



# Napredni modeli podataka – zašto?

# Što se promijenilo

- Nove programske paradigme neusklađene s relacijskim modelom (OO paradigma)
- Nova područja primjene (geoprostorne baze, streamovi,...)
- Ogromne količine podataka (društvene mreže, e-kupovina,...)
  - Facebook: 500+ terabytes of new data ingested into the databases every day
- Jeftin hardver, puno jeftinih računala
- U nekim primjenama ACID svojstva transakcije nisu nužna ili nisu nužna odmah
- Uporaba isključivo relacijskih baze podataka nije najbolje rješenje za sve primjene



IEC prefix		Representations				Customary prefix	
Name	Symbol	Base 2	Base 1024	Value	Base 10	Name	Symbol
kibi	Ki	2 <sup>10</sup>	1024 <sup>1</sup>	1 024	≈1.02 × 10 <sup>3</sup>	kilo	k, K
mebi	Mi	2 <sup>20</sup>	1024 <sup>2</sup>	1 048 576	≈1.05 × 10 <sup>6</sup>	mega	M
gibi	Gi	2 <sup>30</sup>	1024 <sup>3</sup>	1 073 741 824	≈1.07 × 10 <sup>9</sup>	giga	G
tebi	Ti	2 <sup>40</sup>	1024 <sup>4</sup>	1 099 511 627 776	≈1.10 × 10 <sup>12</sup>	tera	T
pebi	Pi	2 <sup>50</sup>	1024 <sup>5</sup>	1 125 899 906 842 624	≈1.13 × 10 <sup>15</sup>	peta	P
exbi	Ei	2 <sup>60</sup>	1024 <sup>6</sup>	1 152 921 504 606 846 976	≈1.15 × 10 <sup>18</sup>	exa	E
zebi	Zi	2 <sup>70</sup>	1024 <sup>7</sup>	1 180 591 620 717 411 303 424	≈1.18 × 10 <sup>21</sup>	zetta	Z
yobi	Yi	2 <sup>80</sup>	1024 <sup>8</sup>	1 208 925 819 614 629 174 706 176	≈1.21 × 10 <sup>24</sup>	yotta	Y

# Model podataka

Informally, a data model is a type of data abstraction that is used to provide conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model hides storage and implementation details that are not of interest to most database users.

*(R. Elmasri, S.B. Navathe: Fundamentals of Database Systems, Addison-Wesley, 2011, 6<sup>th</sup> edition)*

A data model is a collection of high-level data description constructs that hide many low-level storage details. A DBMS allows a user to define the data to be stored in terms of a data model.

*(R. Ramakrishnan, J.Gehrke: Database Management System, McGraw-Hill, 2007, 3rd ed)*

Data model: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical, and view levels.

*(A. Silberschatz, H.F. Korth, S. Sudarshan: Database Systems Concepts, 6<sup>th</sup> Edition, McGraw-Hill, 2010.)*

## Zašto su modeli (baza) podataka važni?

- Predstavljaju temelj na kojem se gradi programska potpora ili informacijski sustav
- Temelji moraju biti stabilni – ako nisu – sve što je nad njima izgrađeno (programska potpora za rad s bazom podataka) uružit će se
- ? Što znači stabilnost modela?
- ? Znači li to da se modeli ne mogu razvijati?
- Jezgra modela mora biti stabilna – nepromjenjiva
- Oko jezgre se dodaju novi elementi koji zahtijevaju nove primjene

# Modeli podataka $\leftrightarrow$ modeli baza podataka

- **Model podataka** je formalni sustav koji koristimo kod modeliranja baza podataka
- Model podataka može biti:
  - Relacijski,
  - Objektni,
  - Objektno-relacijski,
  - Polustrukturirani,
  - Ključ - vrijednost
  - Grafovski
  - itd...
- Model baze podataka je pojednostavnjena slika nekog segmenta stvarnog svijeta
- Model baze podataka temelji se na nekom (logičkom) modelu podataka (formalnom sustavu)

## Koji su modeli podataka dobri?

- ? Koji su modeli podataka (formalni sustavi) dobri?
- ? Postoje li dobri i loši modeli podataka?
- ? Postoje li najbolji modeli podataka?
- Ne postoje apsolutno dobri ili loši modeli podataka
- Neki modeli podataka primjenjiviji su za neke primjene
- Potrebno je voditi računa o primjenjivosti modela u nekom području i o njegovim ograničenjima → *CILJ KOLEGIJA*



# Modeli podataka - koncepti

## ■ Osnovni objekti

- Vrste objekata
- Složenost objekata
- Mogućnost definiranja vlastitih tipova podataka
- Nasljeđivanje
- Apstraktni tipovi podataka

## ■ Operacije

- Razina jezika za upravljanje podacima (3. generacija, 4. generacija, 5. generacija, ..)
- Proceduralnost
- Ekspresivnost
- Učahurivanje
- Ponovna iskoristivost

## ■ Integritetska ograničenja

- Mogućnost definiranja pravila za očuvanje integriteta na razini baze podataka (ne u primijenjenoj programskoj podršci/aplikacijama)
- Uobičajena ograničenja:
  - Entitetski integritet, integritet ključa, referencijski integritet, ograničenje domene, ograničenja odnosa među podacima
  - Poslovna pravila

# Napredni modeli i baze podataka

Predavanja  
Listopad 2015.

---

## **1. Pretraživanje teksta u relacijskim sustavima za upravljanje bazama podataka**



# Motivacijski primjer 1

Google Scholar search results for "databases new trends in 2015".

**Search Query:** databases new trends in 2015

**Results:** Oko 605.000 rezultata (0,07 s)

**Članci**

**Moja knjižnica**

**Bilo kad**

**Od 2015**

**Od 2014**

**Od 2011**

**Odabrani raspon...**

**Razvrstaj po važnosti**

**Razvrstaj po datumu**

☒ uključi patente

☒ uključi citate

☐ Stvori obavijest

**Savjet:** Pretražite rezultate samo na jeziku - hrvatski . Jezik pretraživanja možete postaviti ovdje: Postavke Znalca.

**Can we achieve Millennium Development Goal 4? New analysis of country trends and forecasts of under-5 mortality to 2015**

**CJL Murray, T Laakso, K Shibuya, K Hill, AD Lopez** - The Lancet, 2007 - Elsevier

... Here, we attempt to address some of these limitations by proposing **new** reproducible methods and, in doing so ... Child mortality measurements **database**. We compared **databases** of country measurements of child mortality maintained by UNICEF and WHO as of April 4, 2007. ...

Spominje se 238 puta Srodni članci Svih 20 inačica Web of Science: 120 Citiraj Spremi

**The COG database: new developments in phylogenetic classification of proteins from complete genomes**

**RL Tatusov, DA Natale, IV Garkavtsev** ... - Nucleic acids ..., 2001 - Oxford Univ Press

... frames (ORFs), originally annotated as genes, did not show detectable similarity to any proteins in current **databases**, but overlapped ... Zhang,J., Zhang,Z., Miller,W. and Lipman,DJ (1997) Gapped BLAST and PSI-BLAST: a **new** generation of protein **database** search programs. ...

Spominje se 1539 puta Srodni članci Svih 14 inačica Web of Science: 1042 Citiraj Spremi

**From genomics to chemical genomics: new developments in KEGG**

**M Kanehisa, S Goto, M Hattori** ... - Nucleic acids ..., 2006 - Oxford Univ Press

... Kanehisa, M., Goto, S., Kawashima, S., Nakaya, A. 2002The KEGG **databases** at GenomeNet ... B., Galperin, MY, Fedorova, ND, Koonin, EV 2001The COG **database**: **new** developments in ... Text. ↵ Goto, S., Nishioka, T., Kanehisa, M. 1998LIGAND: chemical **database** for enzyme ...

