

# Administracija Baze Podataka:

Lekcija 4

Dizajn Baze Podataka



# Sadržaj Lekcije

- Od Logičkog Modela do Fizičke Baze
- Dizajn Performanse Baze Podataka
- Denormalizacija
- Pogledi
- Data Definition Language
- Podrška za vremenske podatke

# Fizički Dizajn Baze Podataka

Za dobar fizički dizajn baze potrebno je:

- Detaljno poznavanje objekata baze podataka koje podržava DBMS i fizičke strukture i fajlovi koji su potrebni za podržavanje ovih objekata
- Poznavanje detalja koji se odnose na način na koji DBMS podržava indekse, referencijalni integritet, ograničenja, tipove podataka i ostale osobine koje utiču na funkcionisanje objekata baze.
- Detaljno poznavanje novih i zastarelih osobina za određene verzije ili release DMBS-a
- Poznavanje DMBS konfiguracionih parametara koji se koriste
- Data definition language (DDL) znanje da bi translirali fizički dizajn u objekte baze podataka

# Osnovne preporuke

- Izbeći korišćenje podrazumevanih podešavanja
  - Oni su retko najbolja podešavanja
  - Bolje je znati i eksplicitno navesti stanje podešavanja koji su vam potrebni, za svaki slučaj
- Sinhronizovati logičke i fizičke modele
  - Uvek mapirati promene jednog na drugi i obratno
- Performanse pre estetike
  - Značenje: poželjno potpuno normalizovano, ali odstupiti kada je potrebno da se postignu ciljevi performanse

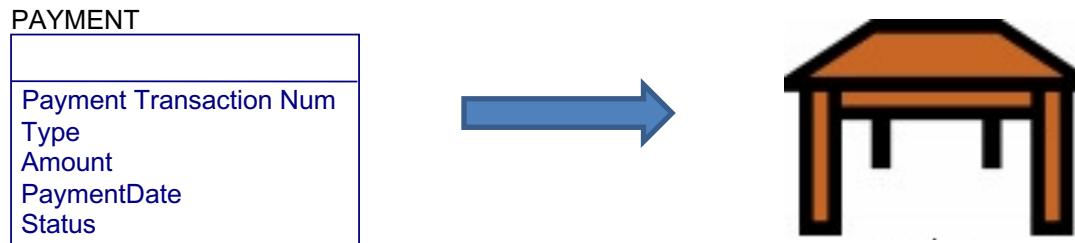
# Transformacija Logičkog u Fizički Model

- Transliranje Logičkog Modela u Fizički
  - Kreirati DDL skript
  - Entitete u Tabele, Atribute u Kolone, Veze i Ključeve u RI i Indekse, itd.
  - ...ali razlike **mogu da se dese**
- Kreirati strukture za smeštanje
  - Fajlove za podatke i indekse
  - Particionisanje
  - Klasterovanje
  - Smeštanje
  - Redosled kolona



# Transformacija Entiteta u Tabele

- Prvi korak:
  - Mapirati svaki entitet u logičkom modelu podataka u tabelu u bazi
- Stvari mogu da budu tako jednostavne, ili ne..
  - Denormalizacija?



# Transformacija Atributa u Kolone

- Atributi postaju kolone
- Transformacija Domena u Tipove Podataka
  - Komercijalni DBMS-ovi ne podržavaju domene
  - Tipovi Podataka i Dužina
    - Promenljive ili Fiksne Dužine
    - Izabratи mudro; uticaj na kvalitet podataka
  - Ograničenja (Check Constraints)
  - Null

# Tipovi Podataka

- CHAR / VARCHAR
- CLOB
- DBCLOB
- BLOB
- GRAPHIC / VARGRAPHIC
- DATE
- TIME
- DATETIME / TIMESTAMP
- XML
- BIGINT
- INTEGER
- SMALLINT
- MONEY
- BINARY
- DECIMAL
- FLOAT
  - REAL
  - DOUBLE

# Oracle Built-in Data Types

---

Following table summarizes Oracle built-in data types.

Types	Description	Size
VARCHAR2(size [BYTE   CHAR])	Variable-length character string.	From 1 byte to 4KB.
NVARCHAR2(size)	Variable-length Unicode character string having maximum length size characters.	Maximum size is determined by the national character set definition, with an upper limit of 4000 bytes. You must specify size for NVARCHAR2.
NUMBER [ (p [, s]) ]	Number having precision p and scale s. Range of p : From 1 to 38. Ranges of s : From -84 to 127. Both precision and scale are in decimal digits.	A NUMBER value requires from 1 to 22 bytes.
FLOAT [(p)]	A FLOAT value is represented internally as NUMBER. Range of p : From 1 to 126 binary digits.	A FLOAT value requires from 1 to 22 bytes.
LONG	Character data of variable length up to 2 gigabytes, used for backward compatibility.	$2^{31} - 1$ bytes

# SQL Server

## Number types:

Data type	Description	Storage
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	<p>Fixed precision and scale numbers.</p> <p>Allows numbers from <math>-10^{38} +1</math> to <math>10^{38} -1</math>.</p> <p>The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18.</p> <p>The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0</p>	5-17 bytes
numeric(p,s)	<p>Fixed precision and scale numbers.</p> <p>Allows numbers from <math>-10^{38} +1</math> to <math>10^{38} -1</math>.</p>	5-17 bytes

# Null

INVEN\_LOC\_TAB

WAREHSE_NO SMALLINT	BIN_NO SMALLINT	PROD_NO CHAR(5)	PROD_QTY INT
1	1	A100	???
1	2	A150	2000
1	3	B167	30
2	1	A100	775
2	2	D400	1585

## NULL:

Has no value

Is not = anything

Is not < anything

Is not > anything

Is not = NULL

**Is an UNKNOWN value**

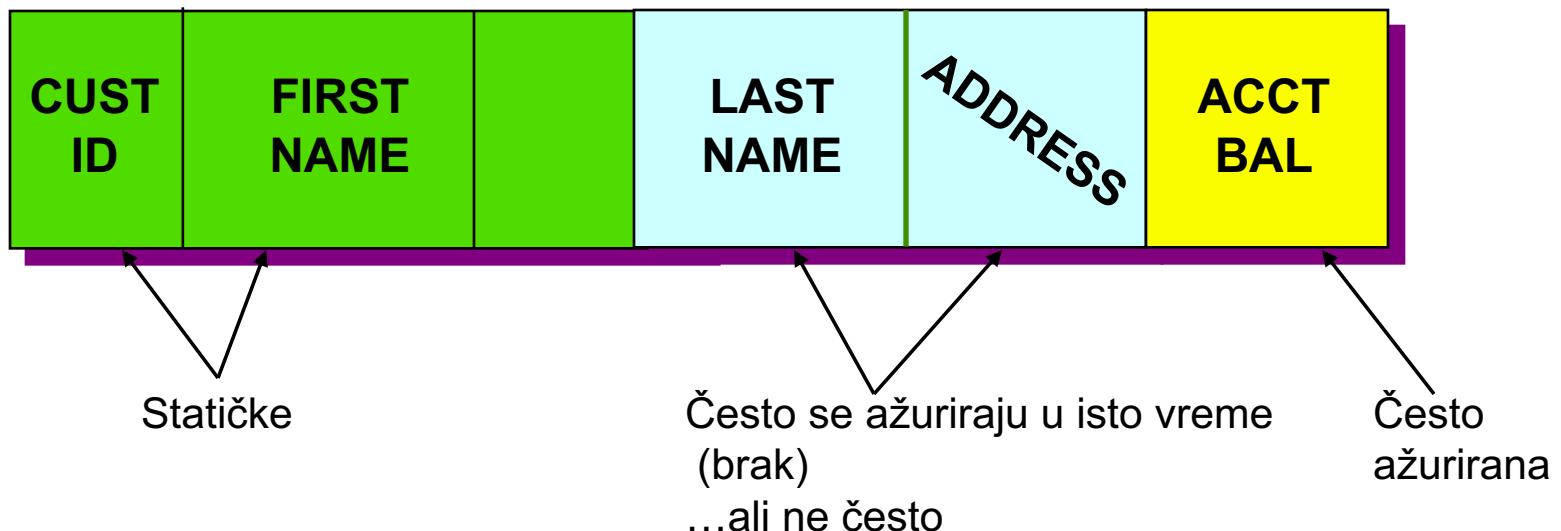
ATTRIBUTE QUALIFIER	DESCRIPTION
DEFAULT NULL	When no value is provided, assigns nulls to the Column
NOT NULL	The column must always contain a <u>value</u> , whether a default or explicitly provided

# Podrazumevane vrednosti

ATTRIBUTE QUALIFIER	DESCRIPTION	DEFAULT USED
(WITH) DEFAULT	When no value is provided, automatically assigns an appropriate default	Numeric Data: ZEROS  Character Data: SPACES  Variable Data: ZERO LENGTH  Chronological Data: CURRENT DATE CURRENT TIME CURRENT TIMESTAMP
(WITH) DEFAULT value		Constant USER CURRENT SQLID NULL

# Redosled Kolona

- Poređati kolone zbog logovanja. Na primer:
  - Kolone koje se često ne ažuriraju staviti na prvo mesto
  - Statičke kolone
  - Često ažurirane kolone poslednje
  - Obe koje se često zajedno modifikuju, staviti jednu pored druge



