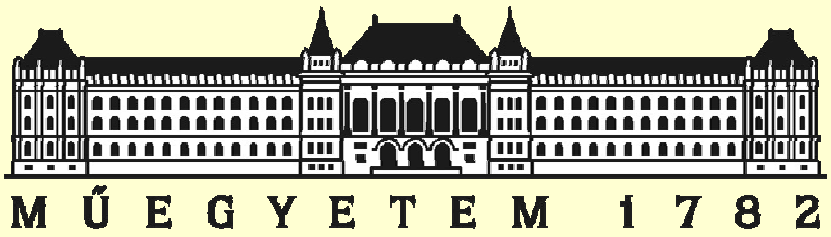
Indexek és SQL hangolás

Ableda Péter
abledapeter@gmail.com



Adatbázisok haladóknak 2012.

2012. november 20.

Miről lesz szó?

- Történelem
- Oracle B*-fa Index
  - Felépítése, karbantartása, típusai
- Bitmap index
- Index Organized Table
- Domain index
- Betekintés az SQL hangolásba

Kérdések, válaszok

Az indexkezelés története

- A hierarchikus adattárolás után a relációs adattárolás jelentős teljesítménybeli visszalépést jelentett
- IBM (DB2) – Clustering (hierarchikus elrendezést örökli – hatékony lekérdezés, rossz DML)
- Ingress – Hashing (Elsődleges kulcs transzformációja – hatékony, de általában kevés)
- Oracle – Indexing