Spominje se 2220 puta Srodni članci Svih 16 inačica Web of Science: 1263 Citiraj Spremi

**[NAVOD] New trends in seismic design methodology**

**H Krawinkler** - 1995 - EUROPEAN CONFERENCE ON ...

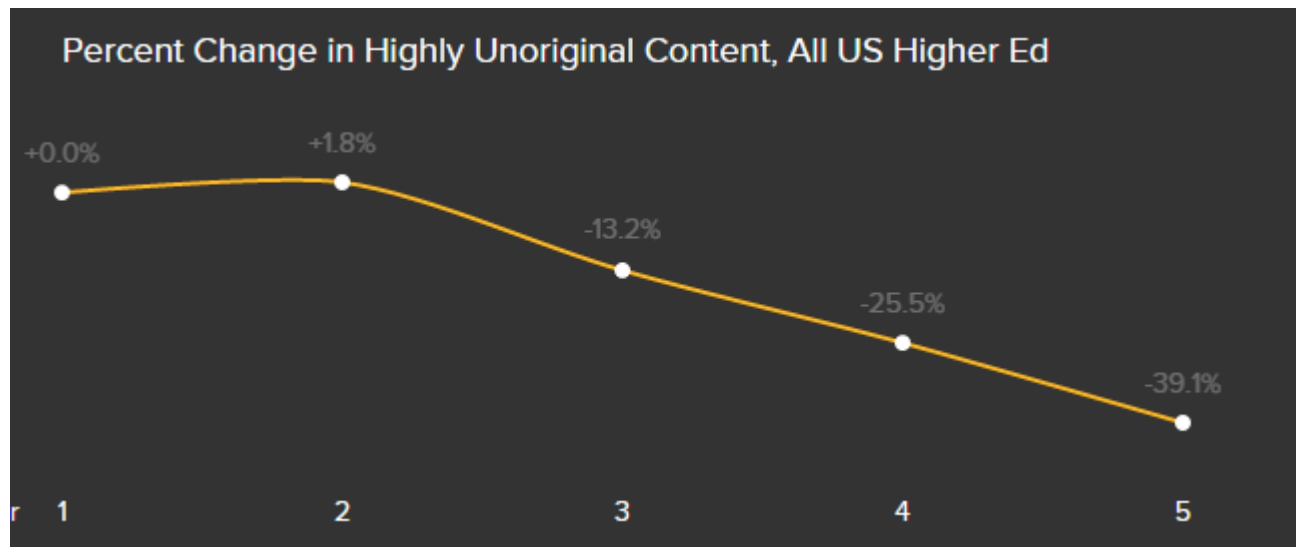
**[PDF] od iapnrfgm.org**  
**EBSCOhost Full Text**

**[HTML] od oxfordjournals.org**

**[HTML] od oxfordjournals.org**

## Motivacijski primjer 2

- AKTIVIRAN LOVAC NA PLAGIJATE Softver će provjeravati sve seminarske i diplomske radove  
(Jutarnji list 22.09.2014)
- Računalni program Turnitin u bazama radova i na oko 35 milijuna web stranica tražit će slične rečenice, reference i citate
- Zakonska obaveza je javno objavljivati diplomske i doktorske radove, a Sveučilište u Rijeci planira kontrolu plagiranja provesti uz pomoć Turnitina (<http://turnitin.com/>).



[http://turnitin.com/assets/en\\_us/media/effectiveness-calculator-us-he/](http://turnitin.com/assets/en_us/media/effectiveness-calculator-us-he/)

## Zbog čega je pretraživanje teksta izazov?

- Tekst pisan prirodnim jezikom podliježe pravopisnim i gramatičkim pravilima jezika. Mnogo jezika, mnogo pravila.
- Procjenjuje se da 80% poslovnih podataka pripada kategoriji nestrukturiranih. Većina je u tekstualnom obliku. Upravljanje (pohrana, pretraga) velikim količinama podataka predstavlja tehnički i tehnologijski izazov.
- Istraživanje u području pretraživanja teksta zahtijeva znanja eksperta iz područja jezikoslovlja (lingvistike) i računarstva.
- Tekst pisan prirodnim jezikom je često *nejasan* i *dvosmislen*

# Nejasan, dvosmislen

- Nejasan

Rezultati pretrage za **jeftin skuter**

1 2 3 4 5 6 7-8 SLJEDEĆA »

148 oglasa Sortiraj Relevantnosti ▼

Njuškalo oglasi



**JEF TINI BIKIKLI I SKUTERI !!** ★

Prodajem 3 bicikla i 2 skutera. Bicikli i skuteri su stari cca. 1-2 godine sve radi ispravno.

Objavljen: 14.08.2015.

300 kn



**PRODAJE SE SKUTER 200 KUBA PIAGGIO hitno i jeftino!!** ★

Rabljeni motor, 35000 km  
Godina proizvodnje: 2005.

Objavljen: 04.08.2015.

1.750 € ~ 13.239 kn

- Dvosmislen

- Na prodaju je kuća mog prijatelja u Splitu.
- Kuća mog prijatelja u Splitu je na prodaju.

## Pretraživanje teksta (Full Text Search - FTS)

- Traženje dokumenata koji zadovoljavaju postavljeni **uvjet** i njihovo rangiranje u skladu sa **sličnošću** dokumenta s postavljenim uvjetom.
- **Uvjet** je obično niz riječi, a **sličnost** je brojčana vrijednost koja je u najjednostavnijoj implementaciji povezana s frekvencijom riječi iz uvjeta u pretraživanom dokumentu.



## Zašto pretraživati tekst u relacijskim bazama podataka?

- Zbog velike količine tekstualnih podataka pohranjenih u relacijskim bazama podataka
- Jer tekst pohranjen u relacijskim bazama podataka najčešće
  - nije redundantan
  - ne sadrži nikakve dodatke - tipa oznake (HTML, XML)
  - je najbolje granularnosti – odgovarajuća razina detalja

## Tekst u relacijskim bazama podataka i standardni SQL

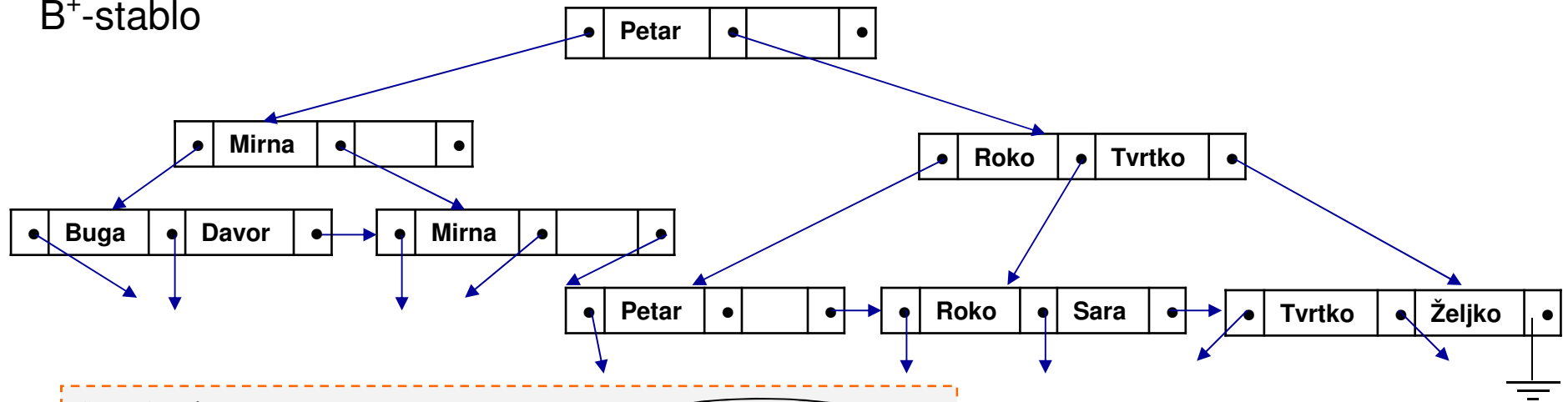
- tipovi podatka
  - CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT; NATIONAL CHARACTER, NATIONAL CHARACTER VARYING, NATIONAL CHARACTER LARGE OBJECT
  - npr. PostgreSQL: CHAR(n), VARCHAR(n), TEXT  
IBM Informix: CHAR (m,r), NCHAR(m,r), VARCHAR (m,r),  
LVARCHAR (m,r), TEXT
- funkcije:
  - SUBSTRING, UPPER, LOWER, TRIM, CHAR\_LENGTH, OCTET\_LENGTH
- operatori:
  - ||,  
LIKE, NOT LIKE (specijalni znakovi '\_' i '%')  
SIMILAR, NOT SIMILAR

## Nedostaci standardnog relacijskog SUBP u pretraživanju teksta

- Ne postoje efikasni indeksi pa pri procesiranju svakog upita treba „obraditi” svaki dokument  $\Rightarrow$  sporo

```
...  
CREATE INDEKS imeOsobaIdx on osoba (ime);  
...
```

B<sup>+</sup>-stablo



```
SELECT *  
FROM osoba  
WHERE ime LIKE 'Ana%';
```

```
SELECT *  
FROM osoba  
WHERE ime LIKE '%Ana';
```

koristi index

ne koristi  
index

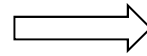


## Nedostaci standardnog relacijskog SUBP u pretraživanju teksta

- Pronalaženje riječi koje ne želimo  $\longrightarrow$  točnost

npr. želimo pronaći zapise koji sadrže riječ „dol”

```
SELECT *  
FROM ...  
WHERE tekst LIKE '%dol%';
```



```
dol  
dolje  
nadoliti  
dolar
```

- Ne postoji lingvistička podrška čak niti za engleski jezik
  - nema načina da se prepoznaju izvedene riječi, npr. *podatak* i *podatkovni*
  - nemoguće je zanemariti veznike, prijedloge i ostale riječi koje se često pojavljuju u tekstovima a nemaju semantičko značenje
  - nemoguće je dovesti u vezu riječi istog značenja (sinonimi)
- Ne postoji rangiranje rezultata što rezultat s mnogo dokumenata čini neupotrebljivim

