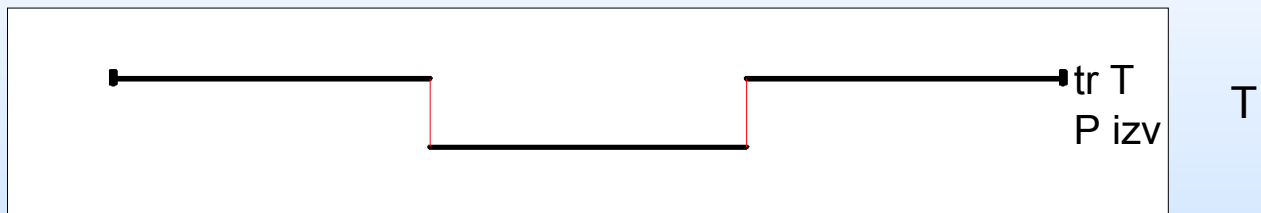
Akcija je bilo koji program. Takav program može da uključuje operacije nad bazom podataka i implicitno ili eksplicitno generiše nove događaje.

Model znanja treba da uključi i implicitne događaje (pravilo se izvršava uvek (ALWAYS) ili prvi put (FIRST) kada uslov bude zadovoljen), (selektivnu) aktivaciju/deaktivaciju pravila, preciziranje konteksta pravila kroz grupisanje, odnosno bolju organizaciju velikih baza pravila itd.

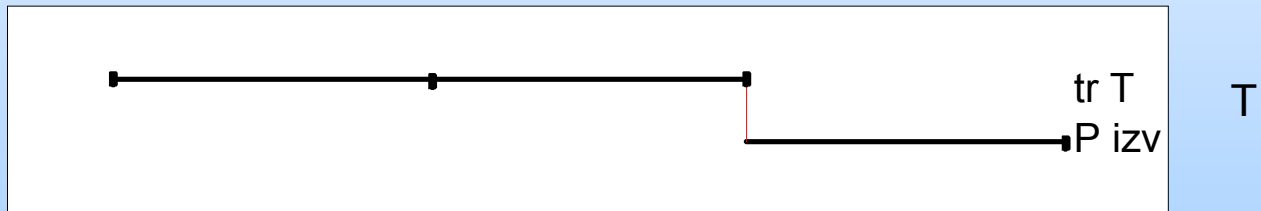
Model izvršavanja: obrada ECA-pravila

Postoji nekoliko različitih načina povezivanja transakcije, odnosno događaja koji pokreće pravilo i izvršavanja ECA-pravila.

