Indexarea bazelor de date relaționale

Curs 5

Regăsirea datelor

- Interogare folosind egalități:
  - “*Find student name whose Age = 20*”
- Interogare folosind intervale:
  - “*Find all students with Grade > 8.50*”

- Parcurgerea secvențială a fișierului este costisitoare

- Dacă datele sunt sortate în fișier:
  - Căutare binară pentru găsirea primei înregistrări (*cost ridicat*)
  - Parcurgerea fișierului pentru găsirea celorlalte înregistrări.

Indecși

- Indecșii sunt fișiere/structuri de date speciale utilizate pentru accelerarea execuției interogărilor.
- Un index se creează pe baza unei *chei de căutare*
- *Cheia de căutare* e un atribut/set de atribute folosit(e) pentru a căuta înregistrările dintr-o tabelă/fișier.
- *Cheia de căutare* este diferită de cheia primară /cheia candidat /supercheia
- Un index conține o colecție de înregistrări speciale și permite regăsirea eficientă a "*tuturor înregistrărilor tabelei indexate pentru care cheia de căutare are valoarea* ***k***".

Proprietățile indecșilor

- *Propagarea modificărilor*
  - Adăugarea/ștergerea de înregistrări în tabela indexată implică actualizarea tuturor indecșilor definiți pentru acea tabelă.
  - Orice modificare a valorilor câmpurilor ce aparțin cheii de căutare presupune actualizarea indecșilor bazați pe acea cheie de căutare

Proprietățile indecșilor

- *Dimensiunea indecșilor*
  - Deoarece utilizarea unui index presupune citirea acestuia în memoria internă → dimensiunea indexului trebuie să fie redusă.
  - Ce se întâmplă dacă indexul e (totuși) prea mare?
    - Utilizare structură de indexare parțială
    - Se indexează fișierul index (stratificat sau pe o structură arborescentă)

Indecși

- Teme importante:
  - Care este structura unei înregistrări de index? *(conținutul indexului)*
  - Cum sunt organizate aceste înregistrări ? (*tehnica de indexare*)

Variante de structurare a conținutului unui index

(1) înregistrările originale ce includ cheia de căutare

(2) <**k**, *rid*>, rid – referință spre înregistrarea cu valoarea **k** pentru cheia de căutare

(3) <**k**, listă de *rid*-uri>

- Ex.. un index construit pe atributul varsta poate conține intrările (*entries*) <varsta, sid>, unde sid identifică o înregistrare student.

- Alegerea variantei de stocare a intrărilor (înregistrărilor) din index e oarecum independenta de tehnica de indexare.
  - Exemple de tehnici de indexare: arbori B^+, acces direct
  - Indexul mai conține și informații auxiliare de direcționare a căutărilor spre înregistrările căutate

Variante de structurare a conținutului unui index

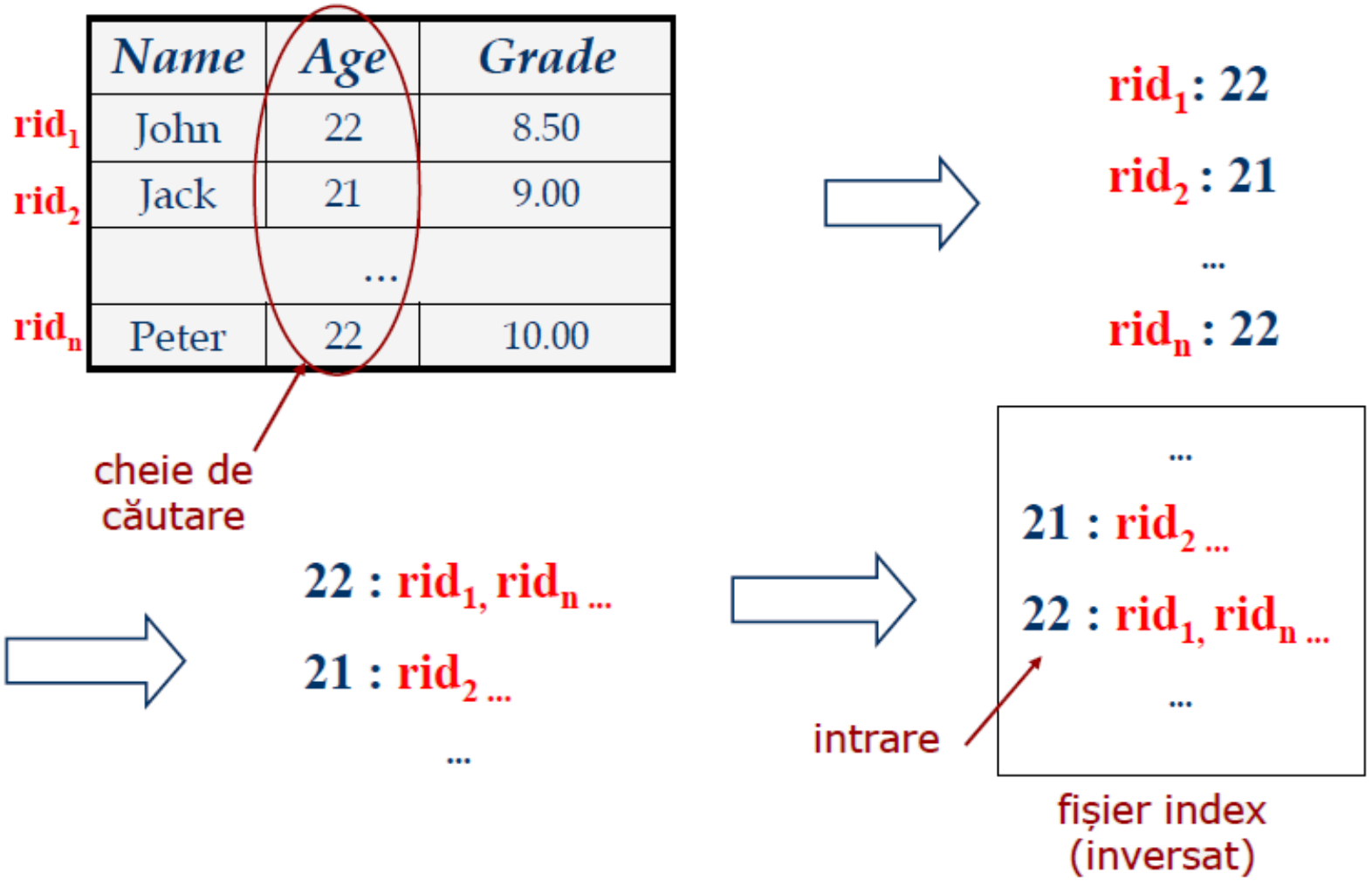
- Varianta(1):
  - Indexul și înregistrările originale sunt stocate împreună
  - Pentru o colecție de înregistrări se poate crea cel mult un index structurat conform variantei(1)
  - (în caz contrar → înregistrări duplicate → redundanță → potențiale inconsistențe)
  - Dacă înregistrările sunt voluminoase, numărul de blocuri în care se stochează înregistrările este mare → dimensiunea informației auxiliare din index este la rândul său mare.

Variante de structurare a conținutului unui index

- Variantele (2) și (3):
  - Intrările din index referă înregistrările originale
    - Intrările din index sunt în general mai mici decât înregistrările (deci sunt variante mai eficiente decât varianta (1),mai ales dacă cheia de căutare este mică).
    - Informația auxiliară din index folosită pentru a direcționa căutarea, este la rândul său mai redusă decât în cazul variantei(1).

- Varianta(3) este mult mai compactă decât varianta (2), dar implică dimensiune variabilă a spațiului de stocare a intrărilor, chiar dacă cheia de căutare este de dimensiune fixă.

Crearea unui index



Clasificarea indecșilor

- primari / secundari , unique / non unique
- clustered / un-clustered
- denși / rari
- cu chei de căutare simple / chei de căutare compuse

Clasificarea indecșilor

- Indecși ***primari*** vs ***secundari***:
  - Un index *primar* se formează pe baza unei chei de căutare ce include cheia primară.
  - Un index e ***unic*** atunci când cheia de căutare conține o cheie candidat
    - Nu vor fi intrări duplicate
  - În general, indecșii *secundari* conțin duplicate

- Indecși ***unique*** vs ***non unique***:
  - un index unique garantează că nu există valori duplicate în cheia indexului
  - la crearea unei chei principale sau secundare (PRIMARY KEY sau UNIQUE), se creează automat un index unique pe coloanele precizate
  - nu se poate crea un index unique dacă există date duplicate în coloanele cheii

Clasificarea indecșilor

- Indecși ***clustered*** vs. ***un-clustered***:
  - Un index este *clustered (*grupat*)* dacă ordinea înregistrărilor este aceeași (sau foarte apropiată) cu ordinea intrărilor din index.
  - Varianta (1) implică grupare; de asemenea în practică, gruparea implică utilizarea primei variante de structurare a indecșilor.
  - O tabelă poate fi indexată grupat (*clustered*) cel mult o pentru o cheie de căutare.
  - Costul regăsirii datelor prin intermediul unui index e influențat *decisiv* de grupare!