# Odrediti veličinu reda

**TABLE** Name: \_\_\_\_\_

## Data Lengths:

**Data Length:**  
**Row Overhead:**  
**Physical Row Length:**

INTEGER	4	FLOAT (DBL PRECISION)	8
DECIMAL PACKED FORMAT		DATE	4
SMALLINT	2	TIME	3
FLOAT (REAL)	4	TIMESTAMP	10

# Veze i Ključevi

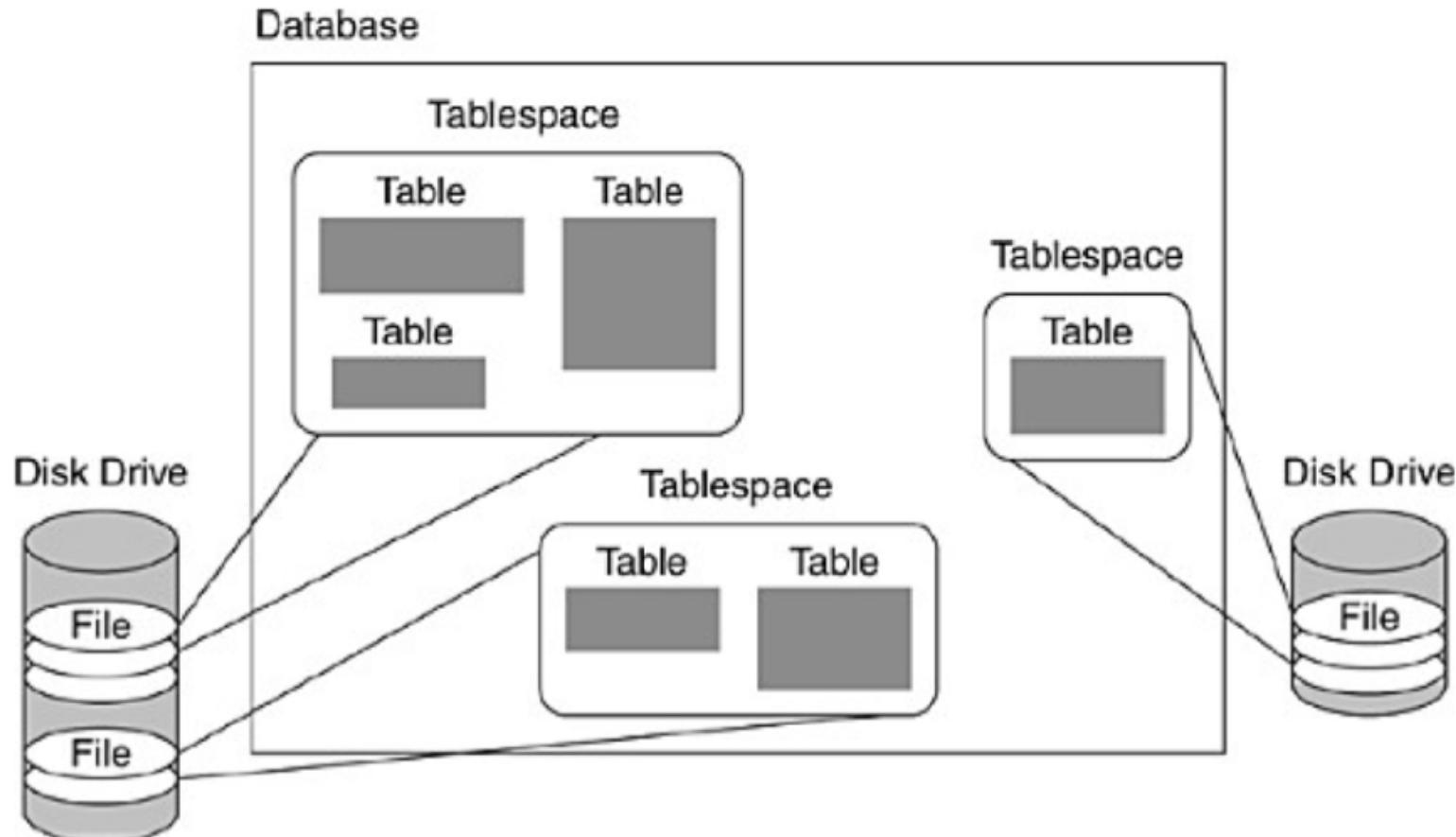
- Koristiti primarni ključ koji je dodeljen u logičkom modelu za fizički ključ
- Druga razmatranja:
  - Dužina ključa
  - Surogat ključ
  - ROWID / SEQUENCE / Identitet
- Izgraditi referencijalna ograničenja za sve veze
  - Strani ključevi

# Izgradnja Fizičkih Struktura

- Table Spaces
- DBSpaces
- Data Spaces
- Filegroups

```
CREATE TABLE [dbo].acr_myTable(
    [Id] [bigint] NOT NULL,
    [label] [nvarchar](max) NOT NULL,
)on ArchiveFileGroup
```

# Mapiranje fajlova u strukture baze



# SQL Server Filegroups

- Filegroups (file grupa) grapišu fajlove koji sadrže podatke jedne baze
- Grupisanjem se pojednostavljuju neki administratorski zadaci (bekap i oporavak)
- Jedna baza može sadržati više file grupa
- Svaka baza se kreira u PRIMARY file grupi
- Kasnije se mogu dodati i druge file grupe bazi
- Ako je baza velika i aktivna, više fajlova se koristi da bi se poboljšale performanse

## Properties



## [Tbl] Production.SpecialtyProducts



## ▲ (Identity)

(Name)	SpecialtyProducts
Database Name	AdventureWorks2012
Description	
Schema	Production
Server Name	win7vm\sqlsrv2012

## ▲ Table Designer

Identity Column	
Indexable	Yes

Lock Escalation	Table
-----------------	-------

## ▲ Regular Data Space Specification

(Data Space Type)	Filegroup
-------------------	-----------

Filegroup or Partition Scheme	PRIMARY
-------------------------------	---------

Partition Column List	
-----------------------	--

Replicated	No
------------	----

Row G UID Column	
------------------	--

Text/Image Filegroup	PRIMARY
----------------------	---------

# Planiranje smeštanja

- Započeti procenom koliko redova je potrebno.
- Izračunati veličinu reda
- Odrediti broj redova po bloku/stranici
- Pomnožiti #red/strana sa veličinom strane
- Ovo daje veličinu objekta
- Osim slobodnog prostora...

# Kreiranje tabele - Oracle

```
CREATE TABLE hr.admin_emp (
    empno      NUMBER(5) PRIMARY KEY,
    ename      VARCHAR2(15) NOT NULL,
    ssn        NUMBER(9) ENCRYPT,
    job        VARCHAR2(10),
    mgr        NUMBER(5),
    hiredate   DATE DEFAULT (sysdate),
    photo      BLOB,
    sal         NUMBER(7,2),
    hrly_rate  NUMBER(7,2) GENERATED ALWAYS AS (sal/2080),
    comm       NUMBER(7,2),
    deptno     NUMBER(3) NOT NULL
        CONSTRAINT admin_dept_fkey REFERENCES hr.departments
        (department_id))

TABLESPACE admin_tbs
STORAGE ( INITIAL 50K);

COMMENT ON TABLE hr.admin_emp IS 'Enhanced employee table';
```

# Slobodan Prostor

Freespace is considered  
only at LOAD or REORG time

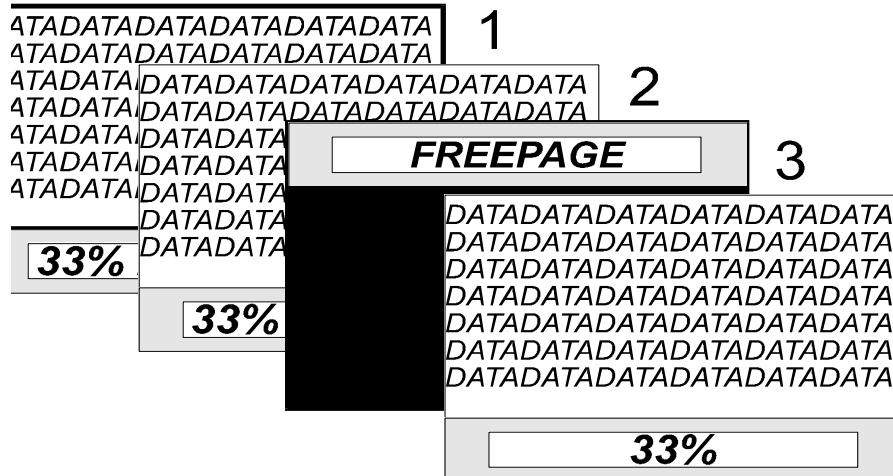
The freespace options:

**PCTFREE**  
**FREEPAGE**

For Example:

**PCTFREE 33**

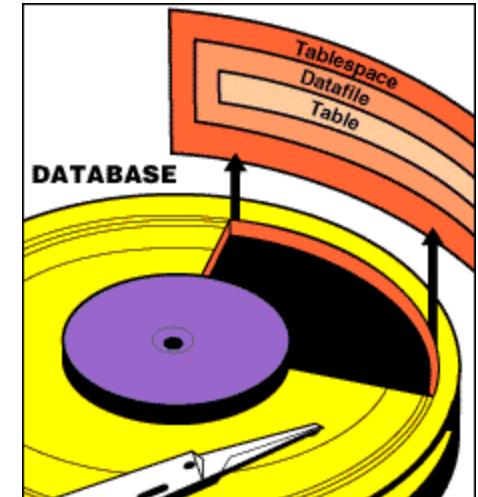
**FREEPAGE 2**



```
CREATE TABLE ORDERS
(
  ORDER_ID NUMBER,
  ITEM_CODE VARCHAR2(100),
  EXEC_NAME VARCHAR2(100))
PCTFREE 20
PCTUSED 50
```

# Vrste fajlova

- Podaci / Indeksi
  - Oba zahtevaju smeštanje
- Raw Fajlovi
  - Mogu da se koriste da se preskoči O/S
- Solid State Devices (SSD)
  - Za objekte kritične performanse
- Kompresija?