# Pristupi pri pretraživanju teksta

1. Točno podudaranje
  - *uvjet* i tekst se u cijelosti podudaraju, identični su
  - *uvjet* je sadržan u dijelu teksta (djelomično podudaranje)
2. Podudaranje temeljem morfologije, sintakse i semantike jezika
  - koriste se tzv. gramatički algoritmi koji pronalaze podudarnost između *uvjeta* i teksta temeljem normaliziranog oblika riječi (korijena, leksema), temeljem sinonima i td.  
Npr. riječi *podatak*, *podatci*, *podatka*, *podatkom*, *podatku*,... imaju jednak korijen
3. Približno podudaranje (fuzzy)
  - Algoritmi temeljeni na „udaljenosti” između znakovnih nizova
  - Q-Gram algoritmi
  - Fonetsko podudaranje

# PostgreSQL: Pretraživanje teksta

## 1. Točno podudaranje

- =
- LIKE, NOT LIKE
- SIMILAR TO (SQL:1999 (SQL3))
- Regularni izrazi (*regular expressions* – *regex*)

Standardne mogućnosti SQL-a  
u relacijskim SUBP

## 2. Podudaranje temeljem morfologije, sintakse i semantike jezika

- TSVector, TSQuery tipovi podataka, @@ operator

## 3. Približno podudaranje

- Algoritmi temeljeni na „udaljenosti” između znakovnih nizova
  - Levenshtein funkcija
- Q-Gram algoritmi
  - % operator, similarity funkcija
- Fonetsko podudaranje
  - Soundex, Metaphone funkcije

# Točno podudaranje: LIKE – NOT LIKE

`znakovniNiz LIKE uzorak [ESCAPE specijalniZnak]`

Znakovi s posebnim značenjem:

- `%` (zamjena za 0 ili više proizvoljnih znakova)
- `_` (zamjena za točno 1 proizvoljan znak)

foaf

osoba1	osoba2
ana123@hotmail.com	zecG@gmail.com
zecG@gmail.com	iva.malic@fer.hr
klaraB@gmail.com	ana123@hotmail.com
zecG@gmail.com	jezV@hotmail.com
Iva.malic@fer.hr	jezV@hotmail.com

```
SELECT * FROM foaf
WHERE osoba1 LIKE '%gmail%';
```

```
SELECT * FROM foaf
WHERE osoba1 LIKE '%gmail_com';
```

osoba1	osoba2
ana123@hotmail.com	zecG@gmail.com

osoba1	osoba2
ana123@hotmail.com	zecG@gmail.com

# Točno podudaranje: SIMILAR TO

`znakovniNiz SIMILAR TO uzorak [ESCAPE specijalniZnak]`

Znakovi s posebnim značenjem:

- `|` ili - jedna od dvije mogućnosti
- `*` ponavljanje prethodnog znaka/znakova 0 ili više puta
- `+` ponavljanje prethodnog znaka/znakova 1 ili više puta
- `?` ponavljanje prethodnog znaka/znakova 0 ili 1 put
- `()` grupiranje znaka/znakova u jednu logičku cjelinu
- `{m}` ponavljanje prethodnog znaka/znakova točno m puta
- `{m,}` ponavljanje prethodnog znaka/znakova m ili više puta
- `{m,n}` ponavljanje prethodnog znaka/znakova najmanje m i najviše n puta
- `[...]` specificira klasu znakova, kao u regularnim izrazima
- ...

```
SELECT * FROM foaf
WHERE osoba1 SIMILAR TO '%(hot|g)mail%'
```

foaf

osoba1	osoba2
ana123@hotmail.com	zecG@gmail.com
zecG@gmail.com	iva.malic@fer.hr
klaraB@gmail.com	ana123@hotmail.com
zecG@gmail.com	jezV@hotmail.com
Iva.malic@fer.hr	jezV@hotmail.com

osoba1	osoba2
ana123@hotmail.com	zecG@gmail.com
zecG@gmail.com	iva.malic@fer.hr
klaraB@gmail.com	ana123@hotmail.com
zecG@gmail.com	jezV@hotmail.com

# Točno podudaranje: Regularni izrazi

```
znakovniNiz ~ regularniIzraz
~          odgovara
!          negacija (!~   ne odgovara)
*          neosjetljiv na velika i mala slova (!~*)
```

~	Odgovara regularnom izrazu, osjetljiv na velika i mala slova
~*	Odgovara regularnom izrazu, nije osjetljiv na velika i mala slova
!~	Ne odgovara regularnom izrazu, osjetljiv na velika i mala slova
!~*	Ne odgovara regularnom izrazu, nije osjetljiv na velika i mala slova

```
SELECT * FROM foaf
WHERE osoba1 ~* '^zecg*'
```


\* kod regularnih izraza ima jednako značenje kao % kod LIKE

foaf

^ je oznaka za početak znakovnog niza

osoba1	osoba2
ana123@hotmail.com	zecG@gmail.com
zecG@gmail.com	iva.malic@fer.hr
klaraB@gmail.com	ana123@hotmail.com
zecG@gmail.com	jezV@hotmail.com
Iva.malic@fer.hr	jezV@hotmail.com

osoba1	osoba2
ana123@hotmail.com	zecG@gmail.com
klaraB@gmail.com	ana123@hotmail.com
Iva.malic@fer.hr	jezV@hotmail.com



## Pretraživanje teksta temeljem morfologije, sintakse i semantike jezika

- **Morfologija** ili **oblikoslovlje** je grana jezikoslovlja koja proučava načine na koji se riječi u nekom jeziku oblikuju i mijenjaju. Na koji način se nosioci osnovnog (korjenitog) značenja dograđuju i pregrađuju putem morfema da bi se izgradile razne vrste riječi.
- **Sintaksa** je dio jezikoslovlja koji proučava odnose među:
  - riječima u rečenici
  - surečenicama (u složenim rečenicama)
  - rečenicama u tekstu
- **Semantika** je potpolje jezikoslovlja posvećeno proučavanju značenja
  - značenje uključuje razne aspekte jezika, između ostalog: sinonime, antonime, hiperonime, hiponime i td.

# Pretraživanje teksta temeljem morfologije, sintakse i semantike jezika

**Morfem** - najmanja jezična jedinica koja ima i svoj oblik i svoje značenje  
primjer: u gleda-m, jedinica m ima značenja: „prvo lice”, „jednina”, „sadašnje vrijeme”.

**Morf** - ostvaraj morfema, odnosno fizički oblik. Isti morfem u svim oblicima iste riječi ili u različitim riječima nema uvijek isti izraz tj. isti morf. npr. morfem *rek* značenja „usmeno priopćiti” ima morf: *reč*, *rek*, *rec*, *re*. Oni se pojavljuju u oblicima *rečeš*, *rekla*, *recimo*, *reći*

- **Korijenski** morf - obavezni, neizostavni dio oblika riječi, nositelj temeljnog značenja (npr. *podvodni*; složenice imaju više korijena npr. *vodopad*)
- **Afiks** - svaki morf koji nije korijenski, morf koji se pričvršćuje na bazu
- **Baza** - bilo koji segment, odsječak (morf ili slijed morfova) na koji se pričvršćuje afiks



# Pretraživanje teksta temeljem morfologije, sintakse i semantike jezika


- Afiksi prema mjestu pričvršćivanja na bazu:
  - **prefiksi** (pra-djed, na-učiti, pre-po-znati)
  - **sufiksi** (npr. vod-a, vod-e; radi-m, radi-š )
  - **interfiksi** (glav-o-bolja, nog-o-met, ka<sup>ž</sup>-i-prst, cjepi-i-dlaka)
  - **infiksi**, **cirkumfiksi**, **transfiksi** i **superfiksi**
- Glasovne promjene
  - Jotacija: mlad – mlađi (mlad+ji)
  - Sibilizacija: majka – majci
  - Palatalizacija: ptica – ptičica
  - Nestojanje a: sestra – sestara
  - ...
- ...

## Zbog čega nam je morfologija, sintaksa i semantika jezika važna?

- Kada **uvjet** npr. sadrži riječ **sreća** očekujemo da su u rezultatu dokumenti koji sadrže riječ **sreća** ali i riječi
  - **sretnik, sretnica, nesretnik,...**  $\Rightarrow$  morfologija
  - **radost, veselje, blaženstvo,...**  $\Rightarrow$  semantika (sinonimi)
- Da bi to bilo moguće moramo poznavati morfološke operacije (načine građenja oblika riječi neovisno o svrsi te gradnje), te sintaktička i semantička pravila jezika
- Jezikoslovlje nije tema ovog predmeta.
- Navedeni pojmovi i primjeri samo ilustriraju problem traženja i prepoznavanja **uvjet-a** u pretraživanom tekstu.