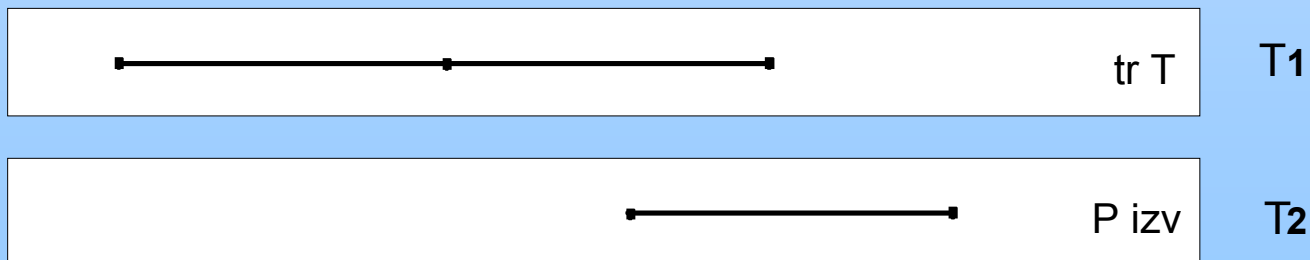
a) trenutni način



b) odložen način



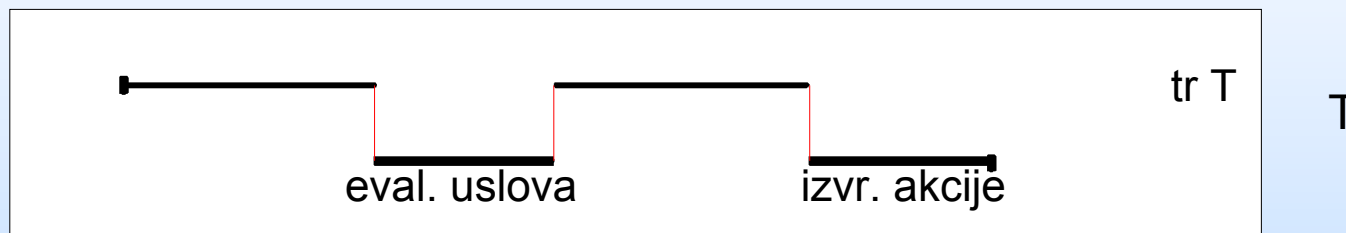
c) razdvojen način



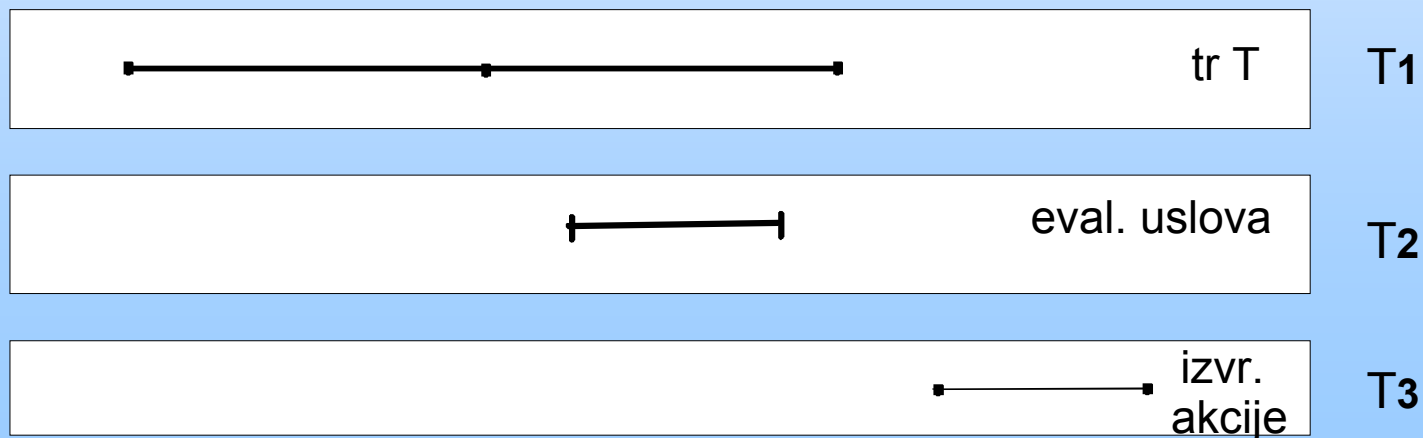
Model izvršavanja: obrada ECA-pravila

Moguće je koristiti i druge, složenije modele izvršavanja ECA-pravila

a) trenutn/odložen način



b) razdvojen/razdvojen način





Model izvršavanja: obrada ECA-pravila

Pored izbora i specifikacije odgovarajućeg načina povezivanja obrade transakcije i pravila, prilikom definisanja modela izvršavanja pravila potrebno je voditi računa i o izboru:

- redosleda izvršavanja pravila (sekvencijalni ili konkurentni) u slučaju da je za isti događaj definisano više pravila;
- najboljeg načina izvršavanja pravila čije akcije pokreću nova pravila (moguće rekurzivno pokretanje samog sebe);
- mehanizma oporavka.



Pregled pristupa razvoju aktivnih baza

Aktivni sistemi se, uslovno rečeno, razvijaju ili kao nadgradnja objektno orijentisanih sistema ili kao nadgradnja relacionih sistema. Najpoznatiji objektno orijentisan sistem za upravljanje bazom podataka, koji podržava ECA pravila, je **HiPAC** (CCA/XAIT, U. Wisconsin).