B*-fa

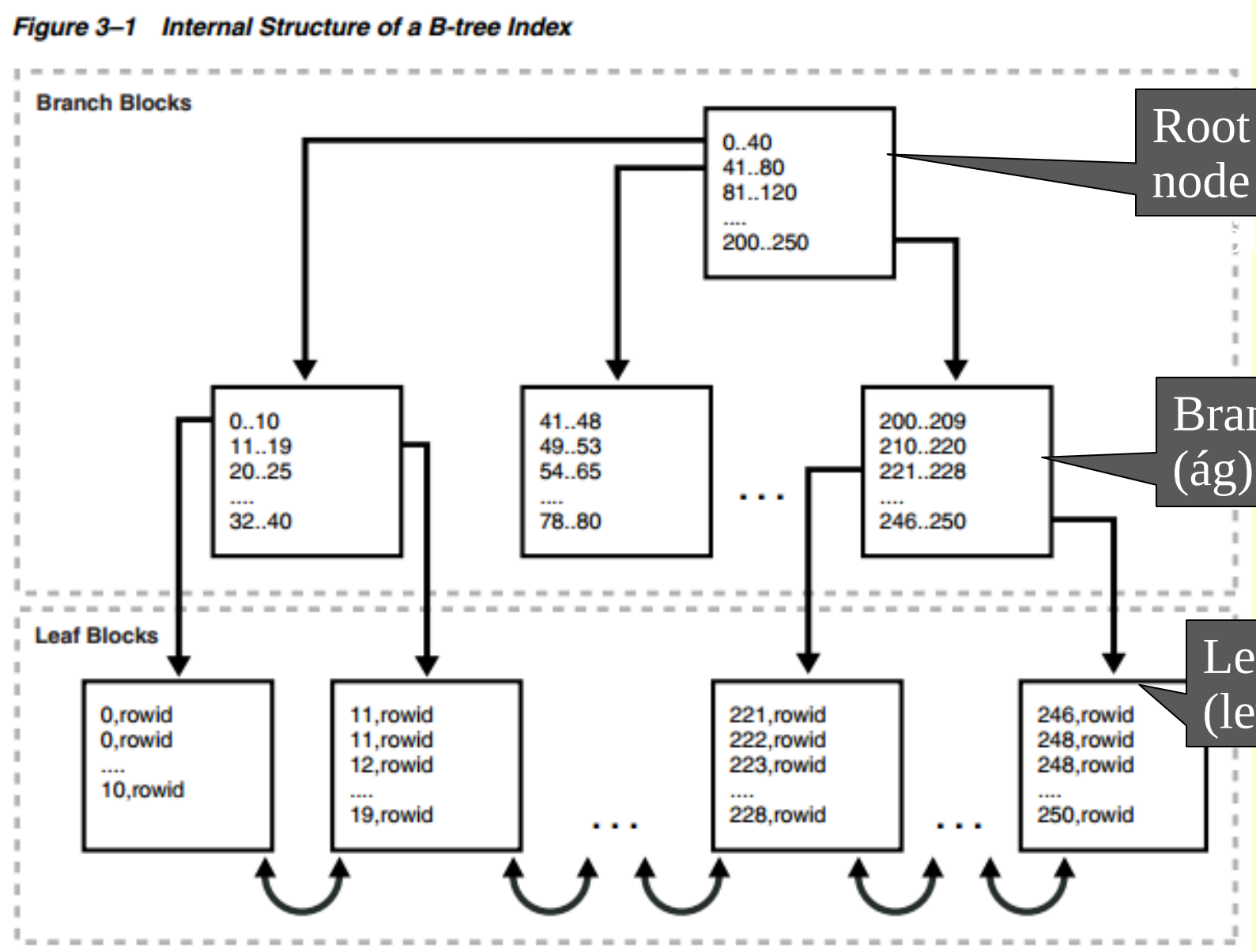


Figure 3–1 Internal Structure of a B-tree Index
Branch Blocks
0..40
41..80
81..120
....
200..250
0..10
11..19
20..25
....
32..40
41..48
49..53
54..65
....
78..80
200..209
210..220
221..228
....
246..250
Leaf Blocks
0,rowid
0,rowid
....
10,rowid
11,rowid
11,rowid
12,rowid
....
19,rowid
221,rowid
222,rowid
223,rowid
....
228,rowid
246,rowid
248,rowid
248,rowid
....
250,rowid
Root node
Branch node (ág)
Leaf node (levél)

B*-fa indexek karbantartása

- **Insert**:
  - Levél blokk megtelik:
    - Vágás (Split) - 50%-50% arányban
    - Túlcsordulás (Overflow) - 90%-10% arányban
- **Delete**:
  - Csak logikai törlés (flag beállítása)
  - Újraépítés (Rebuild Index)
- **Update**:
  - ~Delete+Insert

Indexek csoportosítása (Unique/Non-Unique)

- Create index/Create (unique) index – automatikus
- Fizikai tárolásban nincs különbség közöttük
- Unique
  - Biztosítja az egyediséget (több NULL is lehet)
  - Első találat – találat
  - Gyorsabb keresés

Indexek csoportosítása (Single / Concatenated)

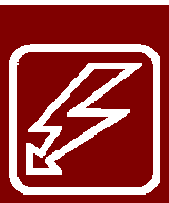
- Single (egyoszlopos)
  - Automatikusan létrejön a Primary key és Unique oszlopokra
- Null értékek kezelése
  - A null értékek nem kerülnek be az indexbe (több hátrány mint előny)
    - Pl: Sorok számának meghatározásához nem lehet indexet használni (kiv. not null oszlop)

Indexek csoportosítása (Single / Concatenated)

- Concatenated index (többoszlopos vagy kompozit index)
  - Több oszlop egy indexben
  - Fontos a sorrend
    - Csak akkor használható, ha az első néhány oszlop szerepel az utasítás (select, update, delete) where feltételében*
    - A kardinalitás (számosság) szempontjából viszont mindegy
  - Jobb mint több single index használata
    - Gyorsabb lekérdezés
    - Gyorsabb frissítés
  - Túl sok oszlop esetén kevés kulcs fér egy blokkba – magas lesz az index

Indexek csoportosítása (Normal / Reverse key)

- 99%-ban normal
- Reverse key
  - Speciális probléma:
    - Right most index leaf block contention
  - Megoldás: visszafelé olvassuk a számokat, és úgy helyezzük el az indexben.
  - Probléma:
    - Csak egyezőség vizsgálható