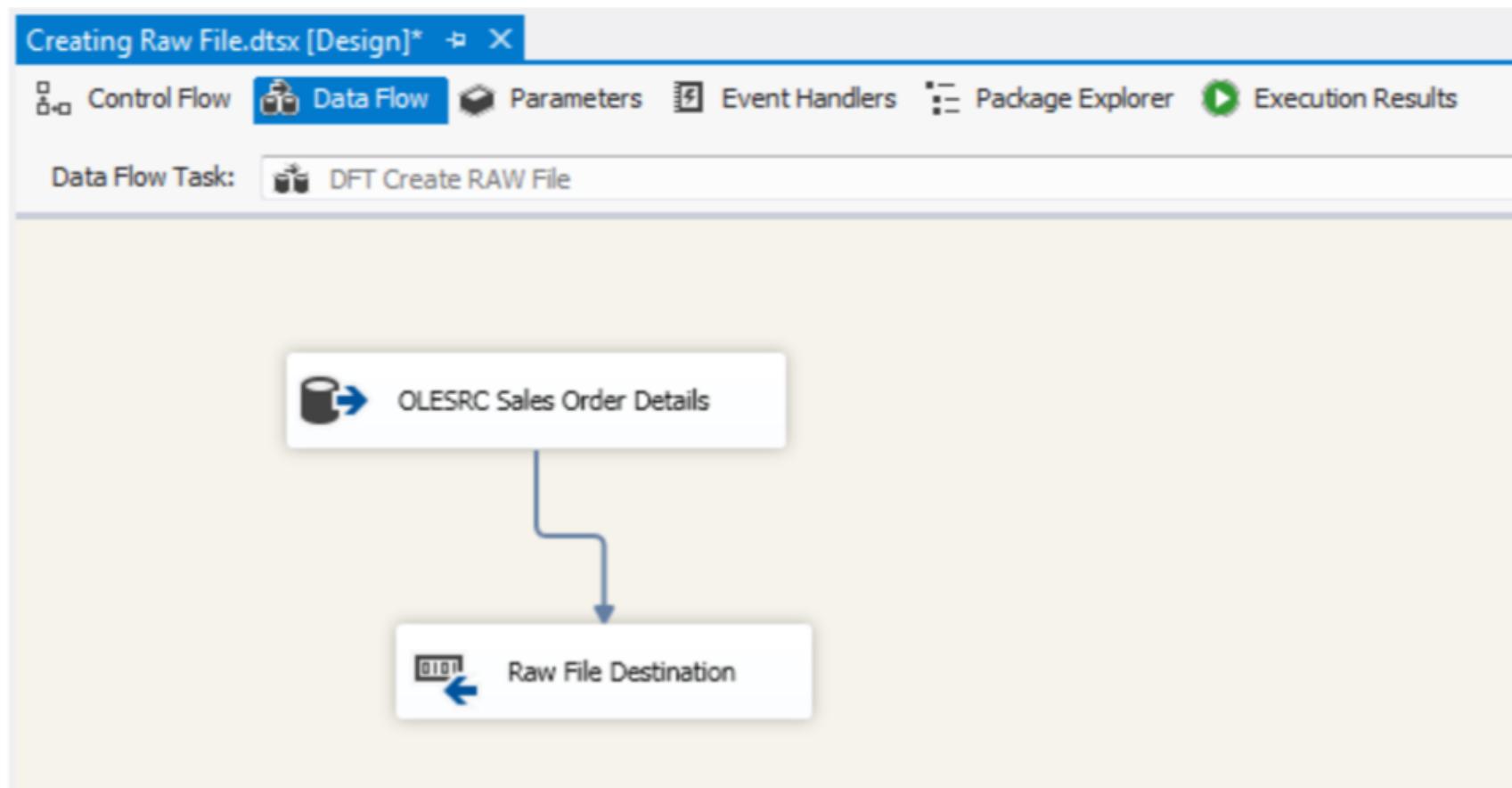


# SQL Server Integration Service

- Iz dokumentacije (važi od verzije SQL Server 2016):

“The Raw File source reads raw data from a file. Because the representation of the data is native to the source, the data requires no translation and almost no parsing. This means that the Raw File source can read data more quickly than other sources such as the Flat File and the OLE DB sources.”

# SQL Server - raw files



# Dizajn Performanse Baze Podataka

- Performanse baze kada aplikacija zahteva pristup podacima?
- Pristup disku je sporiji nego pristup memoriji!
- Kako pristupiti podacima efikasno?
- Dizajniranje Indeksa
  - Particionisanje
  - Klastering
- Hashing
- Interleaving Podataka

# Dizajniranje Indeksa

## *Prednosti Indeksa*

### *Optimizovati pristup podacima:*

- DBMS određuje da li koristiti indeks
- DBMS održava sve indekse
- Prolazak kroz tabelu se može izbeći koristeći indekse
- Preporučuje se za kolone stranog ključa da bi se ubrzao pristup RI
- Indeksi mogu da minimizuju sortiranje
- Može da postoji više indeksa po tabeli da bi podržali način obrade podataka
- Kreirati indekse na osnovu obima posla (ne tabela)
- Ako su sve kolone u indeksima, dobijate index-only pristup (IXO)

### *Garantovati jedinstvenost*

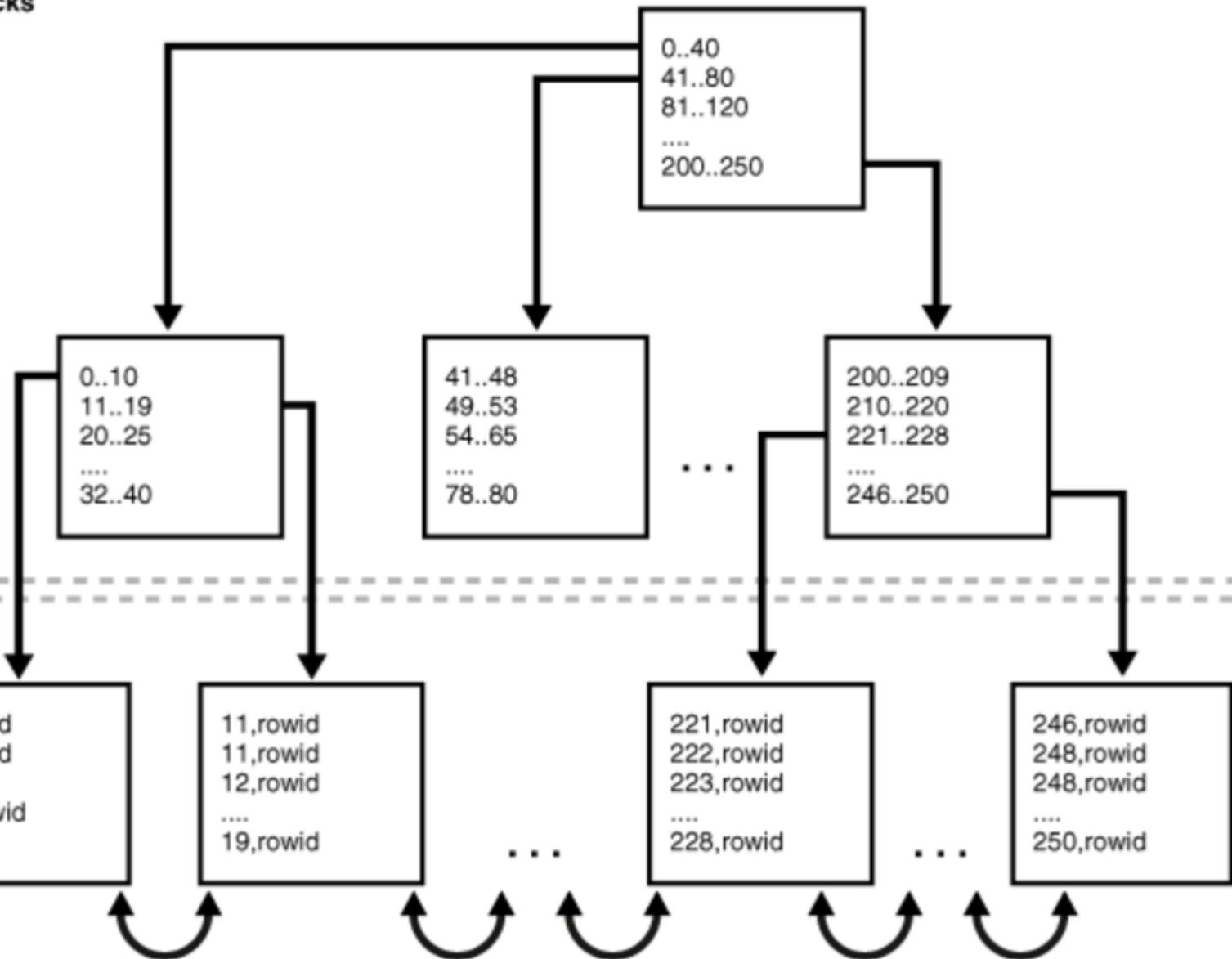
- Može da se koristi da bi se osigurala jedinstvenost vrednosti kolona
- Zahtevano na kolonama primarnog ključa kao deo implementacije referencijalnog integriteta

### *Implementiranje klasteringa:*

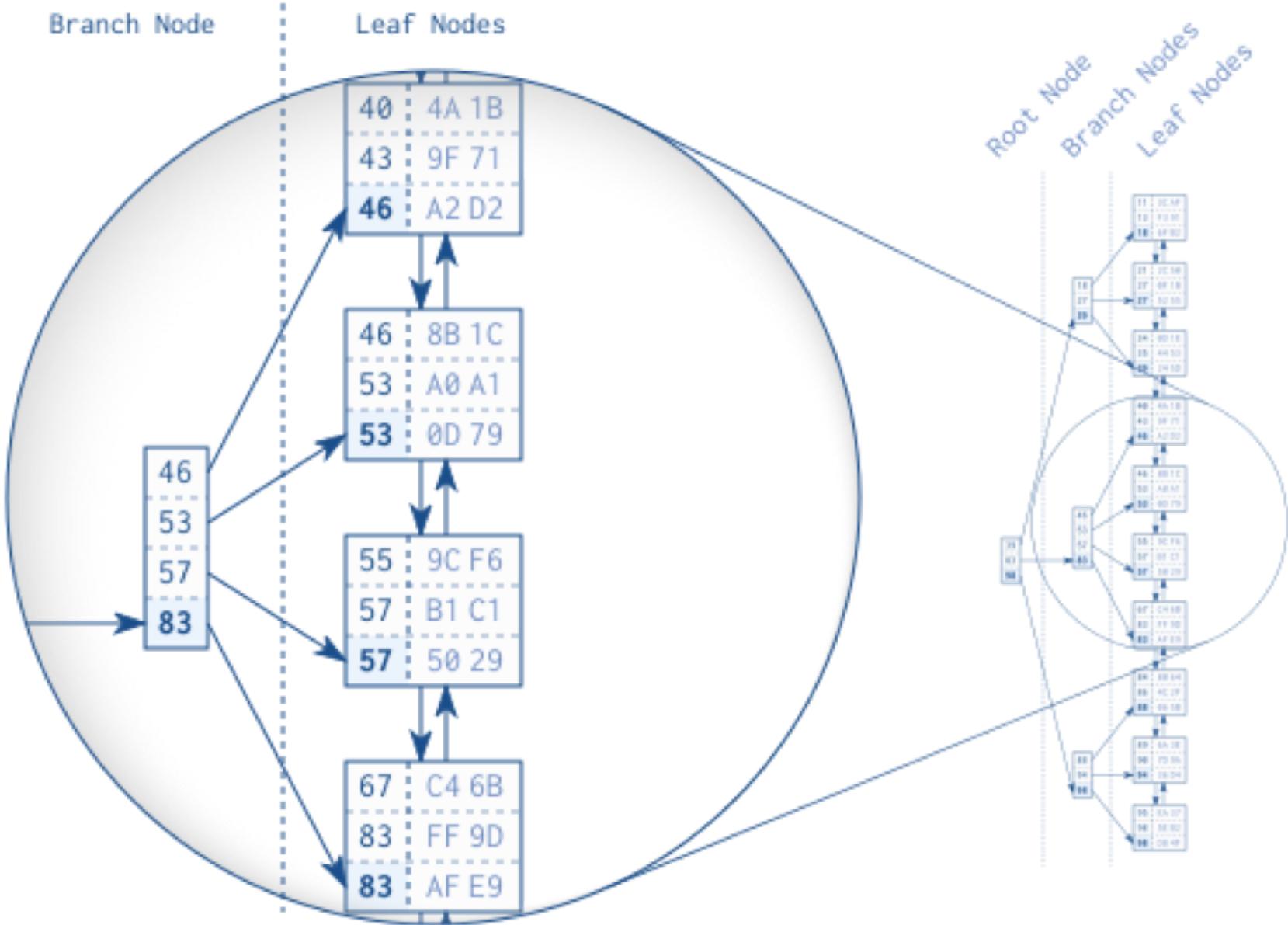
- Indeksi mogu da se koriste za klasterovanje; tj., održavanje redova fizički na disku u sekvenci vrednosti kolona u indexu