Index Clustered vs. Un-clustered

- Pentru a construi indecși grupați:
  - Se ordonează înregistrările în cadrul fișierului (*heap file*)
  - Se rezervă spațiu în fiecare pagină de memorie, pentru a se absorbi viitoarele inserări
    - dacă spațiul suplimentar e consumat, se vor forma liste înlănțuite de pagini suplimentare
    - e necesară o reorganizare periodică pentru a se asigura o performanță ridicată
- Întreținerea indecșilor grupați e foarte costisitoare

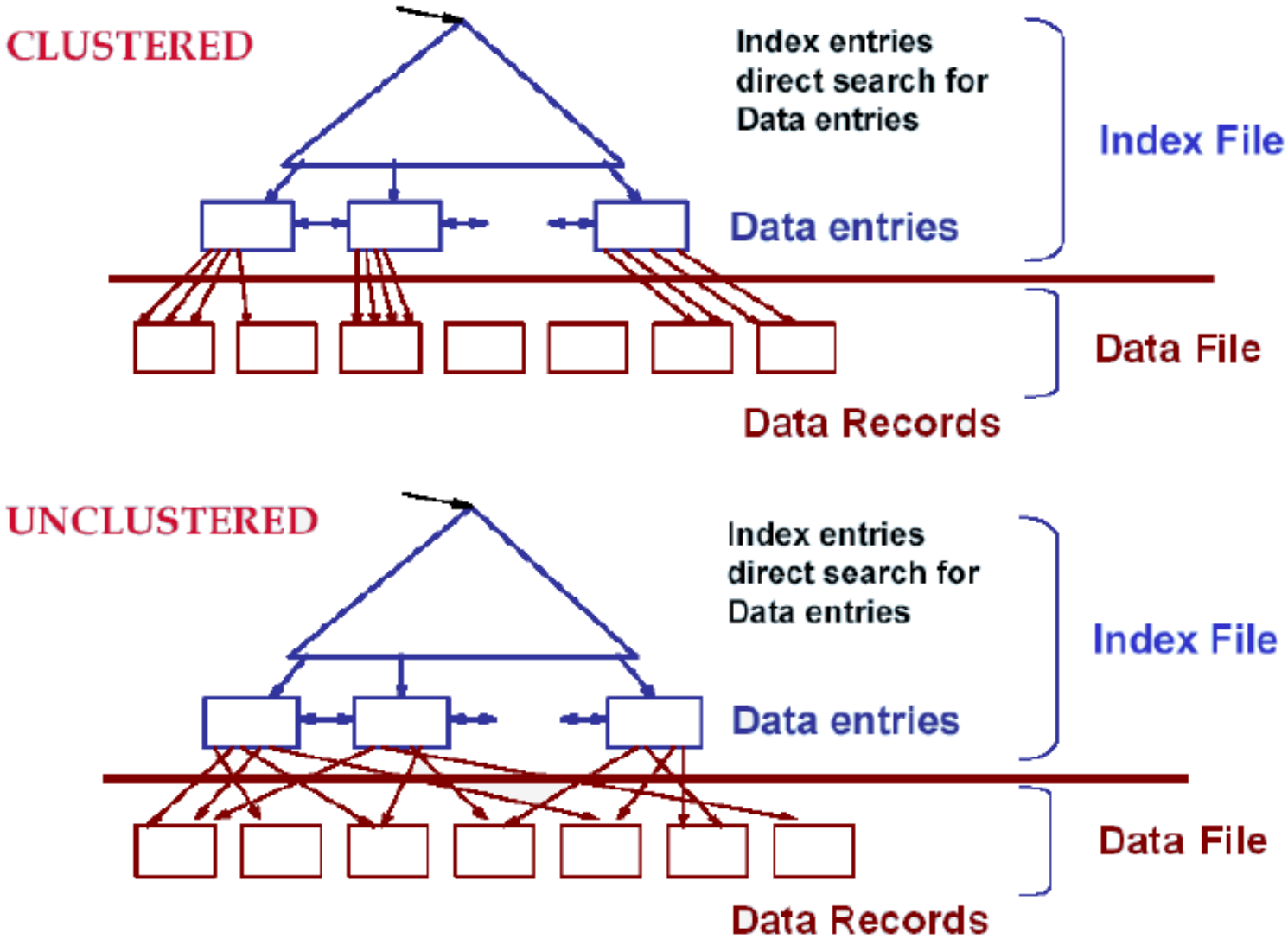
Index Clustered vs. Un-clustered

- La crearea unei chei primare pe un tabel, dacă nu e definit un index clustered și nu e specificat un index nonclustered, se creează un index unique clustered

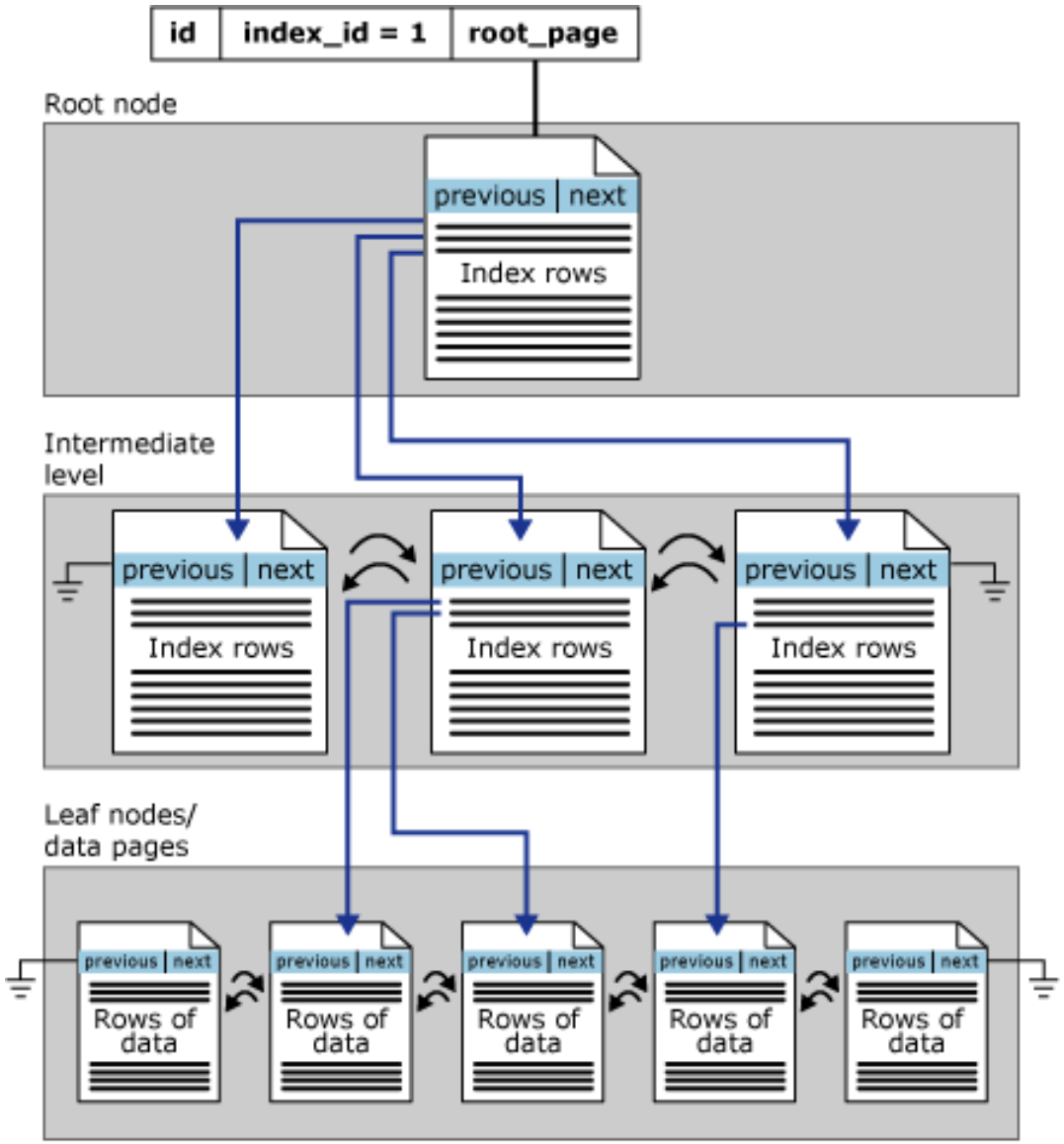
- Dimensiunea cheii → să fie cât mai mică