## Pretraživanje teksta temeljem morfologije, sintakse i semantike jezika

- Tekst/dokument je (multi) skup riječi (bag of words)
- Efikasna pretraga teksta podrazumijeva prethodnu obradu teksta (dokumenta):
  - parsiranje teksta i rastavljanje na tokene (riječi, brojevi, tagovi, razmak, url,...)
  - uklanjanje riječi koje nemaju semantičko značenje u tekstu (stop riječi)
  - utvrđivanje korijena za svaki token  
(npr. *stan* u riječima *stanar*, *stanari*, *stanarski*,...  
ili *bank* za riječi *banking*, *banked*, *banks*, *banks'* i *bank's*)
  - identificiranje sinonima
  - pohranu obrađenog teksta u obliku prikladnom za pretragu



## Pretraživanje teksta temeljem morfologije, sintakse i semantike jezika

- Rječnici su podloga za kvalitetno konvertiranje tokena;
  - dobar rječnik treba omogućiti:
    - definiranje stop riječi, koje ne treba indeksirati
    - definiranje sinonima
    - stvaranje veza između fraza i pojedinih riječi
    - ...

# PostgreSQL: FTS (temeljem morfologije, sintakse i semantike jezika)

Podržan je pomoću:

1. Novih tipova podataka TSVector i TSQuery te skupa pravila za transformaciju teksta/dokumenata i upita u njihove FTS reprezentante
    - parser, rječnici
  2. Skupa funkcija za transformaciju tekstualnih podataka u TSVector i TSQuery
    - *to\_tsVector*, *to\_tsquery*, *plainto\_tsquery*
  3. Funkcija za rangiranje rezultata
    - *rank\_cd*, *rank*,
- TSVector
    - podatkovni tip za reprezentaciju dokumenta
  - TSQuery
    - podatkovni tip za reprezentaciju tekstualnog upita
  - @@
    - FTS operator za rad sa TSVector i TSQuery

# PostgreSQL: Parser

- Razdvaja izvorni tekst na tokene i utvrđuje tip tokena
- Tipovi tokena koje parser prepoznaje unaprijed su definirani
- Ne modificira izvorni tekst
- Ugrađeni parser PostgreSQL-a razlikuje 23 tokena: asciiword, word, numword, email, protocol, url, host, file, tag, blank,...
- ***ts\_debug*** funkcije za testiranje parametara pretrage; za svaki token teksta, predanog kao argument, prikazuje informaciju u skladu s trenutnim parametrima pretrage

```
SELECT alias, description, token
FROM ts_debug ('The Dancing Ladies');
```

alias	description	token
asciiword	Word, all ASCII	The
blank	Space symbols	
asciiword	Word, all ASCII	Dancing
blank	Space symbols	
asciiword	Word, all ASCII	Ladies

```
SELECT alias, description, token
FROM ts_debug (
'http://www.fer.unizg.hr/oferu/podaci')
```

alias	description	token
protocol	Protocol head	http://
url	URL	www.fer.unizg.hr/oferu/podaci
host	Host	www.fer.unizg.hr
url path	URL path	www.fer.unizg.hr/oferu/podaci

## PostgreSQL: Rječnici

- Koriste za se uklanjanje stop riječi, normalizaciju teksta koja omogućuje prepoznavanje riječi s jednakim normaliziranim oblikom (korijenom/leksemom), prepoznavanje različitih riječi jednakog značenja (sinonimi),...
- Time se reducira veličina (tsvector) reprezentacije dokumenta i postižu dobra svojstva pretraživanja teksta.
- PostgreSQL ima nekoliko tipova rječnika:
  - Simple Dictionary  
Uklanja stop riječi i velika slova svodi na mala.
  - Synonym Dictionary  
Različite riječi jednakog značenja zamjenjuje reprezentantnom rječju.
  - Thesaurus Dictionary  
Prepoznaje fraze.
  - iSpell Dictionary  
Svodi riječi na normalizirani oblik.
  - Snowball Dictionary  
Algoritamski svodi riječi na korijenski oblik (stemming) i uklanja stop riječi

# PostgreSQL: Rječnici

- Rječnik je program koji za ulazni *token* vraća:
  - polje leksema ako rječnik prepoznaje token (jedan token može imati više leksema)
  - prazno polje ako rječnik prepoznaje leksem ali predstavlja stop riječ
  - NULL ako rječnik ne prepoznaje token

```
SELECT *  
FROM ts_debug('english', 'The Dancing Ladies');
```

alias	description	token	dictionaries	dictionary	lexemes
asciiword	Word, all ASCII	The	{english_stem}	english_stem	{}
blank	Space symbols				
asciiword	Word, all ASCII	Dancing	{english_stem}	english_stem	{danc}
blank	Space symbols				
asciiword	Word, all letters	Ladies	{english_stem}	english_stem	{ladi}

Za testiranje ponašanja konkretnog rječnika može se koristiti `ts_lexize` funkcija  
`ts_lexize(dict regdictionary, token text) returns text[]`



# PostgreSQL: Rječnici

- Različiti tokeni mogu rezultirati jednakom listom leksema

```
SELECT *  
FROM ts_debug ('english', 'happy happiness happily');
```

alias	description	token	dictionaries	dictionary	lexemes
asciiword	Word, all ASCII	happy	{english_stem}	english_stem	{happi}
blank	Space symbols				
asciiword	Word, all ASCII	happiness	{english_stem}	english_stem	{happi}
blank	Space symbols				
asciiword	Word, all letters	happily	{english_stem}	english_stem	{happili}

# PostgreSQL: Rječnici

- PostgreSQL ima inicijalne rječnike za raširenije svjetske jezike.
- Postoje i predlošci koji se mogu koristiti za kreiranje novih rječnika.
- Novokreirani rječnici se konfiguriraju pomoću parametara.

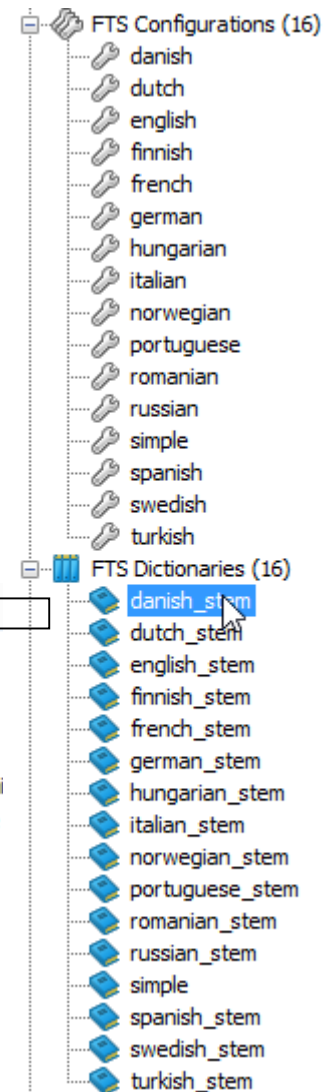
```
CREATE TEXT SEARCH DICTIONARY name  
( TEMPLATE = template  
[, option = value [, ... ]] )
```

```
ALTER TEXT SEARCH DICTIONARY name  
( option [ = value ] [, ... ] )
```

```
CREATE TEXT SEARCH DICTIONARY croatian  
( template = snowball  
, language = croatian  
, stopwords = croatian);
```

**ERROR: no Snowball stemmer available for language "croatian" and encoding "UTF8"**

Property	Value
Name	danish_stem
OID	12318
Owner	postgres
Template	snowball
Options	language = 'danish', stopwords = 'dani
Comment	snowball stemmer for danish language



## PostgreSQL: Stop (zaustavne) riječi

- Riječi koje se vrlo često pojavljuju, gotovo u svakom dokumentu
- Ignoriraju se pri pretrazi teksta izuzev pri traženju fraza
- Npr. tekstovi na engleskom jeziku redovito sadrže riječi "a" i "the" te bi njihovo uključivanje u proces pretrage vratilo sve dokumente
- Utječu na rangiranje dokumenata – vidi slajd o rangiranju kasnije
- Liste stop riječi za raširenije svjetske jezike dolaze s inicijalnom instalacijom PostgreSQL-a

```
-bash-4.1$ pwd
/usr/pgsql-9.3/share/tsearch_data
-bash-4.1$ ls
danish.stop      hungarian.stop      portuguese.stop      turkish.stop
dutch.stop       hunspell_sample.affix  russian.stop          unaccent.rules
english.stop     ispell_sample.affix   spanish.stop          xsyn_sample.rules
finnish.stop    ispell_sample.dict    swedish.stop
french.stop      italian.stop          synonym_sample.syn
german.stop      norwegian.stop        thesaurus_sample.ths
```

```
i
me
my
myself
we
our
ours
ourselves
you
your
yours
yourself
yourselves
he
him
his
himself
she
her
hers
herself
it
its
"english.stop"
```