# B-Tree Index

## Branch Blocks

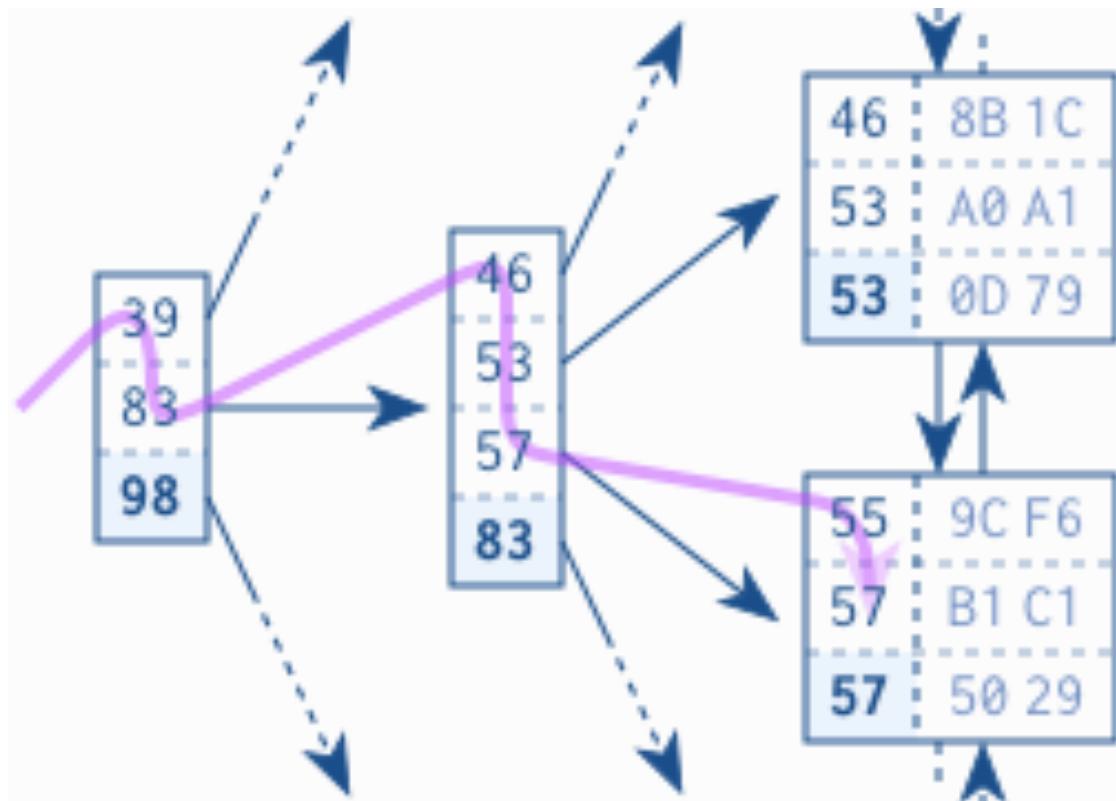


# B-Tree Index



# Prolazak kroz B-Tree

Traži se vrednost indexa 57.



# Bitmap Index

Identifier	Gender	Bitmap
1	Female	0110000010
2	Male	1000011101
3	Unknown	0001100000

```
CREATE BITMAP INDEX emp_bitmap_idx ON big_emp(sex);
```

```
CREATE BITMAP INDEX emp_dept_loc ON emp(dept.loc)
  FROM emp, dept
 WHERE emp.deptno = dept.deptno
 TABLESPACE index_ts1;
```

# SQL Server: indeksi

- Indeksi služe da se ubrza pronalaženje podataka, umesto prolaska kroz celu tabelu
- Indeksi se kreiraju nad jednom ili više kolona

```
CREATE INDEX ix_Ime  
ON Zaposleni (ime)
```

```
CREATE INDEX ix_Ime  
ON Zaposleni (ime, prezime)
```

# SQL Server: indeksi

- Indeksi u SQL serveru ne mogu da se kreiraju nad kolonama sledećih tipova:

**ntext, text, varchar(max), nvarchar(max), varbinary(max), xml, ili image**

- Postoje dve vrste indeksa
  - Klasterovani indeks
  - Ne-klasterovan indeks

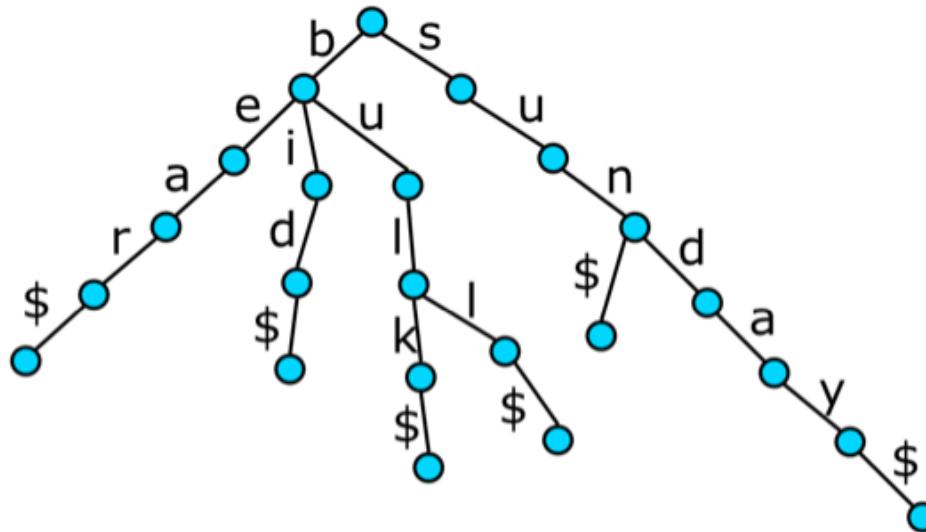
# SQL Server: neklasterovan indeks

- Postoje dva fajla: jedan je za indekse, drugi za podatke
- Može postojati više od jednog neklasterovanog indeksa po tabeli
- Primer kreiranja neklasterovanog indeksa

```
CREATE NONCLUSTERED INDEX IX_tblStudent_Name  
ON student(name ASC)
```

# SQL Server: neklasterovan indeks

- Indeksovanje stringa: implementacija zavisi od samog DBMSa
- Jeden primer je drvo prefiksa



Set of strings: {**bear**, **bid**, **bulk**, **bull**, **sun**, **sunday**}

# SQL Server: neklasterovan indeks

Student Table Data:						IX_tblStudent_Name Index Data	
id	name	gender	DOB	total_score	City	name	Row Address
1	Jolly	Female	1989-06-12 00:00:00.000	500	London	Alan	Row Address
2	Jon	Male	1974-02-02 00:00:00.000	545	Manchester	Elis	Row Address
3	Sara	Female	1988-03-07 00:00:00.000	600	Leeds	Jolly	Row Address
4	Laura	Female	1981-12-22 00:00:00.000	400	Liverpool	Jon	Row Address
5	Alan	Male	1993-07-29 00:00:00.000	500	London	Joseph	Row Address
6	Kate	Female	1985-01-03 00:00:00.000	500	Liverpool	Kate	Row Address
7	Joseph	Male	1982-04-09 00:00:00.000	643	London	Laura	Row Address
8	Mice	Male	1974-08-16 00:00:00.000	543	Liverpool	Mice	Row Address
9	Wise	Male	1987-11-11 00:00:00.000	499	Manchester	Sara	Row Address
10	Elis	Female	1990-10-28 00:00:00.000	400	Leeds	Wise	Row Address

Primeri preuzeti sa: <https://www.sqlshack.com/what-is-the-difference-between-clustered-and-non-clustered-indexes-in-sql-server/>

# SQL Server: neklasterovan indeks

- Indeks sadrži RID reda u kome se nalazi indeksovani podatak
- Ukoliko tabela nema ni jedan klasterovan indeks, kaže se da je u heap-u

# SQL Server: klasterovan indeks

- Samo jedan klasterovan indeks može da se kreira po tabeli
- Kaže se da je tabela u klasterovanom indeksu (umesto u heapu)
- Ne postoji odvojen fajl za indeks, već je tabela uređena

# SQL Server: klasterovan indeks

```
INSERT INTO student  
  
VALUES  
(6, 'Kate', 'Female', '03-JAN-1985', 500, 'Liverpool'),  
(2, 'Jon', 'Male', '02-FEB-1974', 545, 'Manchester'),  
(9, 'Wise', 'Male', '11-NOV-1987', 499, 'Manchester'),  
(3, 'Sara', 'Female', '07-MAR-1988', 600, 'Leeds'),  
(1, 'Jolly', 'Female', '12-JUN-1989', 500, 'London'),  
(4, 'Laura', 'Female', '22-DEC-1981', 400, 'Liverpool'),  
(7, 'Joseph', 'Male', '09-APR-1982', 643, 'London'),  
(5, 'Alan', 'Male', '29-JUL-1993', 500, 'London'),  
(8, 'Mice', 'Male', '16-AUG-1974', 543, 'Liverpool'),  
(10, 'Elis', 'Female', '28-OCT-1990', 400, 'Leeds');
```