Function Based Indexek

- Függvény vagy kifejezés értékét tárolják az indexben
- Példa:
  - ```
    SELECT *
      FROM employees
      WHERE UPPER(first_name) = 'AUDREY'
    ```
  - ```
    SELECT employee_id, 12*salary*commission_pct
      FROM employees
      WHERE (12 * salary * commission_pct) < 30000
    ```

Bitmap index

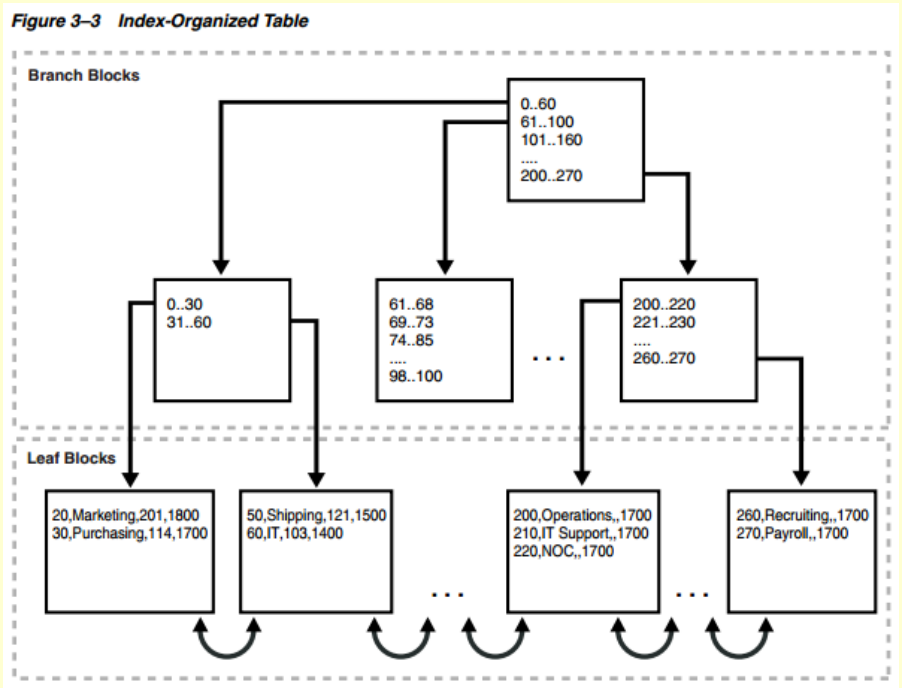
Tábla		Bitmap	
Név	Nem	fiú	lány
Boszorka	lány	0	1
Hapci	fiú	1	0
Hófehérke	lány	0	1
Kuka	fiú	1	0
Morgó	fiú	1	0
Szende	fiú	1	0
Szundi	fiú	1	0
Tudor	fiú	1	0
Vidor	fiú	1	0

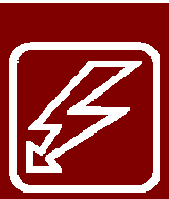
Bitmap index

- Alacsony kardinalitású oszlopokra érdemes
- Read only táblák esetében jó (adattárház)
  - Nagyon hatékonytalan DML műveletek esetén
    - Probléma: Blokk szintű lockolás
  - Gyakorlat: Frissítés helyett inkább Drop-Create
- Nagyon kis helyet foglal
  - Drop-Create esetén
  - 1G tábla – 1M index

Index Organized Table

- Hagyományos táblákban az adatok rendezetlenül vannak
- Index Organized Table:
  - Elsődleges kulcs szerint rendezett B-fában tároljuk az adatokat