- Index clustered – util pentru interogări care se execută frecvent / căutări pe intervale de valori

Index Clustered vs. Un-clustered



Index clustered în SQL Server organizat ca B-arbore

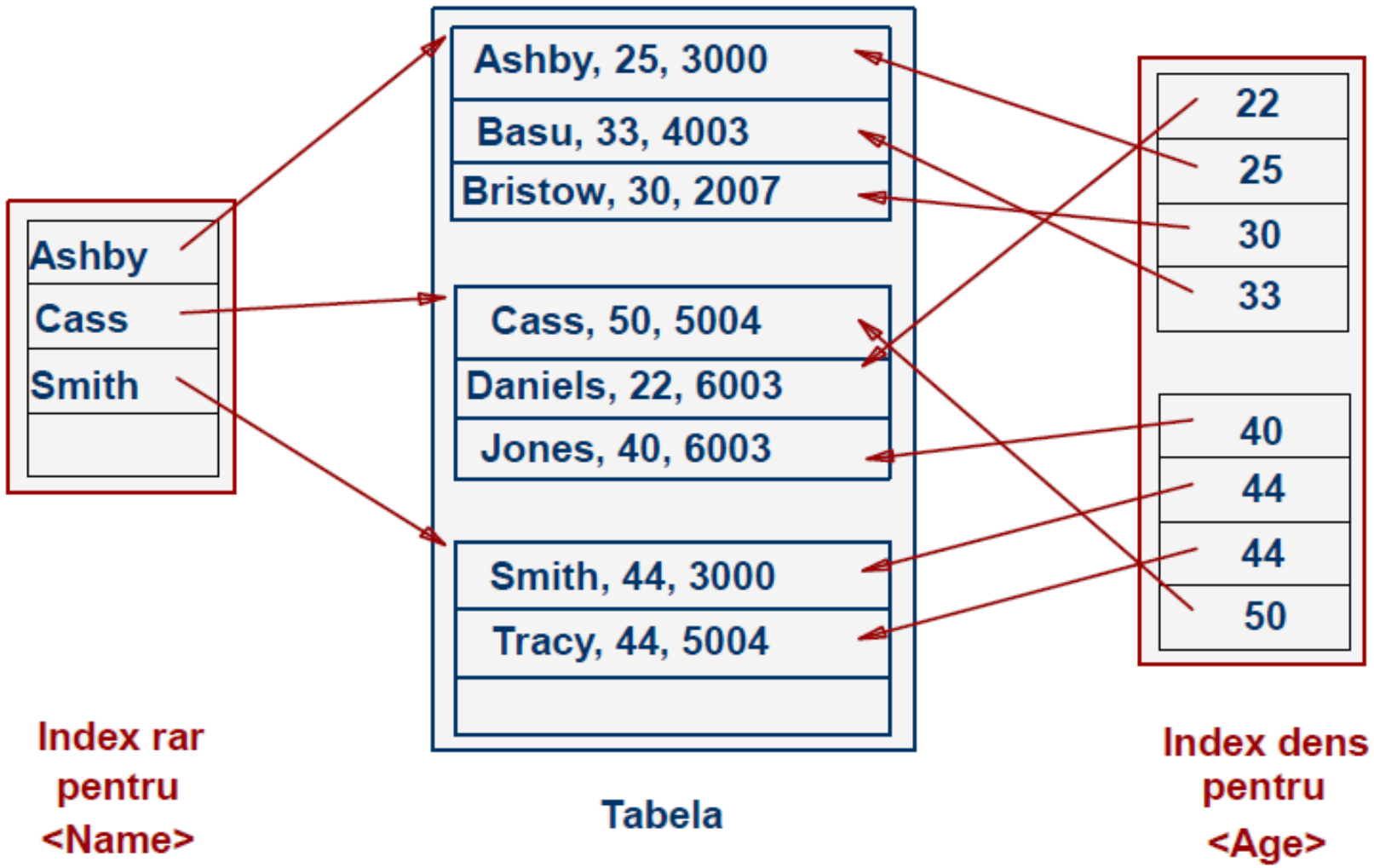


Clasificarea indecșilor

- Indecși ***denși*** vs. ***rari***:
- Un index este ***dens*** dacă are cel puțin o intrare pentru fiecare valoare a cheii de căutare (ce se regăsește în înregistrări)
  - Mai multe intrări pot avea aceeași valoare pentru cheia de căutare în cazul utilizării variantei (2) de stocare
  - Varianta (1) presupune implicit ca indexul e dens.

- Un index e ***rar*** dacă memorează o intrare pentru fiecare pagină de înregistrări
  - Toți indecșii rari sunt grupați (*clustered*)!
  - Indecșii rari sunt mai compacți.

Indecși *denși* vs. *rari*



Clasificarea indecșilor

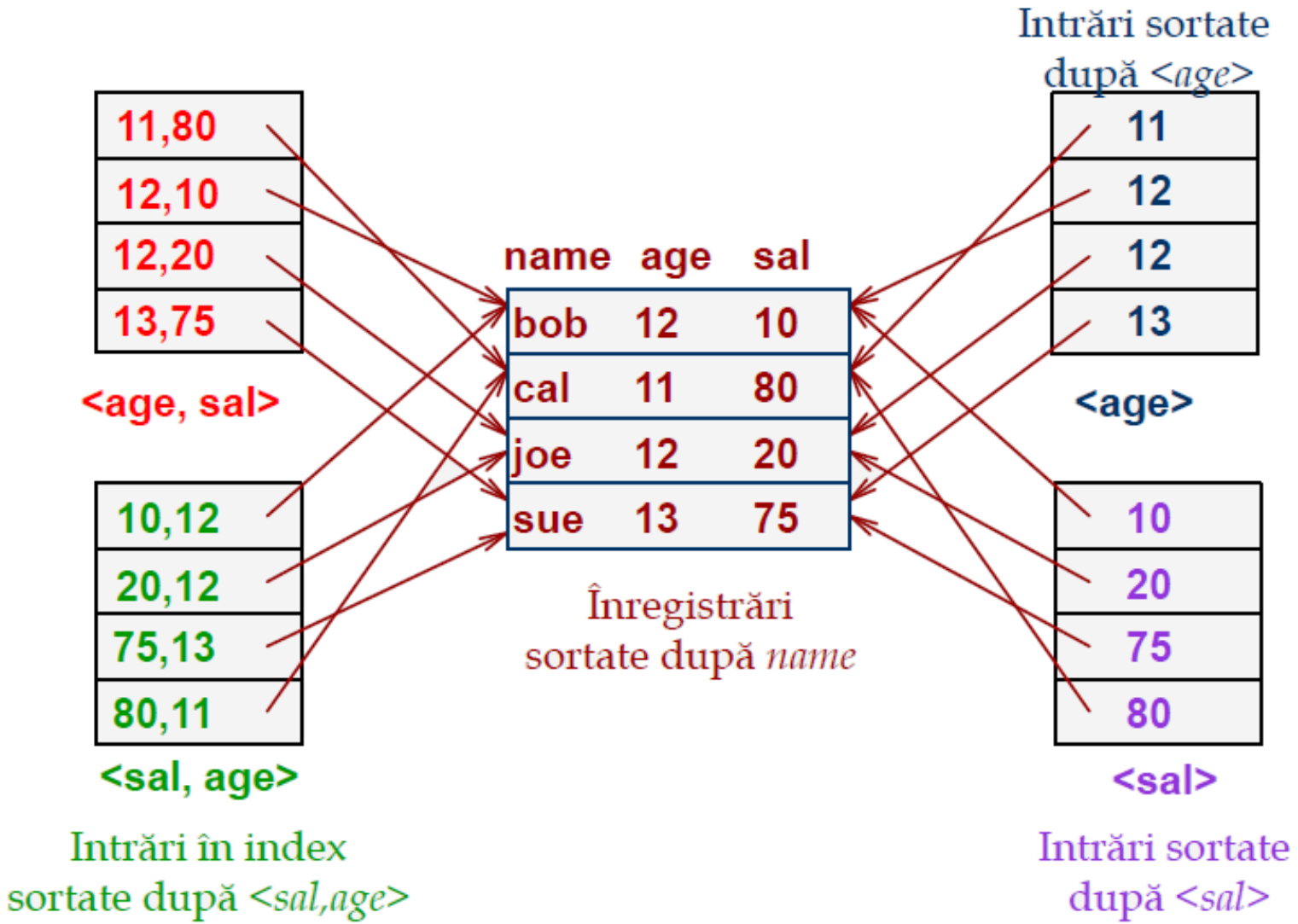
- Indecși ***cu chei de căutare simple*** vs. ***chei de căutare compuse***:
  - O cheie de căutare se numește compusă atunci când conține mai multe câmpuri
  - Cheiele de căutare compuse sunt utile atunci când căutarea se realizează după o combinație de câmpuri
    - Interogări cu egalitate: valoarea fiecărui câmp e egală cu o valoare constantă

      age=20 and sal =75
    - Interogări cu interval: valoarea unui câmp nu e o constantă