Redovi su smešteni uredjeni po Id-ju zato što je kreiran klasterovan indeks nad kolonom primarnog ključa

Ubacili smo redove po nekom redosledu.

	id	name	gender	DOB	total_score	city
	1	Jolly	Female	1989-06-12 00:00:00.000	500	London
	2	Jon	Male	1974-02-02 00:00:00.000	545	Manchester
	3	Sara	Female	1988-03-07 00:00:00.000	600	Leeds
	4	Laura	Female	1981-12-22 00:00:00.000	400	Liverpool
	5	Alan	Male	1993-07-29 00:00:00.000	500	London
	6	Kate	Female	1985-01-03 00:00:00.000	500	Liverpool
	7	Joseph	Male	1982-04-09 00:00:00.000	643	London
	8	Mice	Male	1974-08-16 00:00:00.000	543	Liverpool
	9	Wise	Male	1987-11-11 00:00:00.000	499	Manchester
	10	Elis	Female	1990-10-28 00:00:00.000	400	Leeds

# SQL Server: klasterovan indeks

```
CREATE CLUSTERED INDEX IX_tblStudent_Gender_Score  
ON student(gender ASC, total_score DESC)
```

Uredjena tabela je ispod

id	name	gender	DOB	total_score	city
3	Sara	Female	1988-03-07 00:00:00.000	600	Leeds
1	Jolly	Female	1989-06-12 00:00:00.000	500	London
6	Kate	Female	1985-01-03 00:00:00.000	500	Liverpool
4	Laura	Female	1981-12-22 00:00:00.000	400	Liverpool
10	Elis	Female	1990-10-28 00:00:00.000	400	Leeds
7	Joseph	Male	1982-04-09 00:00:00.000	643	London
2	Jon	Male	1974-02-02 00:00:00.000	545	Manchester

# Fajlovi za smeštanje

- U SQL Serveru su podaci iz tabele smešteni u stranice podataka od 8KB
- Zeleni obeleženi su stranice podataka sa indeksima, sivo su stvarni podaci

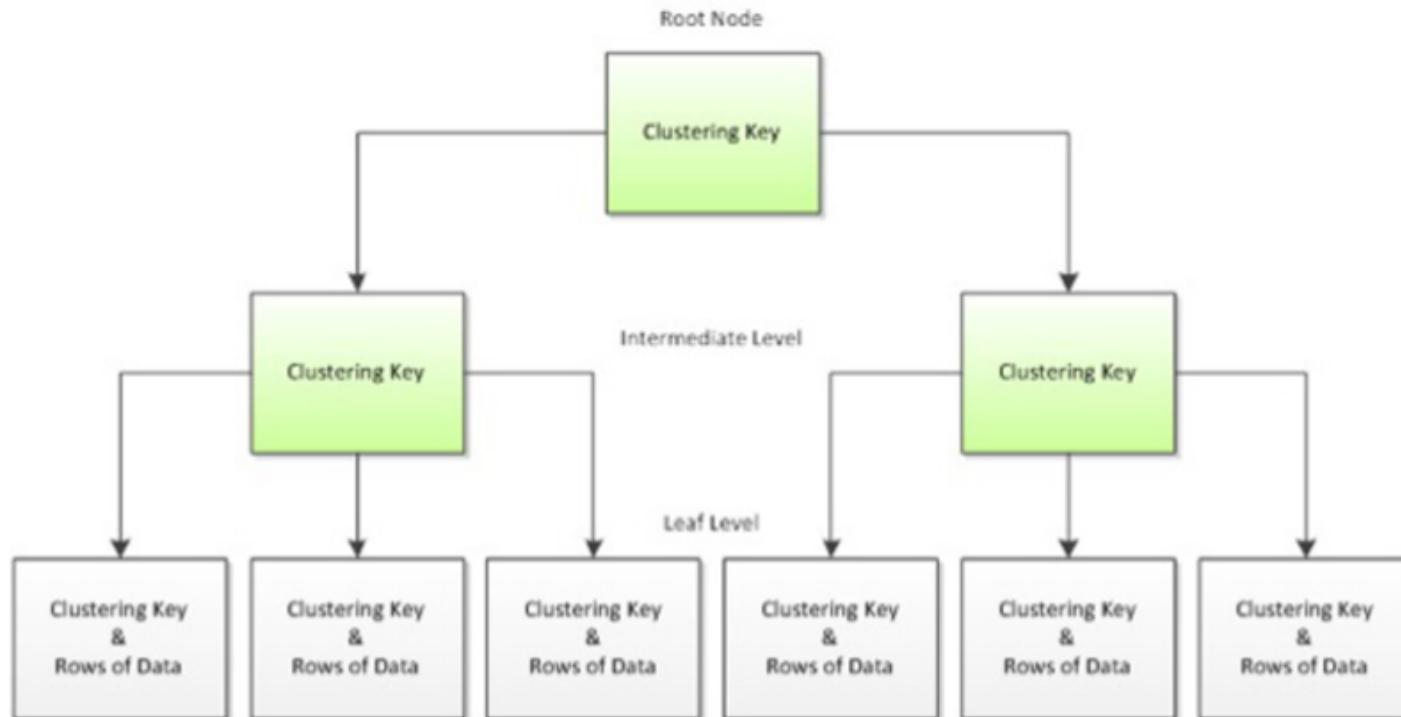


Figure 1: The b-tree structure of a clustered index

# Primer – klasterovan index

```
SELECT CustomerID , OrderDate ,
       SalesOrderNumber
  FROM Sales.SalesOrderHeader
 WHERE SalesOrderID = 44242 ;
```

Root Node	<b>SalesOrderID</b>	<b>PageID</b>	Intermediate	<b>SalesOrderID</b>	<b>PageID</b>
			level		
	NULL	750	(Page 750)		
	59392	751		44150	814
				44197	815
				44244	816
				44290	817
				44333	818

# Primer – klasterovan index

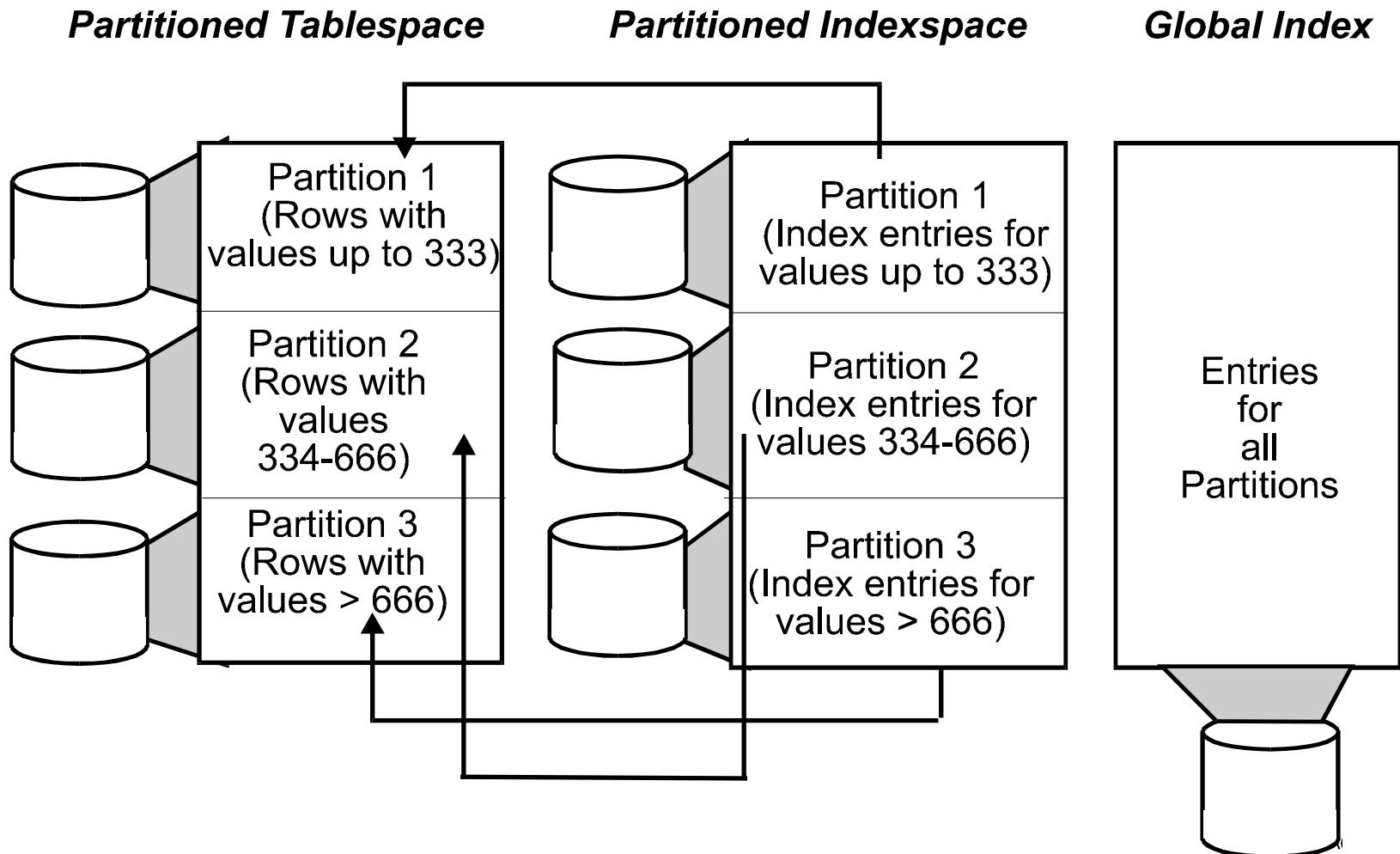
```
SELECT CustomerID , OrderDate ,
       SalesOrderNumber
  FROM Sales.SalesOrderHeader
 WHERE SalesOrderID = 44242 ;
```

Leaf level	SalesOrderID	OrderDate	SalesOrderNumber	AccountNumber	CustomerID
(Page 815)					
	44240	9/23/2005	S044240	10-4030-013580	13580
	44241	9/23/2005	S044241	10-4030-028155	28155
	44242	9/23/2005	S044242	10-4030-028163	28163