Najpoznatiji predstavnici aktivnih sistema koji su se razvijali kao nadgradnja relacionih sistema su **Postgres** (UC Berkley) i **Starburst** (IBM Almaden).



HiPAC (High Performance Active Database System)

HiPAC je najpoznatiji aktivni sistem, koji je zasnovan na objektno orijentisanom modelu podataka.

HiPAC poseduje veoma generalizovan jezik pravila. Pravila i komponente pravila su objekti baze podataka, što znači da:

- mogu biti kreirana, prikazana, izmenjena ili izbrisana slično drugim objektima;
- mogu biti grupisana i organizovana slično drugim objektima;
- mogu pripadati potklasama, imati attribute i biti povezana sa drugim objektima.



HiPAC (High Performance Active Database System)

Jedna od klasa podataka u HiPAC-u je klasa ***Rule*** koja je potklasa klase ***Entity***. Klasa *Rule* ima sledeće attribute i operacije:

Atributi	Operacije	
<i>Event</i>	<i>Create</i>	<i>Enable</i>
<i>Condition</i>	<i>Delete</i>	<i>Disable</i>
<i>Action</i>	<i>Modify</i>	<i>Fire</i>
<i>ostali atributi</i>		



Starburst

Starburst je prototipski sistem koji se, od 1985. godine, razvijao u IBM-ovom istraživačkom centru u San Jose-u u Kaliforniji. Jedan od najznačajnijih ciljeva Starburst projekta bio je izrada aktivnog relacionog sistema čija je osnovna karakteristika **moгуćnost proširenja**, odnosno uključivanja nove tehnologije za svaku komponentu arhitekture sistema i davanje efikasne podrške za aplikacije bilo kog tipa.

Sintaksa pravila njegovog sistema pravila je zasnovana na SQL-u. Kako su relacioni upitni jezici skupovno orijentisani (operand je cela tabela, odnosno grupa n-torki, a ne pojedinačna n-torka tabele), i pravila su skupovno orijentisana - pokreću ih skupovi promena nad bazom podataka, a akcije pravila mogu takođe da proizvedu skupove promena nad bazom.



Starburst

Pravila su zasnovana na pojmu ***prelaza stanja (state transition)***. Prelaz stanja je promena stanja baze podataka nastala kao rezultat izvršavanja nedeljive sekvence operacija za ažuriranje baze (insert, update, delete). U razmatranje se uzimaju ***neto efekti*** prelaza stanja:

- ukoliko je n-torka ažurirana (menjana) više puta, neto efekat je samo jedno ažuriranje n-torke sa kombinovanim vrednostima pojedinačnih izmena;
- ukoliko je n-torka izmenjena, pa nakon toga izbačena, razmatra se samo izbacivanje;
- ukoliko je n-torka ubačena, pa izmenjena, neto efekat je ubacivanje n-torke sa izmenjenim vrednostima;
- ukoliko je n-torka ubačena, pa nakon toga izbačena, smatra se da nije ni došlo do promene stanja baze podataka.



Starburst

Sintaksa naredbe za kreiranje pravila je sledećeg izgleda:

```
create rule <naziv_pravila> on <tabela>  
when <predikat_prelaza>  
[if <uslov>]  
then <lista_akcija>  
[precedes <lista_pravila>]  
[follows <lista_pravila>]
```

Uslov i akcija pravila, posredstvom SQL operacija, mogu da referenciraju tekuće stanje baze podataka. Pored toga, i uslov i akcija mogu da referenciraju ***tabele prelaza*** - logičke tabele koje odražavaju promene nastale u toku prelaza stanja. Postoje četiri tabele prelaza: **inserted**, **deleted**, **new-updated** i **old-updated**. Pravilo može da referencira bilo koju tabelu prelaza koja odgovara nekoj od operacija koje ga pokreću.