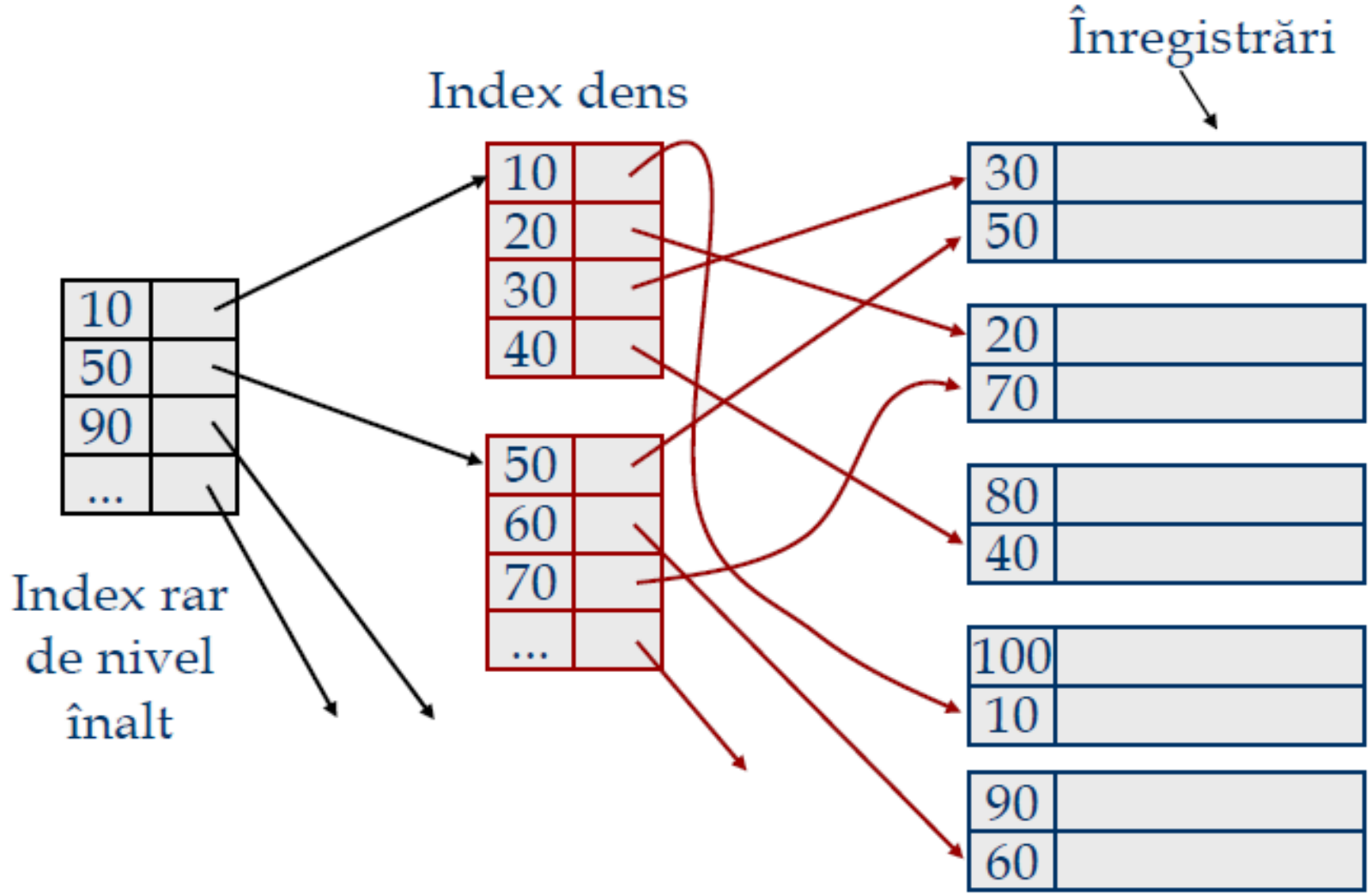
      age=20 and sal > 10

- Intrările din index sunt sortate după cheia de căutare pentru a putea fi utile interogărilor cu interval.
  - Linear (ordine lexicografică)
  - Ordine spațială.

Clasificarea indecșilor



Indecși Secundari



Exemplu

- Înregistrările tabelei Students sunt stocate într-un fișier sortate după câmpul *Age*.Fiecare pagină poate stoca un număr maxim de 3 înregistrări
- Determinați intrările în fiecare dintre următorii indecși (folosiți <*page_id, slot no*> pentru identificarea unui tuplu).

ID	Name	Age	Grade	Course
123	Melanie	18	7.8	DB1
124	Georg	19	8.0	DB1
110	Jan	25	9.4	Alg1
112	Sammy	26	9.2	DB1
100	Sammy	26	9.8	Alg1

ID	Name	Age	Grade	Course
123	Melanie	18	7.8	DB1
124	Georg	19	8.0	DB1
110	Jan	25	9.4	Alg1
112	Sammy	26	9.2	DB1
100	Sammy	26	9.8	Alg1

1. Age – dens, varianta (1)
   - întreaga tabelă
2. Age – dens, varianta (2)
   - (18, <1,1>), (19, <1,2>), (25, <1,3>), (26, <2,1>), (26, <2,2>)
3. Age – dens, varianta (3)
   - (18, <1,1>), (19, <1,2>), (25, <1,3>), (26, <2,1>, <2,2>)
4. Age – rar, varianta (1)
   - un astfel de index nu poate fi construit (definiție)

ID	Name	Age	Grade	Course
123	Melanie	18	7.8	DB1
124	Georg	19	8.0	DB1
110	Jan	25	9.4	Alg1
112	Sammy	26	9.2	DB1
100	Sammy	26	9.8	Alg1

5. Age – rar, varianta (2)
   - (18, <1,1>), (26, <2,1>) - ordinea intrărilor e semnificativă
6. Age – rar, varianta (3)
   - (18, <1,1>), (26, <2,1>, <2,2>) - ordinea intrărilor e semnificativă
7. Note – dens, varianta (1)
   - Ar trebui sortate înregistrările după notă: 7.8, 8.0, 9.2, 9.4, 9.8
8. Note – dens, varianta (2)
   - (7.8, <1,1>), (8.0, <1,2>), (9.2, <2,1>), (9.4, <1,3>), (9.8, <2,2>)

ID	Name	Age	Grade	Course
123	Melanie	18	7.8	DB1
124	Georg	19	8.0	DB1
110	Jan	25	9.4	Alg1
112	Sammy	26	9.2	DB1
100	Sammy	26	9.8	Alg1