# Drugi tipovi indeksa

- Reverse Key Indeks
  - b-tree index gde je redosled bajtova svake indeksirane kolone obrnut; (Craig-giarC)
- Podeljen Indeks
  - b-tree index koji specificira kako podeliti index (i možda tabelu) u odvojene delove, ili particije; za poboljšanje performanse i dostupnosti
- Uređen Index (rastući ili opadajući)

# Partitioning



# Primeri particionisanja tabele

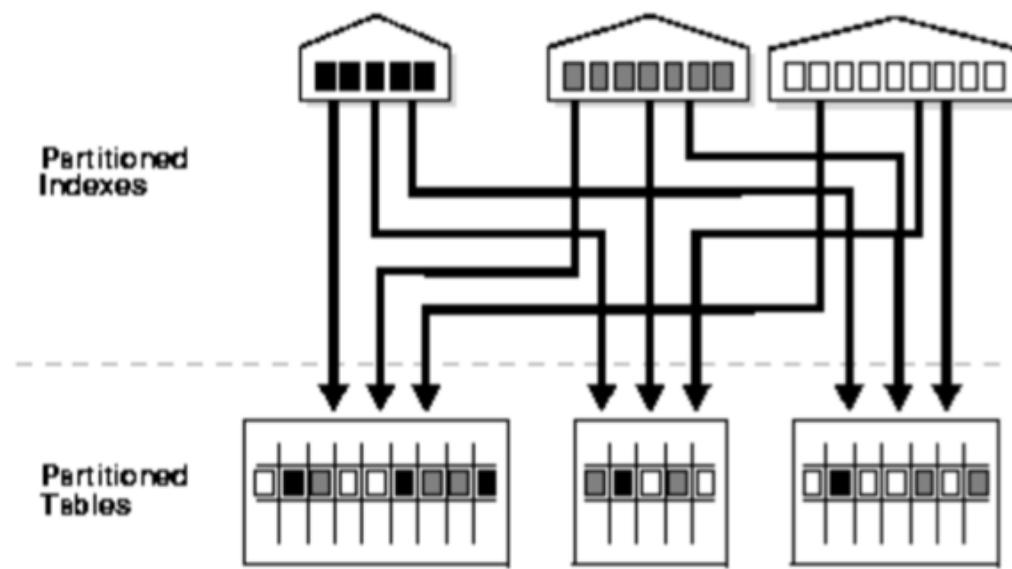
```
CREATE TABLE sales_range
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_amount  NUMBER(10),
sales_date    DATE)
PARTITION BY RANGE(sales_date)
(
PARTITION sales_jan2000 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),
PARTITION sales_feb2000 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),
PARTITION sales_mar2000 VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY')),
PARTITION sales_apr2000 VALUES LESS THAN(TO_DATE('05/01/2000','DD/MM/YYYY'))
);
```

```
CREATE TABLE sales_list
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_state   VARCHAR2(20),
sales_amount  NUMBER(10),
sales_date    DATE)
PARTITION BY LIST(sales_state)
(
PARTITION sales_west VALUES('California', 'Hawaii'),
PARTITION sales_east VALUES ('New York', 'Virginia', 'Florida'),
PARTITION sales_central VALUES('Texas', 'Illinois')
PARTITION sales_other VALUES(DEFAULT)
);
```

# Primeri particionisanja indeksa

```
CREATE TABLE employees
(employee_id NUMBER(4) NOT NULL,
 last_name VARCHAR2(10),
 department_id NUMBER(2))
PARTITION BY RANGE (department_id)
(PARTITION employees_part1 VALUES LESS THAN (11) TABLESPACE part1,
 PARTITION employees_part2 VALUES LESS THAN (21) TABLESPACE part2,
 PARTITION employees_part3 VALUES LESS THAN (31) TABLESPACE part3);
```

```
CREATE INDEX employees_global_part_idx ON employees(employee_id)
GLOBAL PARTITION BY RANGE(employee_id)
(PARTITION p1 VALUES LESS THAN(5000),
 PARTITION p2 VALUES LESS THAN(MAXVALUE));
```



# Clustering

- Definiše način na koji se podaci smeštaju fizički
- Termin se odnosi na smeštanje redova po definisanom redosledu na disk
- Kroz klastering se podaci kojima se pristupa zajedno, obično smeštaju zajedno
- Smeštanjem zajedno podataka se performanse optimizuju kada se podacima pristupa sekvencijalno, zato što je zahtevano manje I/O pristupa

# Clustering

**NON-UNIQUE  
CLUSTERING INDEX  
ORDR\_DATE**

SALES\_ORDER\_TAB

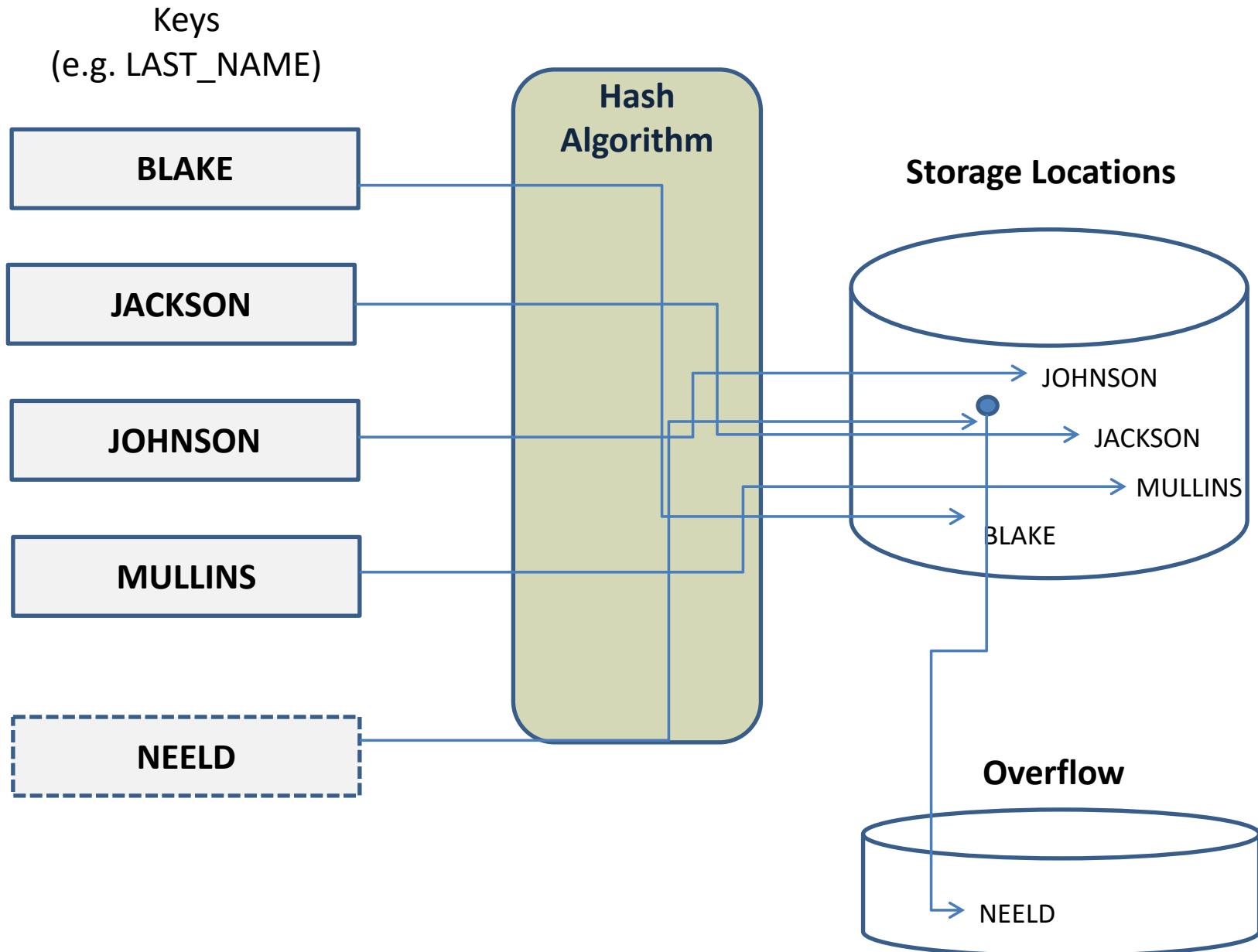
SALES_ORDER_NO INT	ORDR_DATE DATE	CUST_NO SMALL INT	SALES_HIST_CUST_NO SMALL INT	ORDR_AMT DEC(9,2)
1	1997-09-15		2	1923.45
3	1997-09-23		1	2407.53
4	1997-09-23		10	57613.89
5	1997-09-29	3	5	67000.00
6	1997-10-01	2		42345.88
7				122345.61
8				23007.34
9				9823.55
10				223019.27
11				78780.99

ORDR_NO	DATE
000000125	19970923
000004946	19970923
000000013	19971002
000000615	19971002
000008885	19971002
000074155	19971004

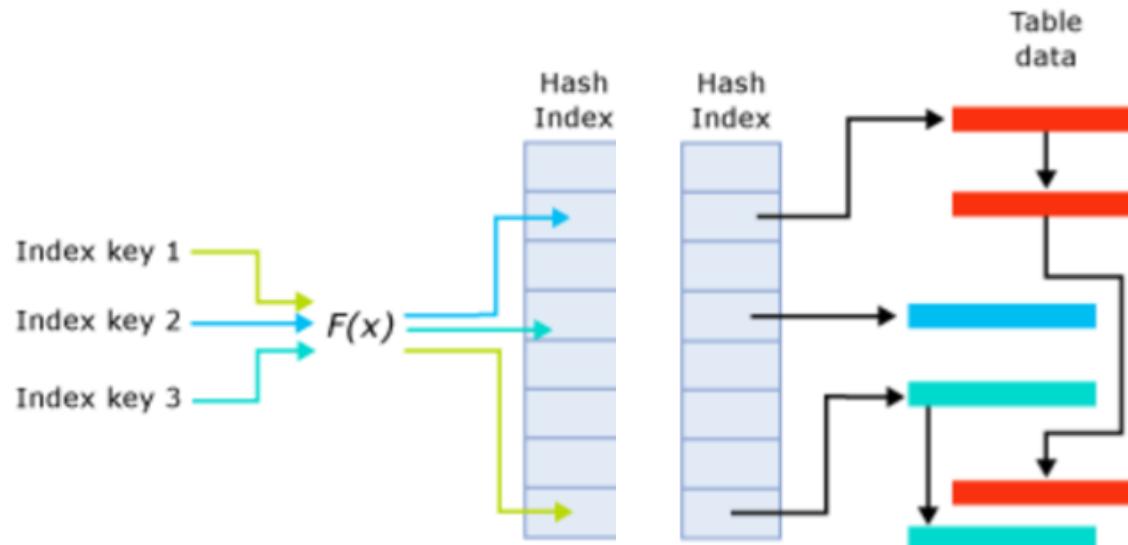
# Kada klasterovati tabele

- Veliki broj upita dohvata opsege podataka neke kolone
- Strani ključ je dobar kandidat
- Primarni nije, zato što često ima random vrednosti
- Kada se podaci često sortiraju (ORDER BY, GROUP BY, UNION, SELECT DISTINCT)
- Paziti na učestalost modifikacije
- Insert i update utiču da podaci postanu neklasterovani