Index Organized Table

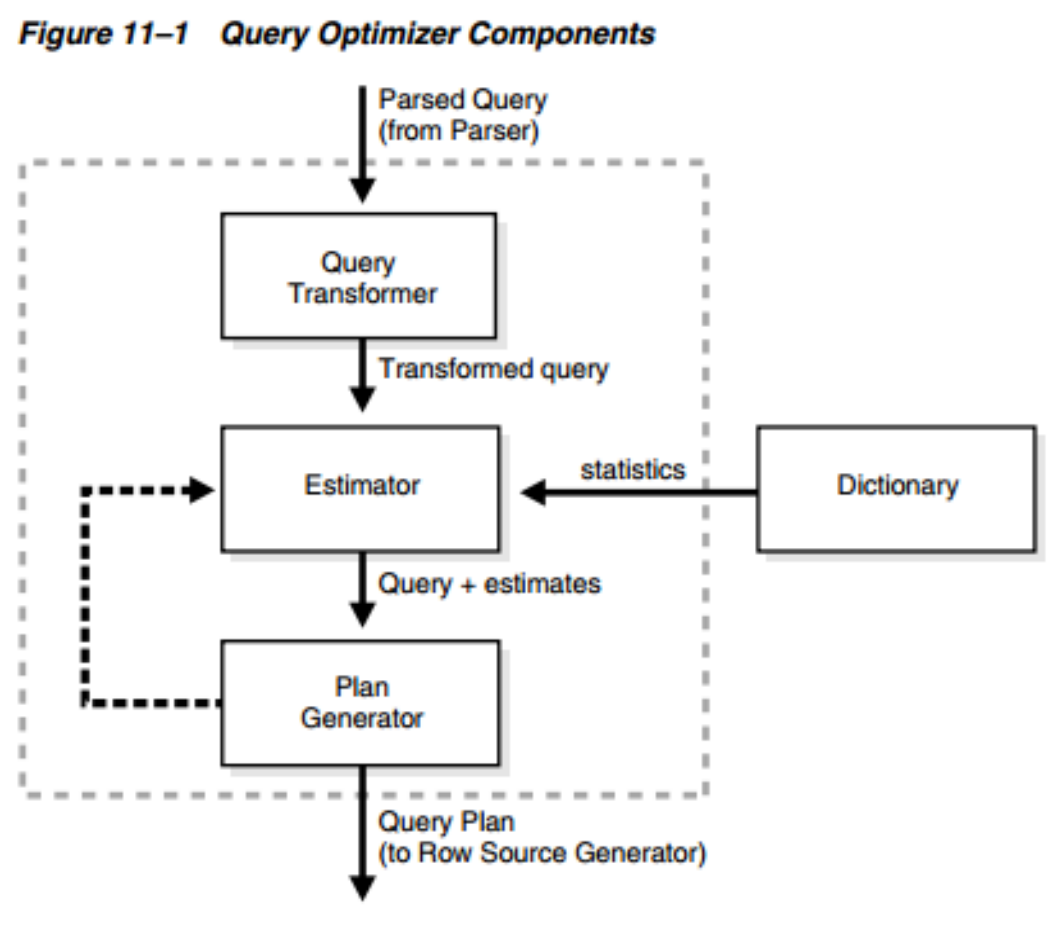
- Primary Key szerinti keresés esetén hatékonyabb – nem kell plusz blokkhozzáférés
- Nem kell külön index az elsődleges kulcsra (kevesebb tárhelyet igényel)
- Másodlagos indexek használata lassabb, mint egyszerű táblák esetében
- Overflow (megoldás a sokszintű fa elkerülésére)
  - Ritkán használt (nem kulcs) oszlopokat ki lehet emelni egy külön kupacba – kisebb B-fa méret

Domain index

- Oracle engedi, hogy saját indextípusokat definiáljunk.
- Alkalmazás-specifikus indexeket hozhatunk létre
- Példa:
  - Virage – Képek, videók indexelése
  - Spatial – Térképadatbázis

Query Optimizer

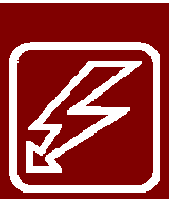
Feladat: Deklaratív kérés átalakítása procedurális utasítások sorozatára