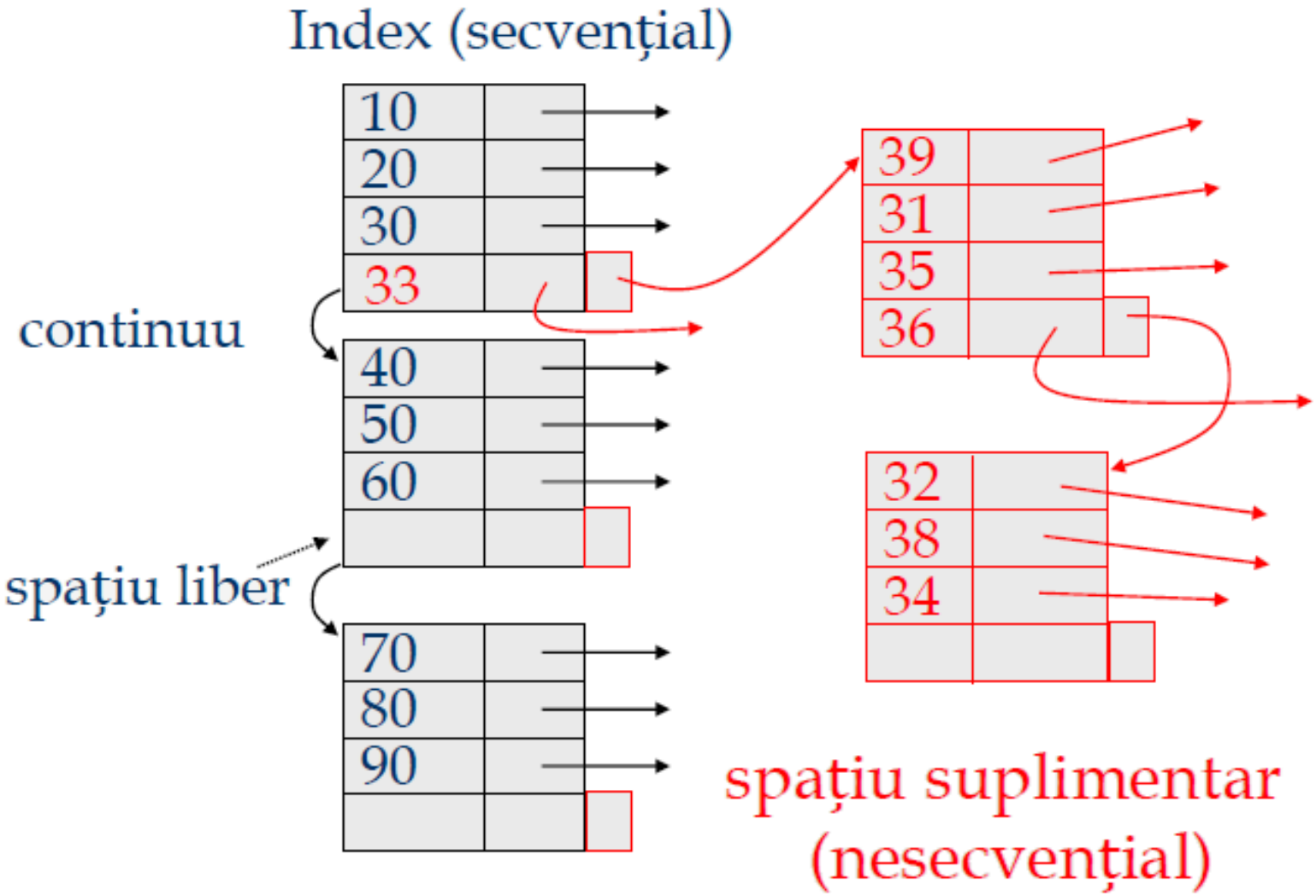
9. Note – dens, varianta (3)
   - (7.8, <1,1>), (8.0, <1,2>), (9.2, <2,1>), (9.4, <1,3>), (9.8, <2,2>)
10. Note – rar, varianta (1)
   - un astfel de index nu poate fi construit (definiție)
11. Note – rar, varianta (2)
   - nu e posibil (valorile cheii de căutare nu sunt ordonate)
12. Note – rar, varianta (3)
   - nu e posibil (valorile cheii de căutare nu sunt ordonate)

Indecși convenționali

- Avantaje:
  - Simpli
  - Indexul e un fișier secvențial → potrivit pentru parcurgeri

- Dezavantaje:
  - Inserările sunt costisitoare, și/sau
  - Se pierd secvențialitatea & echilibrul

Exemplu


Index (secvențial)
10
20
30
33
continuu
40
50
60
spațiu liber
70
80
90
39
31
35
36
32
38
34
spațiu suplimentar
(nesecvențial)

Înțelegere *workload*

- Pentru fiecare interogare utilizată:
  - Care sunt relațiile/tabelele accesate?
  - Care sunt atributele/câmpurile returnate?
  - Care dintre atribute sunt implicate în condiții de selecție/join? Cât de selective sunt aceste condiții?

- Pentru fiecare actualizare:
  - Care dintre atribute sunt implicate în condiții de selecție/join? Cât de selective sunt aceste condiții?
  - Ce tip de actualizare este (INSERT/DELETE/UPDATE), și care sunt atributele afectate?

Alegerea indecșilor

- Ce index trebuie creat?
- Pentru fiecare index, care e varianta de structurare utilizată?
- **Abordare**: Se analizează cele mai importante interogări. Pentru fiecare se consideră cel mai optim plan de execuție folosind indecșii existenți. Vom crea un nou index doar dacă acesta va genera un plan mai bun.
- Evident, acest lucru implică să înțelegem cum evaluează un SGBD interogările și cum sunt create **planurile de execuție**!

- Înainte de a crea un nou index, trebuie să evaluăm **impactul** actualizării acestuia!
- Indecșii accelerează execuția interogărilor, dar încetinesc actualizările. De asemenea, necesită spațiu suplimentar de stocare.

Alegerea indecșilor

- Crearea indecșilor nonclustered pe coloane utilizate frecvent în WHERE și JOIN

- Atributele din clauza WHERE sunt chei de căutare “candidat”
  - Egalitățile sugerează o structură cu acces direct

  - Intervalele sugerează utilizarea unei structuri arborescente
    - Gruparea e foarte utilă în aceste situații; dar și în cazul egalităților atunci când numărul de duplicate este mare.

Alegerea indecșilor

- Atunci când clauza WHERE are mai multe condiții se pot lua în considerare chei de căutare compuse
  - Ordinea atributelor e importantă pentru a evalua condițiile cu intervale
  - Pentru indecșii cu chei compuse – plasarea coloanelor care apar în condiții de căutare cu =, >, < pe prima poziție în index
  - Pentru interogările importante acești indecși pot determina strategii de execuție index-only

Alegerea indecșilor

- Vor fi aleși indecșii de care beneficiază cele mai multe interogări posibile. Deoarece se poate defini un singur index grupat pe o tabelă, acesta va fi ales astfel încât să beneficieze de acest index o interogare foarte important

- Păstraţi lungimea cheii de căutare cât mai redusă pentru indecşii clustered

- Indecşii *clustered* funcţionează f bine pe coloane unice şi nenule

Alegerea indecșilor

- Evitarea supraindexării tabelelor care sunt modificate foarte des
- Utilizarea indecșilor care acoperă întreaga interogare, aceștia putând îmbunătăți performanța acesteia
- Evitarea indexării tabelelor de dimensiuni reduse
- Evitarea indexării coloanelor care au puține valori distincte – examinarea distribuției datelor

Alegerea indecșilor

- Indexare bună → aplicații rapide
- Indexare slabă → poate încetini întregul server

Chei de căutare compuse

- Pentru returnarea înregistrărilor din *Employees* cu *age* = 30 AND *sal* = 4000, un index pe <*age,sal*> e mai potrivit decât un index pe *age* sau unul pe *sal*.

- Dacă avem: 20 < *age* < 30 AND 3000 < *sal* < 5000:
  - Un index arborescent grupat pe <*age,sal*> sau <*sal,age*> e foarte bun.

- În cazul: *age*=30 AND 3000 < *sal* < 5000:
  - Index grupat<*age,sal*> e mai bun decât index pe <*sal,age*>
- Indecșii compuși sunt mai voluminoși și sunt actualizați mai des

Chei de căutare compuse - Matching

	Condiție	Index B-arbore	Hash-Index
1	a = 5 AND b = 3	☑	☒
2	a > 5 AND b < 3	☑	☒
3	b = 3	☒	☒
4	a = 7 AND b = 5 AND c = 4 AND d > 4	☑	☑
5	a = 7 and c = 5	☑	☒

- Dacă avem un index pe cheia de căutare compusă **<a,b,c>**, acesta poate fi folosit pentru predicate care respectă anumite condiții:
  - Condițiile compuse sunt legate cu **AND** între ele
  - **Hash-Index** (tabelă de dispersie) – atunci condițiile trebuie să fie egalități și predicatul trebuie să conțină toate atributele din cheia de căutare
  - **Index B-arbore** – putem avea *match* doar pentru atribute care reprezintă un prefix al cheii de căutare, dar pot fi și alte tipuri de condiții în afară de egalități (de ex. inegalități)

Planuri *Index-Only*

- Anumite interogări pot fi executate fără a accesa tabelele originale atunci când este disponibil un index potrivit.