# Hashing



# Hashing



- Indeksi su ulazna tačka za pristup
- Hash indeks ima kolekciju "kantica" organizovanih u niz
- Hash funkcija  $f(x)$  mapira indeksa u kofice u hash indeksu
- Svaka kofa u nizu ukazuje na prvi red u hash kofama
- Svaki red u kofi ukazuje na sledeći tj. uvezani su u listu

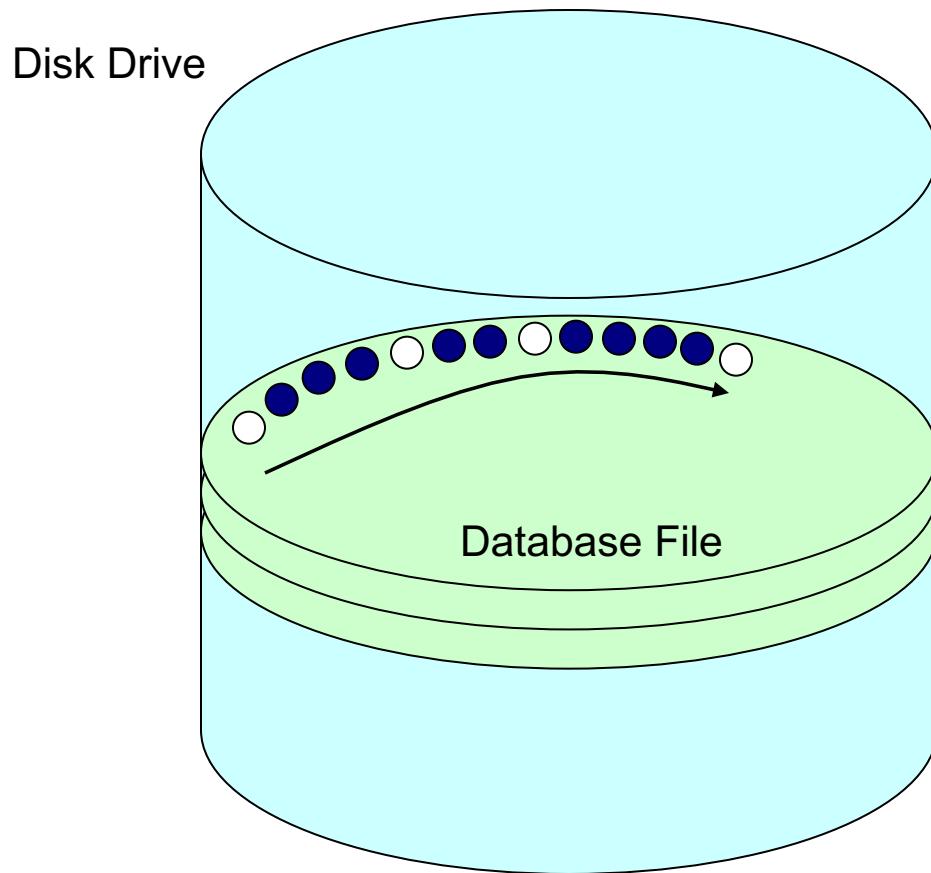
# Hashing

```
CREATE TABLE SupportIncidentRating_Hash
(
    SupportIncidentRatingId      int          not null      identity(1,1)
    PRIMARY KEY NONCLUSTERED,

    RatingLevel                  int          not null,
    SupportEngineerName          nvarchar(16)  not null,
    Description                  nvarchar(64)   null,
    INDEX ix_hash_SupportEngineerName
        HASH (SupportEngineerName) WITH (BUCKET_COUNT = 100000)
)
WITH (
    MEMORY_OPTIMIZED = ON,
    DURABILITY = SCHEMA_ONLY);
go
```

- Najčešće je veličina backeta izmedju 1 i 2 puta broj različitih vrednosti indeksa

# Preplitanje Podataka



Legend

- Table 1
- Table 2

# Denormalizacija

- **Prejoined Tables** – kada su troškovi spajanja previsoki
- **Tabele izveštaja-** za posebne kritične izveštaje (npr. CEO)
- **Mirror Tabele** – kada dve vrste okruženja zahtevaju konkurentni pristup istim podacima (**OLTP (Online transaction processing) vs DSS (Decision-support System)**)
- **Split Tabele** – Kada različite grupe/aplikacije koriste različite delove iste tabele
  - Podele kolona ili redova.
- **Kombinovane Tabele** – za eliminaciju veze 1-1
- **Redundantni Podaci** – za redukovanje broja spajanja neke kolone (e.g. definitional, CA to California)

# Kada Denormalizovati

- Jedini razlog za denormalizaciju:
  - Postići optimalnu **performansu!**
- Ukoliko dizajn baze podataka postiže zadovoljavajuće performanse kada je potpuno normalizovan, onda nema potrebe denormalizovati
- Uvek bi trebalo razmatrati sledeće probleme pre denormalizacije.
  - Da li sistem postiže zadovoljavajuće performanse bez denormalizacije?
  - Da li će performanse sistema posle denormalizacije i dalje biti neprihvatljive?
  - Da li će sistem biti manje pouzdan zbog denormalizacije?

# Administracija Denormalizacije

Odluka da se denormalizuje ne treba da se shvati olako, zato što može izazvati probleme integriteta i uključiti puno administracije.

Dodatni zadaci administracije uključuju:

- Dokumentovanje svake odluke denormalizacije
- Osigurati da svi podaci ostanu validni i tačni
- Zakazivanje migracije podataka
- Informisati krajnje korisnike o stanju tabela
- Periodična analizira baze podataka da bi odlučili da li je denormalizacija još potrebna

# Normalizacija vs. Denormalizacija

Cilj!



## ***Normalized Tables:***

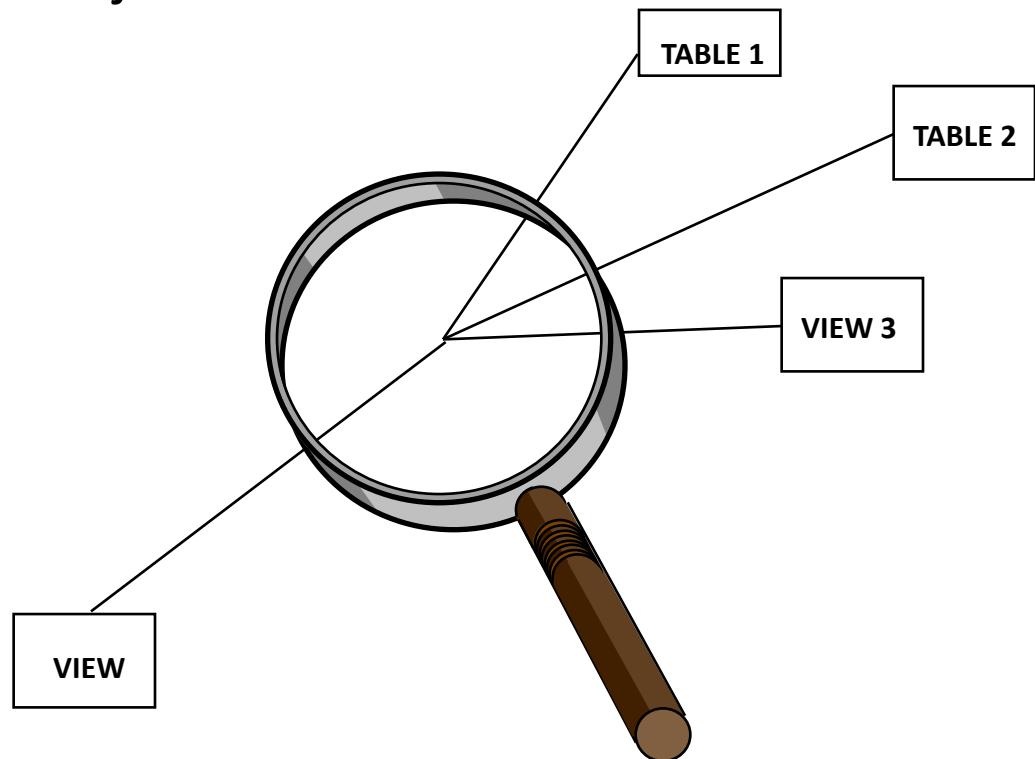
- ◆ More Tables
- ◆ Fewer Columns per Table
- ◆ Fewer Rows per Table
- ◆ Less Redundancy
- ◆ More Joins
- ◆ Update Efficient

## ***Denormalized Tables:***

- ◆ Fewer Tables
- ◆ More Columns per Table
- ◆ More Rows per Table
- ◆ Greater Redundancy
- ◆ Fewer Joins
- ◆ Read Efficient

# Pogledi (Views)

- Pretvaranje SELECT upita u “tabelu”
- Pogled - logička tabela
- Takođe aspekt fizičkog dizajna baze



# Pogledi

Course Table = TCRSE

An actual table:

CRSE_NUM	CRSE_CLASS	CRSE_INST	CRSE_NAME	CRSE_DATE	CRSE_TIME	CRSE_DAY	CRSE_TYPE
INT	CHAR(6)	INT	CHAR(20)	DATE	TIME	CHAR(2)	CHAR(1)
26	DB2004	200	SYSTEMS ADMIN	1990-11-24	10.00.00	TU	A
27	DB2001	--	INTRO TO DB2	1990-11-30	09.00.00	MO	A
28	DB2002	300	SQL PROGRAMMING	1990-11-30	09.00.00	MO	B
29	PHY001	500	RELATIVITY	1990-10-02	09.00.00	WD	A
30	DB2003	--	DATABASE DESIGN	1990-12-07	09.00.00	MO	B
31	DB2001	800	INTRO TO DB2	1990-12-07	09.00.00	MO	A
32	LIT001	600	ENTREPRENEUR	1990-12-09	10.00.00	WD	A
33	DB2001	100	INTRO TO DB2	1990-12-09	09.00.00	WD	B

Course View = VCRSE

CRSE_NUM	CRSE_NAME	CRSE_DATE	CRSE_TIME	CRSE_DAY	CRSE_TYPE
INT	CHAR(20)	DATE	TIME	CHAR(2)	CHAR(1)
26	SYSTEMS ADMIN	1990-11-24	10.00.00	TU	A
27	INTRO TO DB2	1990-11-30	09.00.00	MO	A
28	SQL PROGRAMMING	1990-11-30	09.00.00	MO	B
29	RELATIVITY	1990-10-02	09.00.00	WD	A
30	DATABASE DESIGN	1990-12-07	09.00.00	MO	B
31	INTRO TO DB2	1990-12-07	09.00.00	MO	A
32	ENTREPRENEUR	1990-12-09	10.00.00	WD	A
	INTRO TO DB2	1990-12-09	09.00.00	MO	B

A logical  
view of that  
table

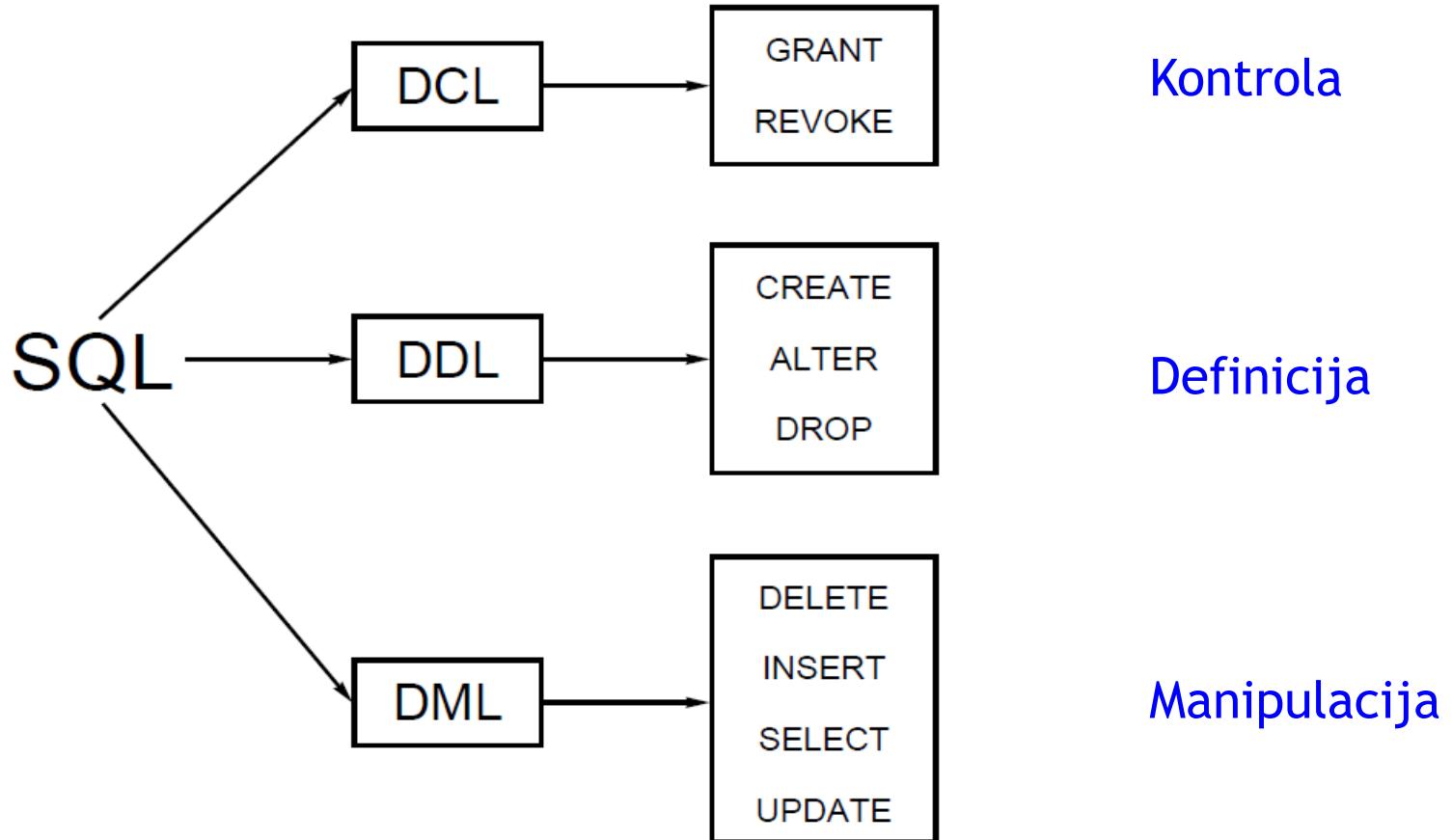
Course list  
'View'

# Pravila korišćenja Pogleda

- Sigurnost – nivo kolone i reda
- Pristup – efikasne putanje pristupa
- Izvodjenje podataka – staviti kalkulacije u pogled
- Zamaskirati kompleksnost – sakriti komplikovan SQL od korisnika
- Promeniti ime tabele
- Promeniti ime kolone – tabele sa boljim imenima kolona (bolje od korišćenja AS)
- Sinhronizovati sve poglede sa osnovnim tabelama...

**NE KORISTITI JEDAN POGLED PO TABELI!**

# Vrste SQL-a



# Podrška za vremenske podatke

- Mnoge vrste podataka se menjaju tokom vremena, i različiti korisnici i aplikacije imaju zahteve da pristupe podacima u različitim trenucima vremena
  - Umesto kreiranja odvojenih istorijskih tabela, tragera, i/ili implementiranje snapshot tabela, DBMS sa osobinama vremenske podrške može da upravlja istorijskim aspektima podataka
- Postoji dve vrste vremenskih podataka koji su podržani:
  - Vreme poslovanja
  - Sistemsko vreme



# Vreme poslovanja vs. Sistemsko Vreme

- **Vreme poslovanja** (aka vreme aplikacije ili validno vreme)
  - Određuje kada su činjenice koje su smeštene u bazi ispravne u odnosu na realnu situaciju
  - To su datumi od značaja korisniku koji komunicira sa podacima
  - Vreme poslovanja je korisno samo za određene vrste podataka koji se menjaju tokom vremena i validnost ovih podataka je značajnoza aplikaciju i korisnike.
- **Sistemsko vreme** (tj. vreme transakcije)
  - Označava kada je činjenica važeća u bazi podataka
  - Može da se koristi da se prati istorija insert-a i modifikacije podataka.
  - Za razliku od vremena poslovanja, vreme transakcije može da se poveže sa bilo kojim entitetom baze podataka.



# DBMS podržava i vreme poslovanja i sistemsko vreme

- **Oba su implementirana preko specifikacije vremenskog perioda**
- **Vreme Poslovanja** se prati u jednoj tabeli
  - Periodi početka i kraja indiciraju koji redovi odgovaraju kom vremenskom periodu.
- **Sistemsko vreme** se prati koristeći dve tabele.
  - Jedna tabela sadrži trenutne podatke.
  - Druga tabela, istorijska tabela, sadrži podatke koji nisu trenutni
  - I dalje je potrebno da se odrede početna i krajnja vremena  
da se odredi koji redovi odgovaraju kom vremenskom periodu
- **Jedna “logička” tabela može da se postavi i za poslovno i za sistemsko vreme**



# Primer

- Zašto bi nam trebalo upravljanje privremenim podacima?
  - Posmatrajte kompaniju OSIGURANJE
    - Uslovi osiguranja važe u toku nekog perioda vremena
    - Posle tog perioda vremena, klijent može da izabere da prekine osiguranje, nastavi sa važećim ili da modifikuje uslove osiguranja
    - U nekim određenim periodima vremena, uslovi polise mogu da se razlikuju
  - Tokom vremena, klijenti imaju potraživanja po polisi. Ova potraživanja treba da budu smeštena, obrađena i analizirana.
    - Istorije nezgoda klijenata su takođe važni podaci sa elementom privremenosti
  - Posmatrati kompleksnost pokušaja razvoja, ne samo dizajna podataka koji se prilagođavaju promeni polise, potraživanja i istorijskih detalja, već i omogućava upite tako da korisnik može da pristupi pokrivenosti klijenta u nekom trenutku vremena
    - Primer: koje polise su bile efektivne za klijenta na datum 15. 04. 2012? Ili neki drugi datum za koje vreme klijent je imao pokrivenost polisom.

# Primer

```
SELECT CustName, PolicyNo, BenefitSummary  
FROM InsurancePolicy  
      FOR BUSINESS_TIME AS OF '2012-04-15'  
WHERE CustNo = ?;
```



# Implementacija vremenskih osobina (Wiki)

The following implementations provide temporal features in a relational database management system (RDBMS).

- MariaDB version 10.3.4 added support for SQL:2011 standard as "System-Versioned Tables".
- Oracle Database – Oracle Workspace Manager is a feature of Oracle Database which enables application developers and DBAs to manage current, proposed and historical versions of data in the same database.
- PostgreSQL version 9.2 added native ranged data types that are capable of implementing all of the features of the pgFoundry temporal contributed extension. The PostgreSQL range types are supported by numerous native operators and functions.
- Teradata provides two products. Teradata version 13.10 and Teradata version 14 have temporal features based on TSQL2 built into the database.
- IBM DB2 version 10 added a feature called "time travel query" which is based on the temporal capabilities of the SQL:2011 standard.
- Microsoft SQL Server introduced Temporal Tables as a feature for SQL Server 2016. The feature is described in a video on Microsoft's "Channel 9" web site.