# PostgreSQL: Rječnik sinonima

- Obavlja zamjenu riječi s njenim sinonimom
- Omogućava pronalaženje dokumenata koji sadrže riječ ili njen sinonim

```
postgres      pgsq
postgresql    pgsq
postgre pgsq
gogle googl
indices index*
~
"synonym_sample.syn" [readonly] 5L, 73C
```

```
[root@localhost tsearch_data]# cp synonym_sample.syn my_syn.syn
[root@localhost tsearch_data]# ls *syn
my_syn.syn  synonym_sample.syn
```

```
postgres      pgsq
postgresql    pgsq
postgre pgsq
gogle googl
indices index*
enjoyment     happiness
gladness      happiness
contentment   happiness
~
"my_syn.syn" 8L, 136C written
```

```
CREATE TEXT SEARCH DICTIONARY my_xsyn
(TEMPLATE = synonym,
SYNONYMS = my_syn)
```

```
SELECT ts_lexize
('my_xsyn', 'enjoyment');
```

```
SELECT ts_lexize
('my_xsyn', 'gladness');
```

ts\_lexize

{happiness}

ts\_lexize

{happiness}

- Više od dvije riječi mogu biti međusobno sinonimi

```
ALTER TEXT SEARCH DICTIONARY xsyn
(RULES= my_rules, KEEPORIG=true);
```

```
SELECT ts_lexize('xsyn', 'happiness');
```

ts\_lexize

{happines, enjoyment, gladness, contentment}

```
# word synonym1 synonym2 ...
#
supernova sn sne 1987a
happiness enjoyment gladness contentment
~
"my_rules.rules" 7L, 180C written
```

# PostgreSQL: Povezivanje parsera i rječnika

- Konfiguracijskim parametrima se parser povezuje sa skupom rječnika pomoću kojih se rezultat parsiranja dalje obrađuje.

```
CREATE TEXT SEARCH CONFIGURATION name (  
    PARSER = parser_name | COPY = source_config )
```

- Za svaki tip tokena definira se lista rječnika i redoslijed kojim se obilaze pri normalizaciji tokena
- Rječnici se pozivaju navedenim redoslijedom
  - Ako rječnik prepozna token, rječnici nakon njega se ne konzultiraju
  - Ako rječnik ne prepozna token (vrati NULL), token se proslijeđuje sljedećem rječniku
- Obično se na početak liste stavlja najspecifičniji rječnik, potom općenitiji rječnici a na kraj liste najopćenitij rječnik (kao Snowball) koji prepoznaje svaku riječ

# PostgreSQL: Konfiguriranje opcija pretraživanja teksta

Inicijalno za tokene tipa *word* i *asciiword* se pretražuje samo *english\_stem* rječnik:

```
SELECT *  
FROM ts_debug('english', 'The Dancing Ladies');
```

alias	description	token	dictionaries	dictionary	lexemes
asciiword	Word, all ASCII	The	{english_stem}	english_stem	{}
blank	Space symbols				
asciiword	Word, all ASCII	Dancing	{english_stem}	english_stem	{danc}
blank	Space symbols				
asciiword	Word, all letters	Ladies	{english_stem}	english_stem	{ladi}

```
ALTER TEXT SEARCH CONFIGURATION english  
ALTER MAPPING FOR word, asciiword WITH xsyn, english_stem;
```

```
SELECT *  
FROM ts_debug('english', 'The Dancing Ladies');
```

alias	description	token	dictionaries	dictionary	lexemes
asciiword	Word, all ASCII	The	{xsyn, english_stem}	english_stem	{}
blank	Space symbols				
asciiword	Word, all ASCII	Dancing	{xsyn, english_stem}	english_stem	{danc}
blank	Space symbols				
asciiword	Word, all letters	Ladies	{xsyn, english_stem}	english_stem	{ladi}

# PostgreSQL: TSVector

- TSVector

- podatkovni tip koji reprezentira dokument i optimiran je za FTS pa obuhvatnije pretražuje tekst od npr. LIKE, SIMILAR TO i ~

- sortirana lista leksema

primjeri preuzeti iz PostgreSQL dokumentacije

```
SELECT 'a fat cat sat on a mat and ate a fat rat'::TSVector;
```

TSVector
'a' 'on' 'and' 'ate' 'cat' 'fat' 'mat' 'rat' 'sat'

- TSVector tip podatka ne provodi normalizaciju (uklanjanje stop riječi, svođenje na korijen,...)

```
SELECT 'The fat rats'::TSVector;
```

TSVector
'fat' 'The' 'rats'

- funkcija *to\_tsvector* provodi normalizaciju:

```
SELECT to_tsvector ('english', 'The fat rats')
```

To_tsVector
'fat':2 'rat':3

'rats'?

Zašto nema 'The'?

Čemu služe brojevi?

# PostgreSQL: TSQuery

- TSQuery
  - podatkovni tip za predstavljanje upita s podrškom za Booleove operatore & (AND) i | (OR)
  - sastoji se od leksema povezanih Booleovim operatorima

```
SELECT 'The & fat & rats'::TSQuery;
```

tsquery
'The' & 'fat' & 'rats'

```
SELECT 'fat & (rats | cat)'::TSQuery;
```

tsquery
'fat' & ('rats'   'cat')

- funkcije *to\_tsquery* i *plainto\_tsquery* provode normalizaciju:

```
SELECT to_tsquery ('english', 'The & fat | rats')
```

to_tsquery
'fat'   'rat'

- *plainto\_tsquery* lekseme uvijek povezuje s &:

```
SELECT plainto_tsquery ('english', 'The fat rats')
```

plainto_tsquery
'fat' & 'rat'



## PostgreSQL: FTS operator @@

- operator za rad s TSVector i TSQuery
- rezultat je *true* ako TSVector (dokument) odgovara TSQuery (upit)
- podržava i TEXT i VARCHAR
- jednostavna podrška za FTS (bez rangiranja rezultata)

```
TSVector @@ TSQuery  
TSQuery  @@ TSVector  
TEXT/VARCHAR @@ TEXT/TSQuery
```

```
SELECT 'a fat cat sat on a mat and ate a fat rat'::TSVector @@  
       'cat & rat':: tsquery
```

?column?

t

```
SELECT 'a fat cat sat on a mat and ate a fat rat'::TSVector @@  
       'cat & dog':: tsquery
```

?column?

f

# PostgreSQL: FTS operator @@ - primjer

movies

movieid	title	To_tsvector(title)
67	Dirty Dancing	""danc':2 'dirti':1"
6569	Dances with Wolves	""danc':1 'wolv':3"
78	Shall We Dance?	""danc':3 'shall':1"
368	The Dancing Masters	""danc':2 'master':3"
90634	The Man Who Loved Cat Dancing	""cat':5 'danc':6 'love':4 'man':2"

```
SELECT title FROM movies
WHERE title::TSVector @@ 'Dancing'::TSQuery
```

title
Dirty Dancing
The Dancing Masters
The Man Who Loved Cat Dancing

```
SELECT title FROM movies
WHERE to_TSVector(title) @@
      to_TSQuery('Dancing')
```

title
Dirty Dancing
Dances with Wolves
Shall We Dance?
The Dancing Masters
The Man Who Loved Cat Dancing

```
SELECT title FROM movies
WHERE to_TSVector(title) @@
      to_TSQuery('The & Dencing & Master')
```

title
-------

# PostgreSQL: @@ brzina?

- Provođenje normalizacije dokumenata u velikom korpusu pri svakom obavljanju upita je sporo

## ESPERIMENT

- Tablica test100k sadrži 100 000 ntorki

```
CREATE TABLE test100K(  
    tekst    VARCHAR(2500)  
);
```

- Sadržaj preuzet s <http://www.anc.org/data/masc/downloads/data-download/>
- U tablici postoji 1 n-torka sa sadržajem  
**'Yeah well, I am lucky I will never lose'**
- Testiram brzinu obavljanja upita:

```
SELECT * from test100k  
WHERE TO_tsVector('english', tekst)  
      @@  
      TO_tsquery('english',  
                  'Yeah & well & I & am & lucky I & will & never & lose');
```

Trajanje: 3570 ms

## PostgreSQL: @@ EKSPERIMENT

- Tablica test1M sheme kao test100K sadrži 1 000 000 ntorki
- Sadržaj je udeseterostručeni sadržaj tablice test100K
- Trajanje ekvivalentnog upita: **27415ms (27 s)**
- Tablica test10M sheme kao test100K sadrži 10 000 000 ntorki
- Sadržaj je udeseterostručeni sadržaj tablice test1M
- Trajanje ekvivalentnog upita: **277735 ms (4.63 min)!**
- 

- Što napraviti?
- **Može li brže?**

tablica	Trajanje (ms)
test100K	3570 (3s)
test1M	<b>27415 (27 s)</b>
test10M	<b>277735 (4.63 min)</b>

# PostgreSQL: @@ EKSPERIMENT

Poboljšanje, prva verzija:

- Dodati atribut tekstTSV tipa TSVECTOR, koji će sadržavati aktualnu reprezentaciju teksta u normaliziranom obliku
- U upitima koristiti tekstTSV umjesto tekst

```
CREATE TABLE test100K
(
    tekst    VARCHAR(2500),
    tekstTSV TSVECTOR
);
```

```
CREATE TRIGGER test100K_InsUpd_Trigg
BEFORE INSERT OR UPDATE ON test100K
FOR EACH ROW
EXECUTE PROCEDURE tsvector_update_trigger
(tekstTSV, 'english', tekst);
```

```
SELECT * from test100k
WHERE tekstTSV @@
      TO_tsquery('english', 'Yeah & well & I & am & lucky I & will & never & lose');
```

Rezultati:

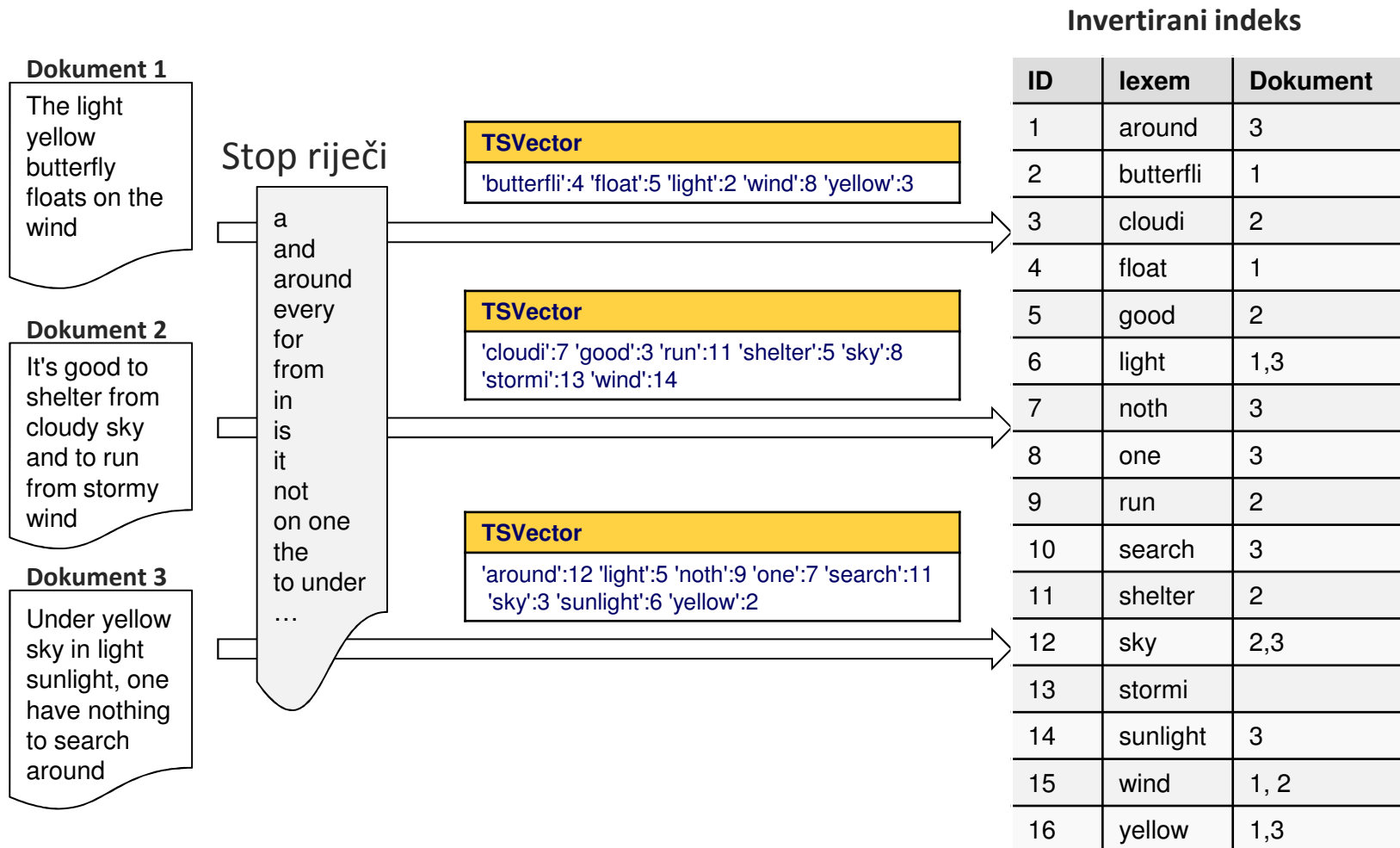
tablica	Trajanje (ms)	
	tekst	tekstTSV
test100K	3570 (3 s)	1000 (1s)
test1M	27415 (27 s)	9648 (9,6s)
test10M	277735 (4.63 min)	58920 (59s)

Može li brže?

Može! Kako?

# Invertirani indeksi

- Ubrzavaju FTS
- Važno je da se izrađuju nad normaliziranim dokumentima



# PostgreSQL: Specijalni indeksi za FTS

- GIN indeks (Generalized Inverted Index)

- atribut mora biti tipa TSVector

```
CREATE INDEX idxName ON tableName USING gin(attrName)
```

- GIST indeks (Generalized Search Tree)

- atribut mora biti tipa TSVector ili TSQuery

```
CREATE INDEX idxName ON tableName USING gist(attrName)
```

- GIN indeksi

- ✓ brža pretraga (oko 3x)
- ✓ Dulje traje izgradnja indeksa (oko 3x)
- ✗ sporiji UPDATE
- ✗ zauzimaju dva do tri puta više prostora za pohranu od GIST indeksa

# PostgreSQL: @@ Eksperiment

```
CREATE TABLE test100K
(
    tekst    VARCHAR(2500),
    tekstTSV TSVECTOR
);
```

```
CREATE TRIGGER test100K_InsUpd_Trigg
BEFORE INSERT OR UPDATE ON test100K
FOR EACH ROW
EXECUTE PROCEDURE tsvector_update_trigger
(tekstTSV, 'english', tekst);
```

```
* CREATE INDEX tekstTsv10M ON test10M USING gin(tekstTSV);
** CREATE INDEX tekstTsv10M ON test10M USING gist(tekstTSV);
```

```
SELECT * from test100k
WHERE tekstTSV @@
      TO_tsquery('english',
        'Yeah & well & I & am & lucky I & will & never & lose');
```

Rezultati:

tablica	Trajanje (ms)				Veličina indeksa (MB)	
	tekst	tekstTSV	tekstTSV s GIN indeksom	tekstTSV s GIST indeksom	GIN*	GIST**
test100K	3570 (3 s)	1000 (1s)	31	31	71	10
test1M	27415 (27 s)	9648 (9,6s)	804	7071	640	100
test10M	277735 (4.63 min)	58920 (59s)	1942	100234	1102	1008



## Približno pretraživanje teksta (Fuzzy Text Search)

- Tehnika pronalaženja dokumenta/niza znakova koji se približno podudara s traženim uzorkom
- Kvaliteta podudaranja se mjeri ovisno o vrsti primijenjenog algoritma npr.
  - brojem operacija koje je potrebno obaviti nad znakovnim nizom da bi se u potpunosti podudario s traženim uzorkom
  - brojem podudanih podnizova i sl.
- različite vrste algoritama:
  - algoritmi temeljeni na udaljenosti znakovnih nizova
    - Hamming, **Levenshtein**
  - Q-Gram algoritmi
    - slični znakovni nizovi imaju više zajedničkih Q-Gram-ova (podskupovi znakovnog niza duljine Q)
  - Soundex, Metaphone algoritam
    - traže riječi koje se slično izgovaraju
  - ...

# Algoritmi temeljeni na udaljenosti znakovnih nizova

- Udaljenost uređivanja („edit distance“) znakovnih nizova  $s_1$  i  $s_2$  je minimalan broj operacija potrebnih da se jedan niz transformira u drugi. Moguće operacije su: izmjena, umetanje, brisanje znaka.
- Autor ideje je Vladimir Levenshtein 1965. - Levenshteinova udaljenost
- Rješenje ne mora biti jednoznačno
- Problem se svodi na pronalaženje slijeda navedenih operacija kojim će se jedan niz transformirati u drugi uz minimalan trošak (navedene 3 operacije ne moraju jednako „koštati“)

Primjer: odrediti udaljenost između SREĆA i SRETNA

Pokušaj 1

1. Zamijeniti T sa Ć
2. Zamijeniti N s A
3. Obrisati A

Udaljenost = 3

S	R	E	Ć	A	
S	R	E	T	N	A

S	R	E	Ć	A	
S	R	E	Ć	N	A

S	R	E	Ć	A	
S	R	E	Ć	A	A

S	R	E	Ć	A	
S	R	E	Ć	A	

Pokušaj 2

1. Zamijeniti T sa Ć
2. Obrisati N

Udaljenost = 2

S	R	E	Ć	A	
S	R	E	T	N	A

S	R	E	Ć	A	
S	R	E	Ć	N	A

S	R	E	Ć	A	
S	R	E	Ć	A	

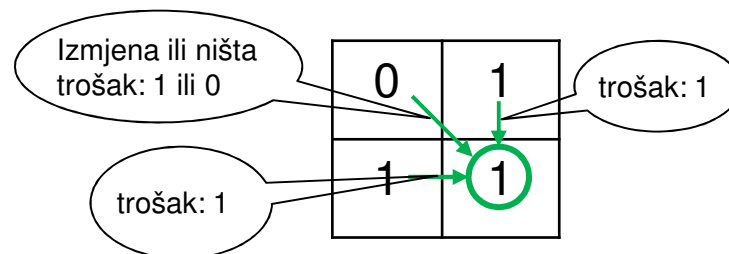
**Je li broj operacija manji ako mijenjam SREĆA u SRETNA?**