Starburst - primer pravila

Jezik pravila ilustrovaćemo pravilom koje kontroliše srednja primanja radnika. Pokreće se uvek kada se ubacuju podaci o novim radnicima ili menjaju plate postojećih. Ukoliko srednja primanja radnika pređu iznos od 100000, transakcija se poništava:

```
create rule kontrola-plata on radnik
when inserted, updated(plata)
if (select avg(plata) from radnik) > 100000
then (select * from inserted) union
      (select * from new-updated);
      rollback
```



Postgres

Postgres je prototipski sistem koji se razvijao na Kalifornijskom univerzitetu, Berkli, kao naslednik INGRES projekta. Postgres je najpre imao PRS I sistem pravila koji je zatim zamenjen PRS II sistemom. Sintaksa pravila oba sistema zasnovana je na Postquel-u.

PRS I sistem pravila se zasnivao na ideji da se bilo koja Postquel naredba može transformisati u pravilo promenom semantike naredbe, tako da se naredba logički izvršava ili ***uvek*** (ALWAYS) ili ***nikad*** (REFUSE) ili ***tačno jedanput*** (ONE-TIME).



Postgres

Sintaksa PRS I pravila je sledećeg izgleda:

{ALWAYS | REFUSE | ONE-TIME} Postquel-naredba

Događaji u PRS I sistemu pravila su ***implicitni***. Za razliku od Starburst pravila, koja su skupovno orijentisana, PRS I pravila su rekord (tuple) orijentisana; pokreće ih promena nad pojedinačnom n-torkom relacije. PRS I pravila nemaju mogućnost referenciranja "starih" i "novih" vrednosti.



Postgres

PRS II sistem pravila je nastao kao rezultat sagledanih nedostataka PRS I sistema i predstavlja povratak na konvencionalni način predstavljanja znanja u vidu produkcionih pravila. Sintaksa PRS II pravila je sledećeg izgleda:

```
define [tuple | rewrite] rule <naziv>  
on <dogadjaj> to <objekat>  
[where <uslov>]  
do [instead] <lista akcija>
```



Postgres - primer pravila

Jezik pravila ilustrovaćemo pravilom koje obezbeđuje da Pavle uvek ima istu platu kao Milan:

```
define rule MilanPavle  
on new to radnik.plata  
where radnik.ime = "Milan"  
    do replace radnik (plata = new.plata)  
    where radnik.ime = "Pavle"
```



SQL:1999 Trigeri

Trigeri su specifična vrsta ECA (Event-Condition_Action) pravila.

Događaj je operacija ažuriranja baze podataka, uslov je proizvoljni SQL predikat, a akcija je sekvenca SQL naredbi.

Sekvenca SQL naredbi se izvršava ako je detektovan događaj (pokrenuta odgovarajuća operacija ažuriranja baze) i ako je uslov zadovoljen (sračunava se u true).