<E.dno,E.eid> Tree index!

```
SELECT  D.mgr, E.eid
FROM  Dept D, Emp E
WHERE  D.dno=E.dno
```

<E.dno>

```
SELECT E.dno, COUNT(*)
FROM Emp E
GROUP BY E.dno
```

<E. age, E.sal> or *<E.sal, E.age>*
Tree index!

```
SELECT AVG(E.sal)
FROM Emp E
WHERE E.age=25 AND
 E.sal BETWEEN 3000 AND 5000
```

Indecși

- Sintaxă în SQL Server

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
INDEX nume_index
ON <obiect> (coloana[ ASC | DESC ][ ,...n ] )
[ INCLUDE (nume_coloana[ ,...n ] ) ]
[ WHERE <predicat_filtrare> ]
```

Indecși clustered / nonclustered

Crearea unui index clustered:

```
CREATE CLUSTERED INDEX Nume_Index ON Nume_Tabel(coloana)
[ASC | DESC];
```

Indecși clustered / nonclustered

- Index nonclustered
  - valorile cheii și un pointer către datele din heap / indexul clustered
  - ordinea fizică a rândurilor este independentă de ordinea indexată a acestora
  - cel mult 999 indecși nonclustered pe tabel în SQL Server

```
CREATE INDEX Nume_Index ON Nume_Tabel(coloana)[ASC | DESC];
```

- limită cheie (clustered sau nonclustered) – 16 coloane / 900 octeți (verificați versiunea)
- tabel în SQL Server
  - tabel clustered – are un index clustered
  - heap – nu are index clustered

Indecși cu coloane cheie / care nu fac parte din cheie

- coloanele care nu fac parte din cheie – coloane adăugate în clauza INCLUDE a unui index nonclustered

```
CREATE INDEX Nume_Index ON Nume_Tabel(coloana)
INCLUDE (coloana1, coloana2)
```

- Avantaje:
  - coloanele pot fi accesate când se scanează indexul
  - Pentru coloanele non-cheie e permis orice tip de date
  - Coloanele non-cheie nu sunt luate în considerare la calculul limitei de 900 bytes
- Dacă toate câmpurile returnate într-o interogare se află în index: *covering index*

Indecşi filtraţi

Index filtrat: un index *non-clustered* optimizat, potrivit pentru acoperirea interogărilor ce selectează date dintr-un subset bine definit

```
CREATE NONCLUSTERED INDEX FI_EndDate ON
Products (ProductID, EndDate)
WHERE EndDate IS NOT NULL ;
GO
```

- Avantaje:
  - Crește performanța interogării
  - Reduce costurile de întreținere a indexului
  - Reduce costurile de stocare a indexului

Dezactivarea Indecşilor

```
ALTER INDEX IX_Address_StateProvinceID
ON Person.Address DISABLE
```

Reactivarea Indecşilor

```
ALTER INDEX IX_Address_StateProvinceID
ON Person.Address REBUILD
```

Vizualizarea indecșilor

- Procedura de sistem ***sp_helpindex***

```
EXEC sys.sp_helpindex @objname = N’Table1’
```

- Cu ajutorul tabelelor și a view-urilor de sistem:

```
SELECT *
FROM sys.indexes
WHERE object_id = OBJECT_ID(‘Table1')
```

Indecşi pentru ştergere

- La DELETE:
  - SQL Server verifică dependenţa înregistrărilor prin examinarea tuturor cheilor străine
  - Toate tabelele identificate vor fi verificate dacă nu conţin referinţe către înregistrarea de şters
    - Dacă există un index, SQL Server îl va folosi la verificare
    - Dacă nu există un index, SQL Server va trebui să scaneze întreaga tabelă
  - Ştergerile pot fi foarte lente în absenţa indecşilor pentru chei străine

Fragmentare

- *Fragmentare internă*: înregistrările nu sunt stocate într-o zonă contiguă în interiorul paginii. Fragmentarea internă apare dacă este spaţiu neutilizat între înregistrări într-o pagină.

- Gradul de "umplere" a unei pagini poate varia în timp. Acest spațiu neutilizat duce la o utilizare ineficientă a cache-ului și la mai multe transferuri de pagini de memorie între disc și memoria internă.

Fragmentare

- *Fragmentare externă* : Pe disc, paginile şi extensiile (*extent* - grup de 8 pagini de memorie*)* nu sunt stocate într-o zonă contiguă. Trecerea de la o extensie la alta va cauza rotaţii mai mari ale discului.

- *Fragmentare logică*: Fiecare pagină a unui index este legată de precedenta şi următoarea pagină in ordinea logică a valorilor cheii. Din cauza umplerii unora dintre pagini si a redistribuirii valorilor, paginile devin *out-of-order*.
- O pagină *out-of-order* este o pagină pentru care următoarea pagină fizică din index nu este şi următoarea pagină logică.

Fragmentare

- sys.dm_db_index_physical_stats
  - **avg_fragmentation_in_percent**: valoare procentuală ce reprezintă fragmentarea externă
  - **avg_page_space_used_in_percent:** medie procentuală a utilizării paginilor corespunzătoare fragmentării interne.

Reducerea Fragmentării

Reducerea fragmentării in *heap*:

- Pentru reducerea fragmentării in *heap* se adaugă un index *clustered* tabelei

- Crearea indexului *clustered*: se ordonează înregistrările, și apoi se plasează în zone contigue pe disc.

Reducerea Fragmentării

Reducerea fragmentarii intr-un Index:

- Dacă *avg_fragmentation_in_percent* > 5% si < 30%, atunci folosiți ALTER INDEX REORGANIZE:
  - reordonează frunzele indexului dupa cheie.
- Dacă *avg_fragmentation_in_percent* > 30%, atunci folosiți ALTER INDEX REBUILD:
  - O variantă cu efect similar constă în ștergerea și re-crearea indexului.
- Ștergerea și re-crearea indexului *clustered*:
  - Re-crearea unui index clustered va determina redistribuirea datelor și are ca efect obținerea unor pagini de date nefragmentate si "pline". Gradul de umplere poate fi configurat prin opțiunea FILLFACTOR in CREATE INDEX.

Fragmentare

- Crearea unui index cu gradul de umplere dat:

```
CREATE NONCLUSTERED INDEX FI_EndDate ON
Products (ProductID ASC, EndDate DESC)
WITH (FILLFACTOR = 70) ;
GO
```

Planuri de execuție

- La generarea planului de execuție se alege algoritmul cu complexitatea cea mai mică în contextul existent (al bazei de date)– îmbunătățirea performanței interogării

- Pentru a accesa datele unui tabel:
  - Table Scan
  - Index Scan
  - Inde Seek

Table Scan

- pentru anumiți operatori este necesară parcurgerea tuturor înregistrărilor dintr-un tabel
- se aduce în memoria principală fiecare tuplu dintr-o relație

Index Scan

- acest algoritm se folosește pentru evaluarea unei condiții de forma:
- A=v, A<v, A<=v, A>v, A>=v, A IS NULL, A IS NOT NULL, indexul s-a construit pentru un atribut A

- înregistrările se obțin prin parcurgerea parțială sau totală a indexului și citirea înregistrărilor din relație conform adreselor furnizate de index (unele blocuri se pot citi de mai multe ori)

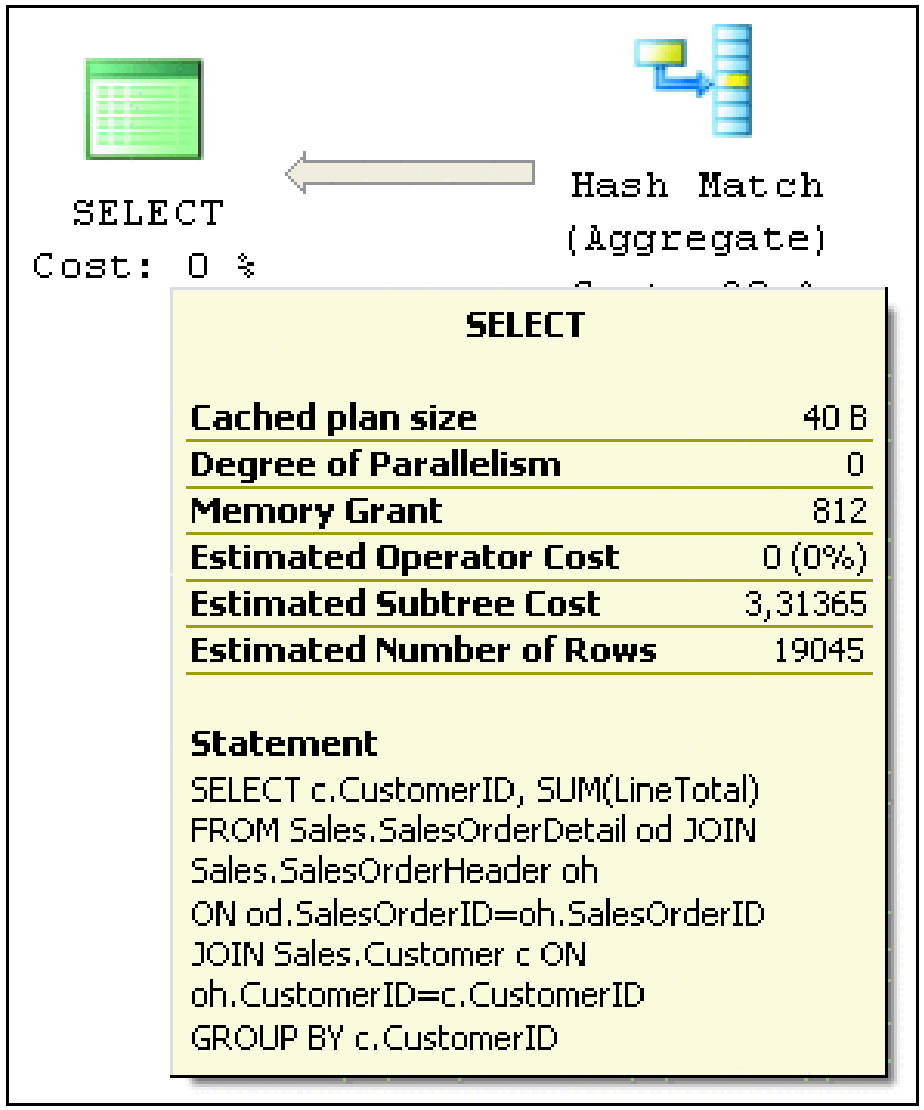
Index Seek

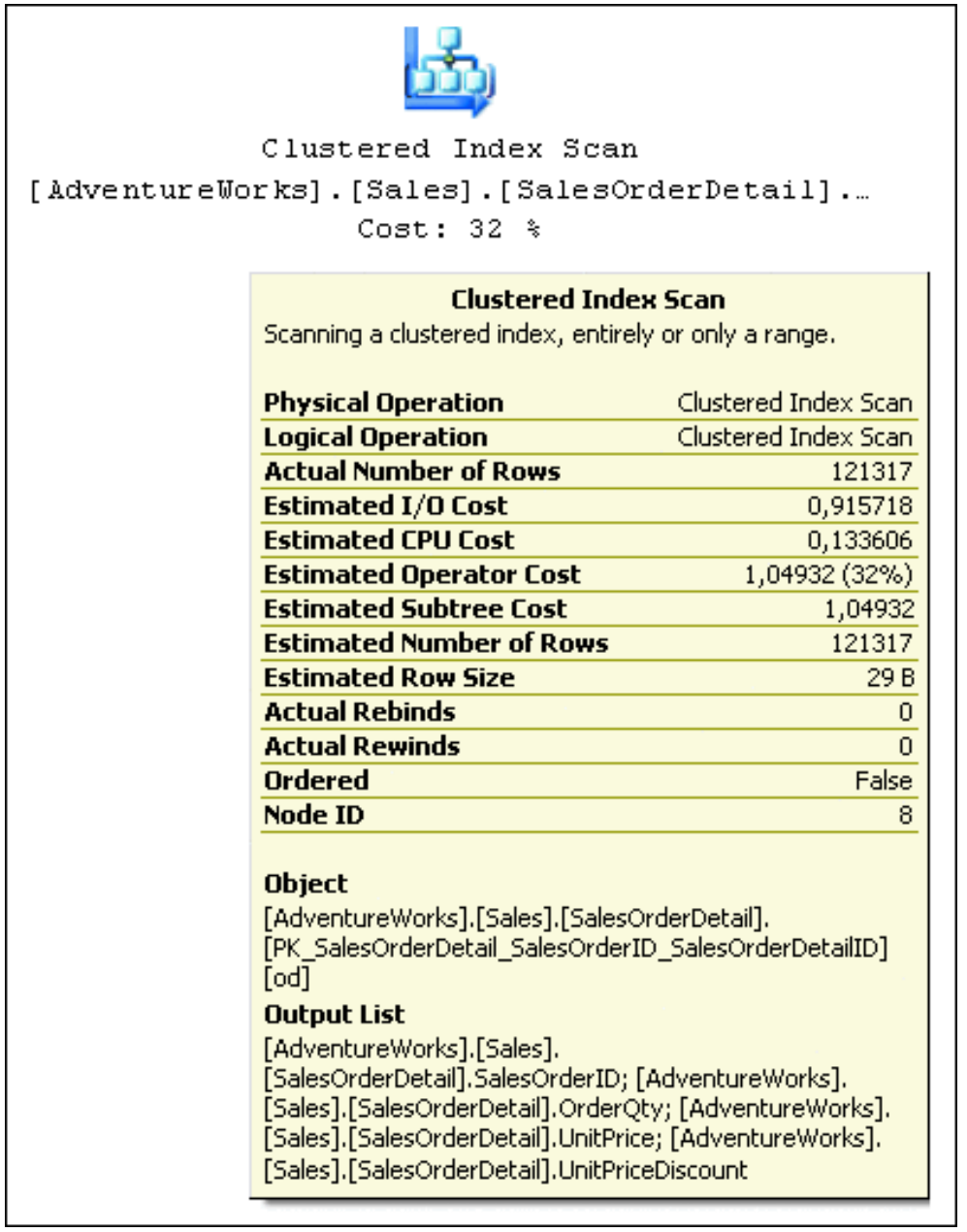
- Acest algoritm este cel mai performant și se folosește la căutarea după valoarea unei chei C
- Căutarea este de forma: C=K0
- Se consultă un index – C poate fi o cheie simplă sau compusă

Graphical execution plan

```
SELECT c.CustomerID, SUM(LineTotal)
FROM Sales.SalesOrderDetail od
JOIN Sales.SalesOrderHeader oh ON
od.SalesOrderID=oh.SalesOrderID
JOIN Sales.Customerc ON
oh.CustomerID=c.CustomerID
GROUP BY c.CustomerID
```

Graphical execution plan

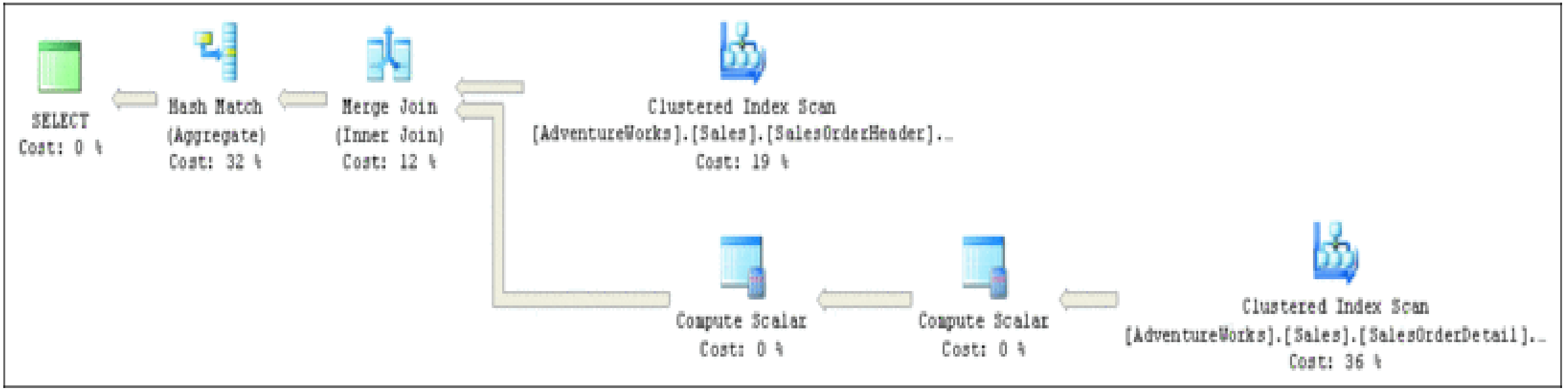




Graphical execution plan

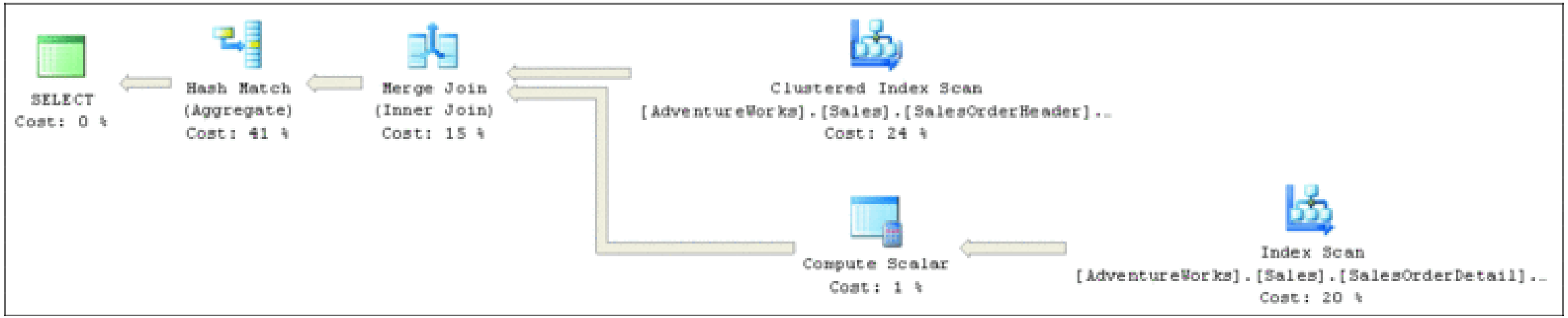
```
SELECT oh.CustomerID, SUM(LineTotal)
FROM Sales.SalesOrderDetail od
JOIN Sales.SalesOrderHeader oh ON
od.SalesOrderID=oh.SalesOrderID
GROUP BY oh.CustomerID
```

Graphical execution plan

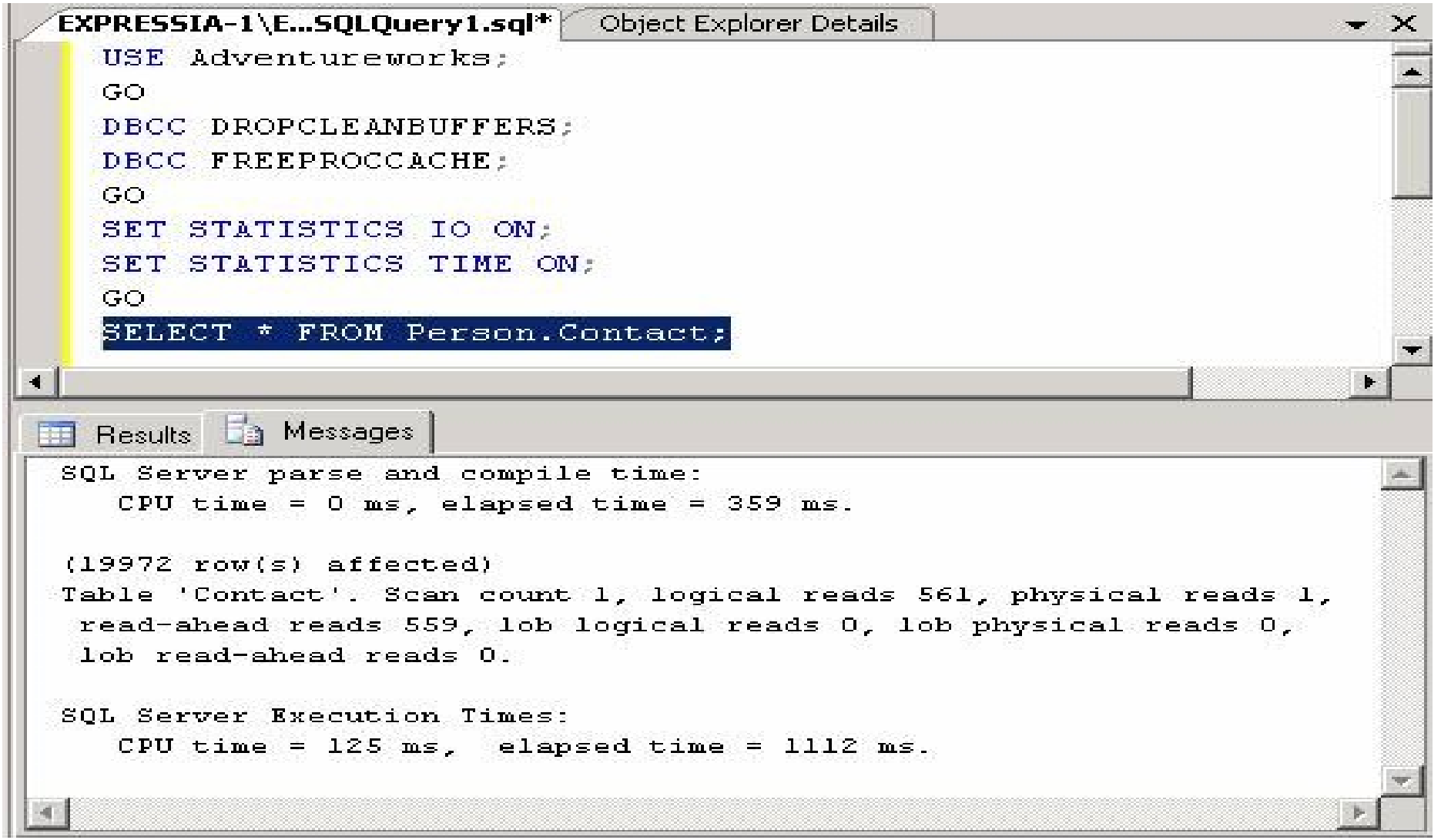


Graphical execution plan