# Levenshteinova udaljenost

		s2						
s1			S	R	E	T	N	A
		0	1	2	3	4	5	6
	S	1	0	1	2	3	4	5
	R	2	1	0	1	2	3	4
	E	3	2	1	0	1	2	3
	Ć	4	3	2	1	1	2	3
	A	5	5	4	2	2	2	2

```

m = len(s1);
n = len(s2);
Inicijaliziraj prvi redak (0...m);
Inicijaliziraj prvi stupac (0...n);
for (i=1 to m)
    for (j=1 to n)
        ako je s1[i] == s2[j]
            trošak=0;
        inače
            trošak = 1;
        d[i,j] = min(d[i-1,j] + 1;
                    d[i,j-1] + 1;
                    d[i-1,j-1] + trošak)
Udaljenost = d[m,n];
    
```



Kretanje

udesno : umetanje znaka (u s1)

$(d[i-1, j] + 1)$

dolje : brisanje (iz s1)

$(d[i, j-1] + 1)$

dijagonalno: slaganje ili neslaganje uz izmjenu  $(d[i-1, j-1] + \text{trošak})$

# Levenshteinova udaljenost

Rezultat mora biti jednak kao u prethodnom postupku

		s2					
s1			S	R	E	Ć	A
		0	1	2	3	4	5
	S	1	0	1	2	3	4
	R	2	1	0	1	2	3
	E	3	2	1	0	1	2
	T	4	3	2	1	1	2
	N	5	4	3	2	2	2
	A	6	5	4	3	3	2

Linkovi za dodatna pojašnjenja:

<http://www-igm.univ-mlv.fr/~lecroq/seqcomp/node1.html>

<http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Fall2006/Assignments/editdistance/Levenshtein%20Distance.htm>

<http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Dynamic/Edit/>

<http://www-igm.univ-mlv.fr/~lecroq/seqcomp/node2.html>

# PostgreSQL: Levenshtein

Funkcija	Rezultat
levenshtein(text source, text target, int ins_cost, int del_cost, int sub_cost)	Cijeli broj - udaljenost između source i target
levenshtein(text source, text target)	

```
SELECT levenshtein ('The Dancing Masters', 'The Dancing Masters') lev1
      , levenshtein ('The Dancing Masters', 'The Dencing Master') lev2
      , levenshtein ('The Dancing Masters', 'Dancing Master') lev3
```

lev1	lev2	lev3
0	2	5

```
SELECT title FROM movies
WHERE levenshtein (lower(title), lower('The Dencing Master')) < 5
```

title
The Dancing Masters

```
SELECT title, length(title) tlen, length(title)/4 tlen14
      , levenshtein (lower(title), lower('The Dencing Master')) lev
FROM movies
WHERE levenshtein (lower(title), lower('The Dencing Master')) < length(title)/4
ORDER BY levenshtein (lower(title), lower('The Dencing Master')), length(title)/4
```

Link za dodatna pojašnjenja:

<http://www.postgresql.org/docs/9.4/static/fuzzystmatch.html>

title	tlen	tlen14	lev
The Dancing Masters	19	4	2

## Q-Gram algoritmi

- Tekst/dokument je (multi) skup n-grama
- n-gram može biti
  - riječ
  - znakovni niz, podskup teksta, duljine Q znakova
- Teza: ako je riječ A slična riječi B, one vjerojatno sadrže barem jedan podudaran (zajednički) podniz duljine Q

Primjer:  $q = 2$  (bigrami)

happiness:	ha	ap	pp	pi	in	ne	es	ss
happily:	ha	ap	pp	pi	il	ly		

- 4 podudarna od ukupno 10 bigrama (bez ponavljanja podudarnih)
- mjera sličnosti bi se mogla odrediti kao kvocijent:  $4/10 = 0,4$

## Q-Gram algoritmi

- Q-gram algoritam ne radi uvijek dobro – za kraće riječi i veći  $q$

Primjer:  $q = 3$  (trigrami)

votka : vot otk tka

vodka : vod odk dka

- 0 podudarnih od ukupno 6 trigrama  $\Rightarrow$  sličnost iznosi 0

- Prednost ove kategorije algoritama: primjenjivi bez obzira na jezik i domenu

## PostgreSQL: Q-Gram

- $Q = 3$  - trigram
- PostgreSQL dodaje 2 praznine na početak i 1 prazninu na kraj svakog niza

Primjer:

happiness	__h	_ha	hap	app	ppi	pin	ine	nes	ess	ss_
happily	__h	_ha	hap	app	ppi	pil	ily	ly_		

- 5 podudarnih od ukupno 13 trigrama (bez ponavljanja podudarnih)
- sličnost:  $5/13 = 0,384615$



# PostgreSQL: Trigram

Funkcija	Rezultat
<code>similarity(text, text)</code>	Realni broj $\in [0, 1]$ – mjera podudarnosti dva argumenta. 0 za potpuno različite, 1 za identične argumente.
<code>show_trgm(text)</code>	Polje trigrama ulaznog znakovnog niza.
<code>show_limit()</code>	Trenutni prag za podudarnost kojeg koristi operator %. Prag predstavlja minimalnu sličnost koja mora postojati između dva znakovna niza da bi se smatrali "sličnima".
<code>set_limit(real)</code>	Postavlja vrijednost praga podudarnosti kojeg koristi operator %. Prag mora biti između 0 i 1. Inicijalna vrijednost je 0.3.
<b>Operatori koji koriste funkciju similarity</b>	
<code>text % text</code>	<i>True</i> ako je podudarnost među argumentima veća od trenutnog praga podudarnosti .
<code>text &lt;-&gt; text</code>	Realni broj broj $\in [0, 1]$ koji predstavlja "udaljenost" između argumenata ( <code>1-similarity()</code> ).

Link za dodatna pojašnjenja:

<http://www.postgresql.org/docs/9.4/static/pgtrgm.html>

## PostgreSQL: Trigram - primjeri

```
SELECT show_trgm('happiness')
```

**Show\_trgm**

"{" h"," ha",app,ess,hap,ine,nes,pin,ppi,"ss {"

```
SELECT show_trgm('happily')
```

**Show\_trgm**

"{" h"," ha",app,hap,ily,"ly ",pil,ppi}"

```
SELECT similarity('happiness', 'happily')
```

**similarity**

0.384615

sličnost:  $5/13 = 0,384615$

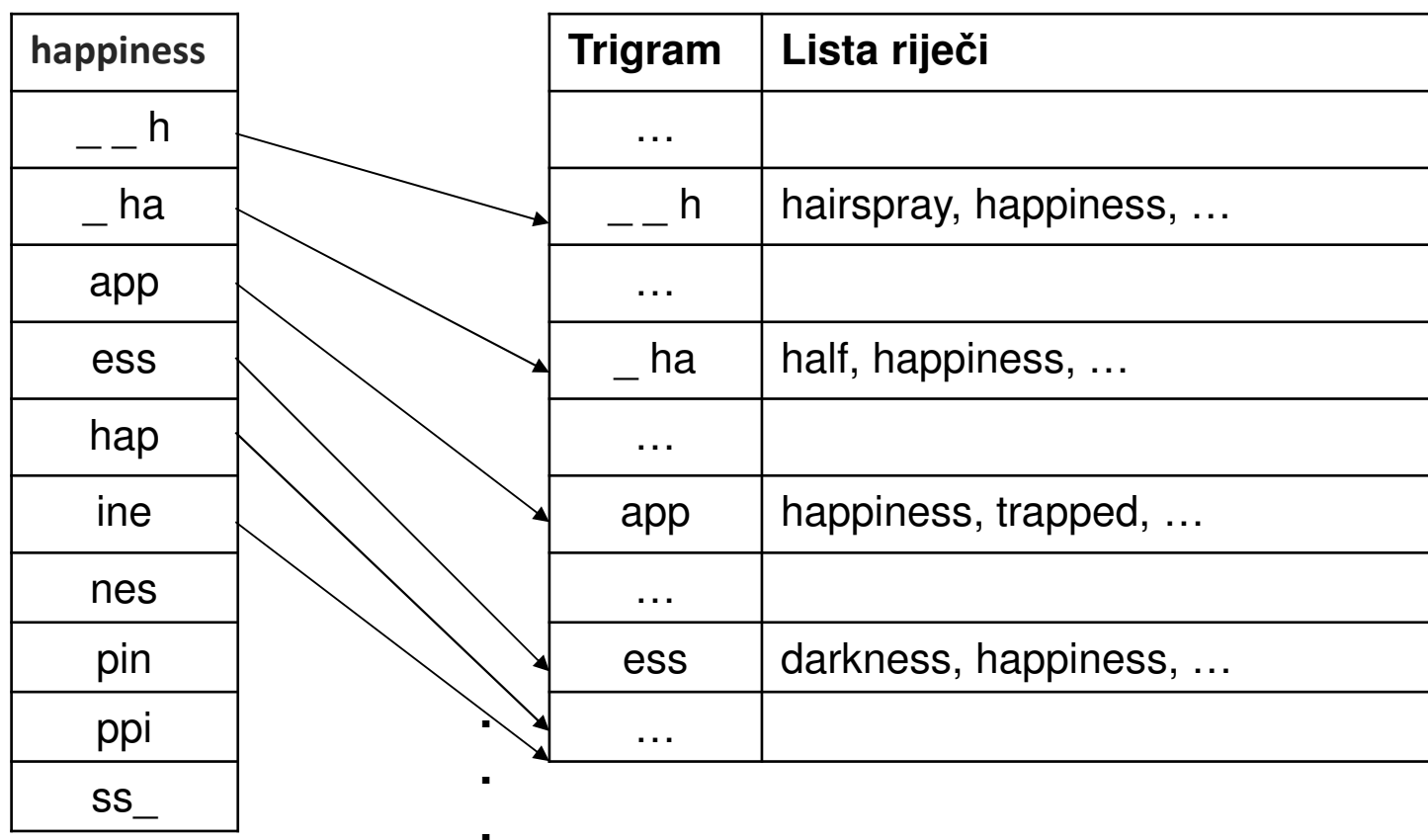
```
SELECT title, similarity (title, 'The Dencing Master')
FROM movies
WHERE title % 'The Dencing Master'
-- WHERE similarity (title, 'The Dencing Master') > 0.5
```