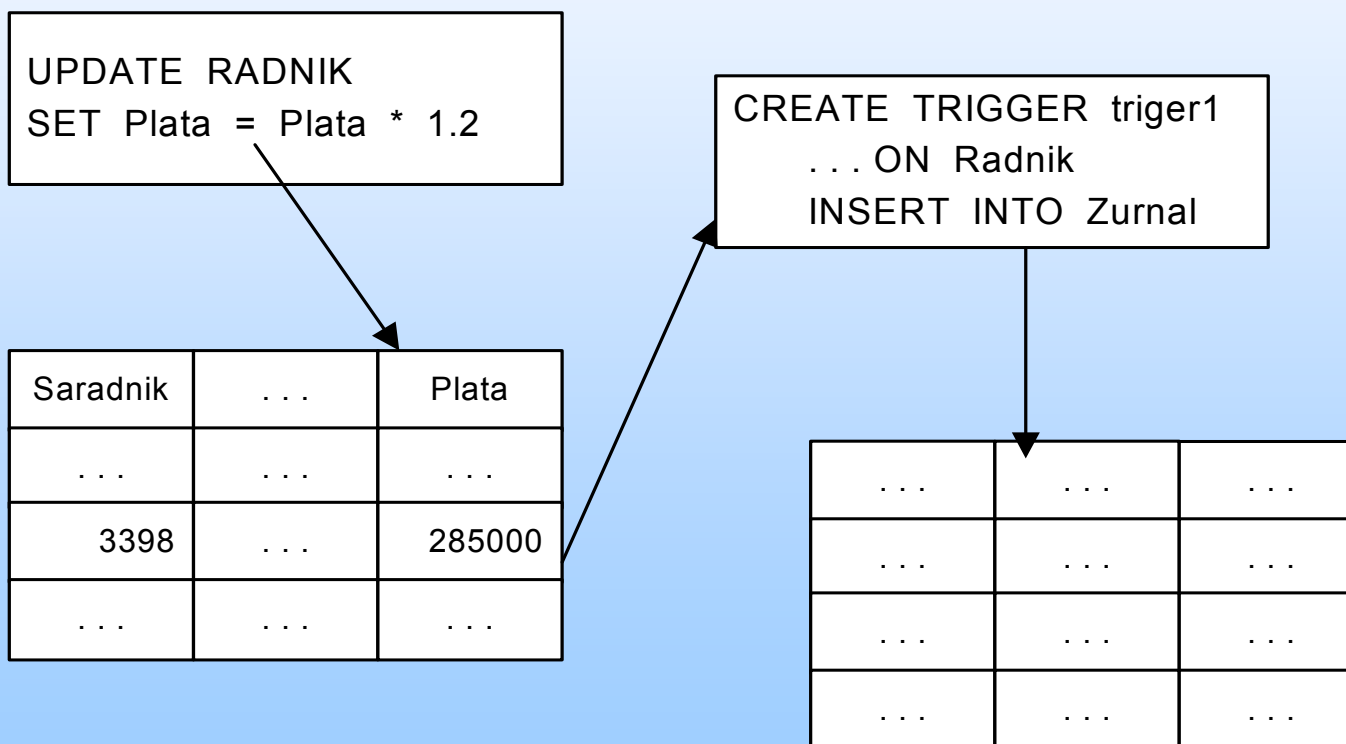
Osnovne karakteristike trigera

SQL:1999 standard definiše trigere kao objekte šeme baze podataka, koji su vezani za tačno jednu baznu tabelu i koji se pozivaju svaki put:

- kada se ubaci jedan ili više redova u tabelu za koju je definisan triger (INSERT naredba),
- kada se promeni jedan ili više redova u tabeli za koju je definisan triger (UPDATE naredba), ili
- kada se obriše jedan ili više redova iz tabele za koju je definisan triger (DELETE naredba).

Osnovne karakteristike triger

Primer korišćenja triger za vođenje žurnala o izvršenim operacijama nad bazom podataka:





Klasifikacija trigeri

SQL:1999 trigeri mogu biti kategorizovani na više načina, u skladu sa različitim aspektima posmatranja:

- Trigeri mogu da se pozivaju neposredno pre (BEFORE) ili neposredno posle (AFTER) izvršavanja SQL naredbe (operacije ažuriranja) za koju je definisan triger. U skladu sa time mogu se razvrstati kao:
 - BEFORE trigeri
 - AFTER trigeri
- U zavisnosti od toga koji od tri tipa operacija ažuriranja baze podataka (INSERT, UPDATE, DELETE) predstavlja događaj koji izaziva pokretanje trigeri, trigeri mogu biti:
 - INSERT trigeri
 - UPDATE trigeri
 - DELETE trigeri



Klasifikacija triger

Treba napomenuti da SQL:1999 standard ne dozvoljava definisanje triger za SELECT naredbu, odnosno za prikaz podataka.

Triger se može izvršiti jednom za operaciju ažuriranja koja ga pokreće ili jednom za svaki red koji je ažuriran (ubačen, izmenjen ili obrisani) operacijom koja pokreće triger. U skladu sa time razlikuju se:

- trigeri koji se pokreću na nivou naredbe (statement-level trigeri)
- trigeri koji se pokreću na nivou reda (row-level trigeri)



Klasifikacija triger

U skladu sa opisanom klasifikacijom, moguće je, na primer, posmatrati triger opisati kao

"a before insert statement-level trigger" (triger koji se pokreće samo jednom neposredno pre izvršavanja insert naredbe nad tabelom za koju je definisan)

ili kao

"an after update row-level trigger" (triger koji se pokreće neposredno posle update naredbe nad tabelom za koju je definisan i to jedanput za svaki red koji se modifikuje tom naredbom).

