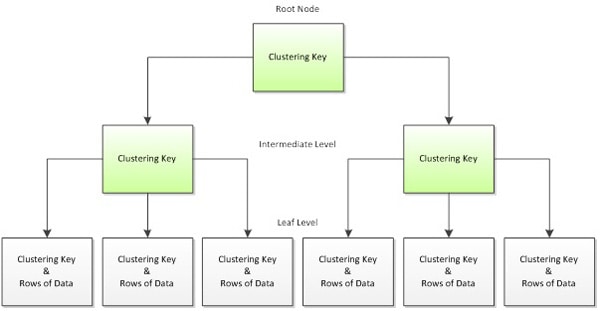
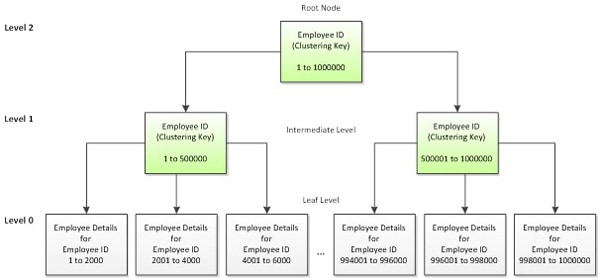
* **Indexul** structura de date asociata unui tabel pentru a extrage datele mai rapid
* Index nu ocupa prea multa memorie
* Index folosesc **Balanced Tree** data structure, desi nu functioneaza in mod identic cu BST. Asadar, toate keys luate sunt sortate. Asa, cautarea se face cu **O(logN)**
* Totusi, indexarea face mai rapide operatiile de cautare, insa le incetineste pe cele de insert si update,dar si delete, caci e necesar sa mai faca operatii suplimentare
* **Primary key** este mereu indexat, dar rolul de a crea manual index(non-clustered) este de asigura ca cautarea dupa alte atribute, ca de ex dupa email, va fi cam la fel de rapida ca cautarea dupa primary key.
* **Primary key este mereu un clustered index**
* Indexes sunt stocate in memoria disk.
* Fara indexes, orice tuple e cautat liniar.
* Primary key ar fi bine sa fie un int sau bigint, deoarece compararea numerelor e foarte usoara si rapida.
* Daca nu cream noi un primary key, oricum se va crea unul automat cu tot cu clustered index. Daca vom crea una, primary key recenta va fi stearsa.

**Clustered Index**

* **Clustered index** – index care defineste ordinea fizica in care tuples sunt stocate in tabel. Clustered index sorteaza datele chiar in tabel, dupa primary key.
* Din cauza ca primary key poate fi doar una, putem avea si doar un clustered index, caci e logic ca putem sorta datele in tabel doar dupa un sigur atribut.
* **Data page** – sunt fisiere in care sunt stocate datele dintr-un tabel. Daca avem un tabel, asta nu inseamna ca toate datele din acel tabel sunt stocate doar intr-un singur fisier. DBMS creaza fisiere mici, de 8 KB, ca sa poata lucra rapid cu ele, si apoi pe baza la aceste fisiere restabileste tabelul.
* **Clustered index** – spune anume cum sa fie ordonate data pages pentru tabelul respectiv. Aceste data pages sunt organizate intr-un Balanced Tree asa:



In imagine sunt doar 2 intermediate level(1 si 2), dar in realitate pot fi mult mai multe. Leafs sunt mereu level 0

* Toate nivelele deasupra la leaf, adica ultimul, contin Index Pages. Index Pages nu contin propriu zis datele, ci doar referinte catre alte index pages. In **Index Page** se pastreaza primary key si referinta paginii corespunzatoare aceostor keys.