```
CREATE INDEX IDX_OrderDetail_OrderID_TotalLine
ON Sales.SalesOrderDetail(SalesOrderID)
INCLUDE (LineTotal)
```



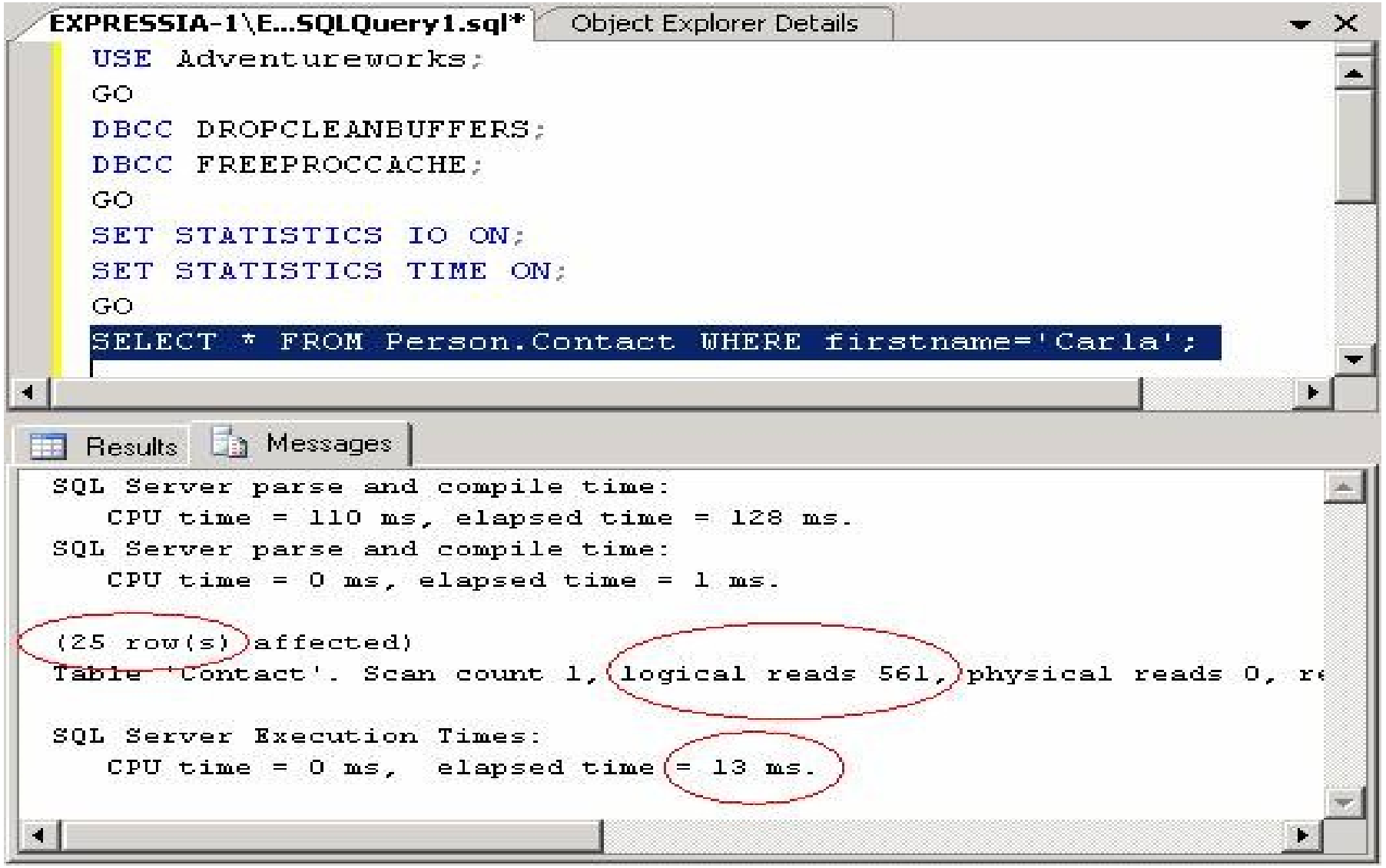
STATISTICS IO und STATISTICS TIME



STATISTICS IO und STATISTICS TIME

- Elapsed time – cât timp a durat sa se execute interogarea
- Physical reads – numărul de pagini citite de pe disc
- Read-ahead reads – numărul de pagini salvate in cache pentru interogare
- Scan count – de câte ori au fost accesate tabelele
- Logical reads – numărul de pagini citite din data cache

STATISTICS IO und STATISTICS TIME



STATISTICS IO und STATISTICS TIME

```
USE [AdventureWorks] GO
CREATE NONCLUSTERED INDEX [IDX_firstname] ON
[Person].[Contact]
(
[FirstName] ASC
)
GO
```

STATISTICS IO und STATISTICS TIME