Specifikacija trigeru u SQL:1999 standardu

Za specifikaciju trigeru u SQL:1999 standardu koristi se sledeća sintaksa:

- 1) CREATE TRIGGER <naziv trigeru>
- 2) { BEFORE | AFTER }
- 3) { INSERT | DELETE | UPDATE [OF <naziv kolone₁>, <naziv kolone₂>, ...] }
- 4) ON <naziv tabele>
- 5) [REFERENCING { OLD [ROW] [AS] <sinonim za red pre ažuriranja>
- 6) | NEW [ROW] [AS] <sinonim za red posle ažuriranja>
- 7) | OLD TABLE [AS] <sinonim za tabelu sa starim redovima>
- 8) | NEW TABLE [AS] <sinonim za tabelu sa novim redovima> }, ...]
- 9) [FOR EACH { ROW | STATEMENT }]
- 10) [WHEN (<uslov>)]
- 11) { <SQL naredba>
- 12) | BEGIN ATOMIC
- 13) { <SQL naredba>; } ...
- 14) END }



Specifikacija trigeru u SQL:1999 standardu

Sledi primer trigeru kojim se ograničava povišica plate na najviše 20%:

```
CREATE TRIGGER KontrolaPovecanjaPlate
AFTER UPDATE OF Plata ON Radnik
REFERENCING
    OLD ROW AS StariRed
    NEW ROW AS NoviRed
FOR EACH ROW
WHEN (NoviRed.Plata > StariRed.Plata * 1.2)
    UPDATE Radnik
    SET Plata = StariRed.Plata * 1.2
    WHERE Sradnik = NoviRed.Sradnik;
```

Korisnik koji kreira triger mora da ima TRIGGER privilegiju nad tabelom za koju kreira triger. Vlasnik tabele već ima TRIGGER privilegiju, a korisniku koji nije vlasnik tabele TRIGGER privilegija mora da bude dodeljena naredbom GRANT.



Realizacija SQL:1999 trigera u komercijalnim sistemima

- Mnogi proizvođači sistema za upravljanje bazom podataka su realizovali trigere pre objavljivanja SQL:1999 standarda, uprkos tome što SQL-92 standard nije uključio specifikaciju za trigere. Posledica toga je postojanje razlike u implementaciji trigera u različitim komercijalnim proizvodima. Druga bitna karakteristika je da su mehanizmi trigera u komercijalnim proizvodima dosta moćniji u odnosu na karakteristike propisane SQL:1999 standardom. Većina komercijalnih proizvoda podržava SELECT operaciju i naredbe za definisanje podataka kao događaje koji pokreću trigere, mogućnost definisanja trigera nad pogledom i INSTEAD OF klauzulu, itd.
- Posebno je značajna mogućnost specificiranja INSTEAD OF klauzule pri definisanju trigera, umesto već opisanih BEFORE i AFTER klauzula. Time se omogućuje da se akcija trigera izvrši umesto, a ne kao dodatak na izvršavanje događaja trigera.



Realizacija SQL:1999 trigera u komercijalnim sistemima

- U cilju ilustracije podsetimo se definicije pogleda Odeljenje30:

```
CREATE VIEW Odeljenje30 AS  
SELECT Sradnik, Ime, Posao  
FROM Radnik  
WHERE Odeljenje# = 30;
```

- Navedeni pogled nije ažurljiv jer ne uključuje atribut Odeljenje#, koji je obavezan (NOT NULL). Kako znamo da se kroz pogled Odeljenje30 ubacuju samo radnici odeljenja sa šifrom 30, možemo napisati sledeći triger, koji će navedeni pogled učiniti ažurljivim:

```
CREATE TRIGGER RadnikOdeljenja30  
INSTEAD OF INSERT ON Odeljenje30  
REFERENCING NEW ROW AS NoviRed  
FOR EACH ROW  
INSERT INTO Radnik (Sradnik, Ime, Posao, Odeljenje#)  
VALUES (NoviRed.Sradnik, NoviRed.Ime, NoviRed.Posao, 30);
```