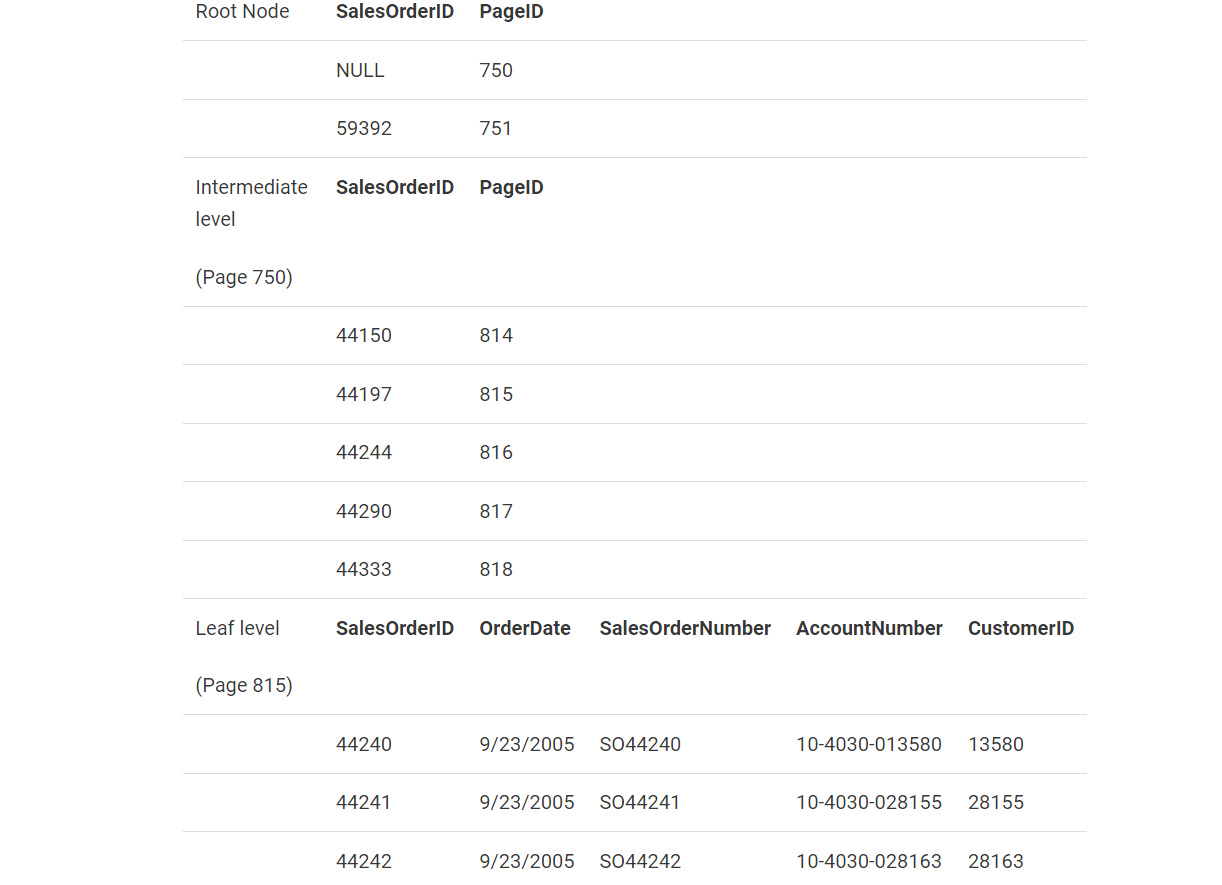
Deci, Level2 contine o index page cu 1000000 keys si spune fiecare key in ce alta pagina se gaseste(intervale). Apoi, In Level, Avem alte 2 index pages care deja ne spun concret in ce Data Pages se gasesc chiar datele noastre, nu alte index pages.

* Asa, cautarea la un tuple se face de la un Index Page la altul pana se ajunge la Data Page

**Exemplu:**

Fie ca facem un asa query:

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT  CustomerID ,          OrderDate ,          SalesOrderNumber  FROM    Sales.SalesOrderHeader  WHERE   SalesOrderID = 44242 ; |



De ex, fie tabelul SalesOrderHeader ce are foarte multe date. In root index page, se stocheaza intervalele la keys si spune in ce pagini se gasesc keys ce fac parte de la x la y, De ex, vedem ca keys de la 0 la 59392 se gasesc in Page 750. In Page 750 deja vedem ca keys intre 44197 si 44244 se gasesc in Page 815 si Page 815 deja e Data Page cu datele noastre chiar. Asa, nu a trebui parcursa inutil fiecare Data Page cu fiecare tuple din ea ca sa se caute key 44242.\

* Caracteristici pentru clustered index:

- Primary key trebuie sa fie cat mai mic, 4 bits de ex. Primary Key oricum va fi pusa in IndexPage, si cu cat sunt mai mici in volum, cu atat mai multe pot fi puse in Index Page si cu mai putin spatiu ocupa, deci si mai putine Index Pages vor fi create.