title	similarity
The Dancing Masters	0.625

Restriktivniji uvjet  
od inicijalnog praga

# PostgreSQL: Trigram

- može biti jako sporo za puno velikih dokumenata
- mogućnost kreiranja specijalnog (invertiranog) indeksa



## PostgreSQL: Trigram

- Invertirani indeks brzo "uparuje" vektor trigrama dokumenta s vektorom trigrama znakovnog niza koji se traži

```
CREATE INDEX title_trigram_idx ON movies USING gist(title gist_trgm_ops);
```

- *gist\_trgm\_ops* je opcija koja osigurava kreiranje indeksa za trigrame

# PostgreSQL: Soundex i Metaphone

- Aktualno u jezicima u kojima se izgovor riječi razlikuje od zapisa (ne pretjerano korisno za hrvatski)
- Ideja je dovesti u vezu riječi koje se jednako ili slično izgovaraju ali drugačije zapisuju
- Algoritam riječ predstavlja znakovnim nizom koji prezentira izgovor (zvučanje) riječi
- Soundex je ograničen na vlastita imena u engleskom jeziku

```
SELECT soundex('Anne') sAnne, soundex('Ann') sAnn
```

sanne	sann
A500	A500

- Metaphone koristi ideju algoritma Soundex ali nije ograničen na vlastita imena

```
metaphone(text source, int maxLength) returns text
```

```
SELECT metaphone ('Dancing', 7) m1, metaphone('Dencing', 7) m2
```

m1	m2
TNSNK	TNSNK

```
SELECT metaphone ('Bruce Wilis', 7) m1  
      , metaphone('Broos Wils', 6) m2
```

m1	m2
BRSWLS	BRSWLS

## PostgreSQL: Približno pretraživanje teksta - sažetak

- Levenshtein
  - prikladan za pronalaženje znakovnih nizova koji sadrže neznatne razlike (manje pravopisne pogreške) ali moraju biti jako slični
- Trigram
  - prikladan i za veća odstupanja (pravopisne pogreške)
- Soundex, Metaphone
  - prikladan za pronalaženje riječi (znakovnih nizova) koje se jednako ili slično izgovaraju

```
SELECT title
      , metaphone ('Dencing Master',15) m1, metaphone (title, 15) met
      , levenshtein (lower ('Dencing Master'), lower(title)) lv
      , similarity (lower ('Dencing Master'), lower(title)) sim
FROM movies
WHERE similarity (lower ('Dencing Master'), lower(title)) > 0.1
ORDER BY sim DESC, lv
```

title	met	lv	sim
The Dancing Masters	0TNSNKMSTRS	6	0.458333
Dirty Dancing	TRTTNSNK	12	0.217391
The Man Who Loved Cat Dancing	0MNHLFTKTTNSNK	25	0.184211

## Rangiranje rezultata

- Prema relevantnosti dokumenta/znakovnog niza za postavljeni upit
- Najbolji algoritam za rangiranje tek treba definirati
- Osnovni elementi: učestalost traženog znakovnog niza u dokumentu, podudarnost traženog znakovnog niza s dokumentom (ili njegovim dijelom) i u kojem dijelu dokumenta se traženi niz pojavljuje.
- Pretpostavka je da je relevantniji dokument u kojem se traženi uzorak riječi pojavljuje na bliskim pozicijama

Npr. je li za traženi uzorak *Važno je poznavati sebe.* relevantniji dokument

*Samopouzdanje se može povećati, važno je znati kako.*

ili dokument

*Nije važno što je potrebno znati.*

?

- Važno je znati na kojim pozicijama u tekstu se koja riječ pojavljuje (tsvector)
- Većina dokumenata ima sljedeće dijelove: naslov, ključne riječi, sažetak i tijelo
- Različitim dijelovima dokumenta ima smisla pri određivanju ranga dodijeliti različite težine – npr. najveću težini naslovu, a najmanju tijelu dokumenta

# PostgreSQL: Rangiranje rezultata

Funkcija	Rezultat
<code>ts_rank([ weights float4[], ] vector tsvector, query tsquery [, normalization integer])</code>	Realni broj $\in [0, 1]$ relevantnost dokumenta za upit
<code>ts_rank_cd([ weights float4[], ] vector tsvector, query tsquery [, normalization integer])</code>	

- `ts_rank` rangira rezultate temeljem frekvencije leksema koji se podudaraju u pitu i dokumentu.
- `ts_rank_cd` računaju rang prema radu Clarke, Cormack, and Tudhope's "Relevance Ranking for One to Three Term Queries" .
- Dokument duljine 1000 riječi u kojem se tražena riječ pojavljuje 10 puta nije jednako relevantan kao dokument duljine 100 riječi u kojem se tražena riječ pojavljuje također 10 puta.
- *normalization*  $\in \{0, 1, 2, 4, 8, 16, 32\}$  omogućuje reguliranje utjecaja duljine dokumenta na rang.
  - 0 (default) ne uzima u obzir duljinu dokumenta
  - 1 dijeli rang s  $1 + \log$  od duljine dokumenta
  - 2 dijeli rang s duljinom dokumenta
  - ...



# PostgreSQL: Rangiranje rezultata i stop riječi

```
SELECT *  
FROM to_tsvector('english', 'fat cat sat mat ate fat rat')
```

TSVector

'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4

```
SELECT * FROM  
ts_rank_cd (to_tsvector('english', 'a fat cat sat on a mat and ate a fat rat'),  
            to_tsQuery ('english', 'fat & cat & sat & mat & ate & rat & rat'))
```

ts\_rank\_cd

0.02

```
SELECT *  
FROM ts_rank_cd (to_tsvector('english', 'fat cat sat mat ate fat rat'),  
                to_tsQuery ('english', 'fat & cat & sat & mat & ate & fat & rat'))
```

ts\_rank\_cd

0.1

Za jednak upit, rang istog dokumenta je različit ovisno o tome jesu li stop riječi uklonjene ili nisu ➡ normalizirati i upit i dokument.

# PostgreSQL: Rangiranje rezultata - primjer

```
SELECT title
      , similarity ('Dencing Master', title)      sim
      , ts_rank(to_tsvector(title),
                to_tsquery ('Dencing & Master')) rank
  FROM movies
 WHERE similarity ('Dencing Master', title)      > 0.1
 ORDER BY rank desc, sim desc
```

title	sim	rank
The Dancing Masters	0.458333	1e-20
Dirty Dancing	0.217391	1e-20
The Man Who Loved Cat Dancing	0.184211	1e-20

Ako se u upitu *Dencing Master*'  
zamijeni s *'Dancing Master'*

title	sim	rank
The Dancing Masters	0.666667	0.0991032
Dirty Dancing	0.4	1e-20
The Man Who Loved Cat Dancing	0.285714	1e-20
Shall We Dance?	0.153846	1e-20

**Proučiti funkciju `ts_headline`. Korisna za izradu prvog projekta.**

```
ts_headline([ config regconfig, ] document text, query tsquery [, options text ]) returns text
```