- Key sa fie unique, dar primary key si asa e unique deja

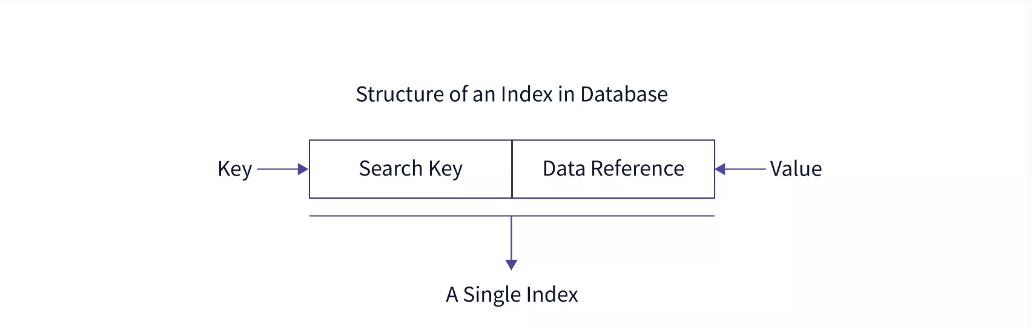
* Cand cream un clustered index, se creaza in aceesi zona de memorie ca si tabelul o structura de date, un **B-Tree(Balanced Tree, nu Binary!)**in care se pastreaza ordinea in care ar trebui sa fie datele sortate in mod normal in tabel
* Deci, doar OS e cel care decide unde anume in Memorie se pastreaza datele. DBMS nu are control total asupra memoriei, si nu poate face ca datele sa fie stocate in mod sortat
* **Mereu se creaza automat un clustered index pentru primary key si nu il putem crea manual!**

[An in-depth look at Database Indexing (freecodecamp.org)](https://www.freecodecamp.org/news/database-indexing-at-a-glance-bb50809d48bd/)

[Effective Clustered Indexes - Simple Talk (red-gate.com)](https://www.red-gate.com/simple-talk/databases/sql-server/learn/effective-clustered-indexes/)

**Non-clustered Index**

* Sunt indexes care creaza un tabel separat pentru a ordona datele.
* El nu ordoneaza direct datele in tabel, ci creaza un nou tabel, pe baza la atributul la care am creat index, si il sorteaza in acel tabel. El va avea in acel tabel inca o coloana, ce va fi referinta catre tuple din tabelul original.
* **Index** e format din 2 coloane: key si referinta



**Search key** – este primary sau candidate key aleasa sa fie indexata

**Data Reference** – pointer ce stocheaza adresa de memorie la intregul tuple a carui key data e indexata

* Sintaxa de creare:

CREATE INDEX index\_name

ON table\_name(column-name\_1, column-name\_2, ...);

* Un nonclustered index e creat cand mai avem nevoei sa facem cautari prin tabel nu doar dupa primary key, si cream nonclustered index anume pentru coloanele pe care le vom folosi.
* Are sens sa cream index doar daca in tabel sunt foarte multe date, de altfel timpul de a crea index va lua mai mult ca cautarea liniara a datelor.
* Este o idee rea sa indexam coloanele ce vor contien date care se repeta, sau date care oricum nu vor fi folosite pentru cautare, gen numele, prenumele etc. Asta va avea consecinte foarte mari asupra la viteza si nu ne va ajuta prea mult. Index sunt bune pentru coloanele ce au unique.
* Daca mai folosim si unique la o coloana pe care o indexam, cautarea va fi si mai rapida, asa cum odata ce o data a fost gasita, cautarea se va opri

Tabel din baza de date mereu e stocat in mai multe fisiere, daca e mare, si deci