Végrehajtási terv

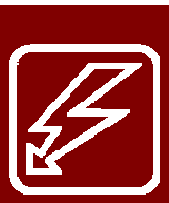
```
SELECT first_name, last_name, salary
  FROM employees
      WHERE email IS NOT NULL
   AND salary > 500
  ORDER BY salary
```

```
Execution Plan
----------------------------------------------------------
Plan hash value: 3447538987

-------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |   107 |  2889 |     4  (25)| 00:00:01 |
|   1 |  SORT ORDER BY     |           |   107 |  2889 |     4  (25)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| EMPLOYEES |   107 |  2889 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter("SALARY">500)
```



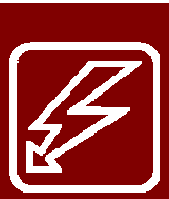
Full Table Scan

- Minden rekordot sorról sorra beolvasunk
- Az adatokon való szekvenciális keresés indexek használata nélkül
- Legegyszerűbb, de leglassabb megoldás
- Használjuk, ha:
  - Nincs index (nem elérhető) a táblán
  - Az összes adatra szükség van
  - Túl kicsi a tábla, nem éri meg indexekhez nyúlni



Full Index Scan

- A teljes indexet végigolvassuk
- Előnye: Nem szükséges az adatok rendezése, hiszen azok az indexben rendezve vannak.



Fast Full Index Scan

- A Full Table Scan alternatívája
- Az indexekből olvassa ki az adatokat, a táblához nem kell hozzáférnie
- Használjuk, ha:
  - Az index a lekérdezés végrehajtásához szükséges minden oszlop értékét tartalmazza

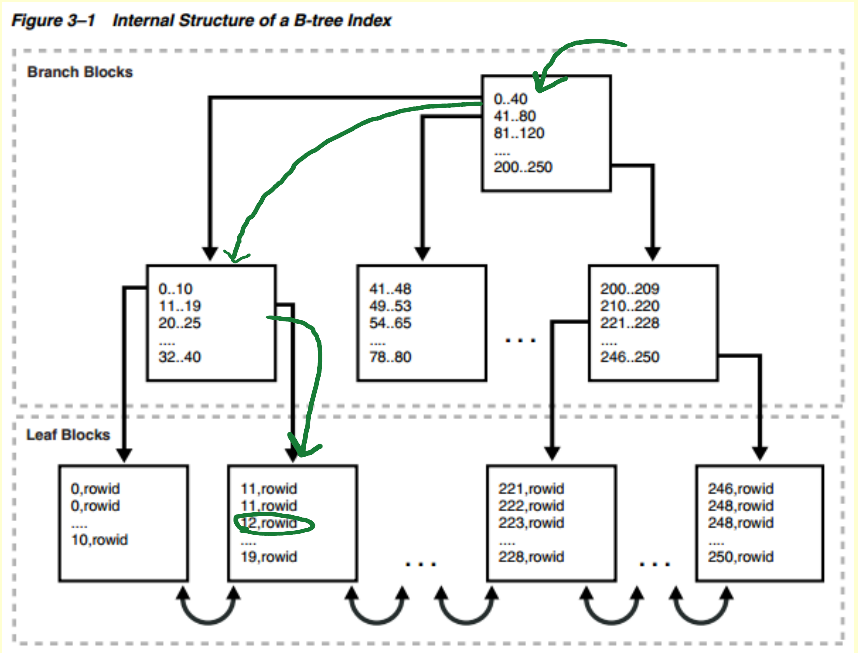
Fast Full Index Scan

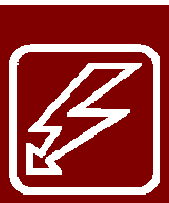
```
SELECT last_name, salary
  FROM employees
```

```
Execution Plan
----------------------------------------------------------
Plan hash value: 225593660

-------------------------------------------------------------------------------------------------
| Id  | Operation        | Name                          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                               |   107 |  1284 |     1   (0)| 00:00:01 |
|   1 |  INDEX FULL SCAN | EMP_DEPTID_LASTNAME_SALARY_IX |   107 |  1284 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

Index Unique Scan

- Egy (vagy 0) sort kapunk vissza
- Unique index kell hozzá
- Használható:
  - Ha létezik index a WHERE feltételben lévő oszlopra





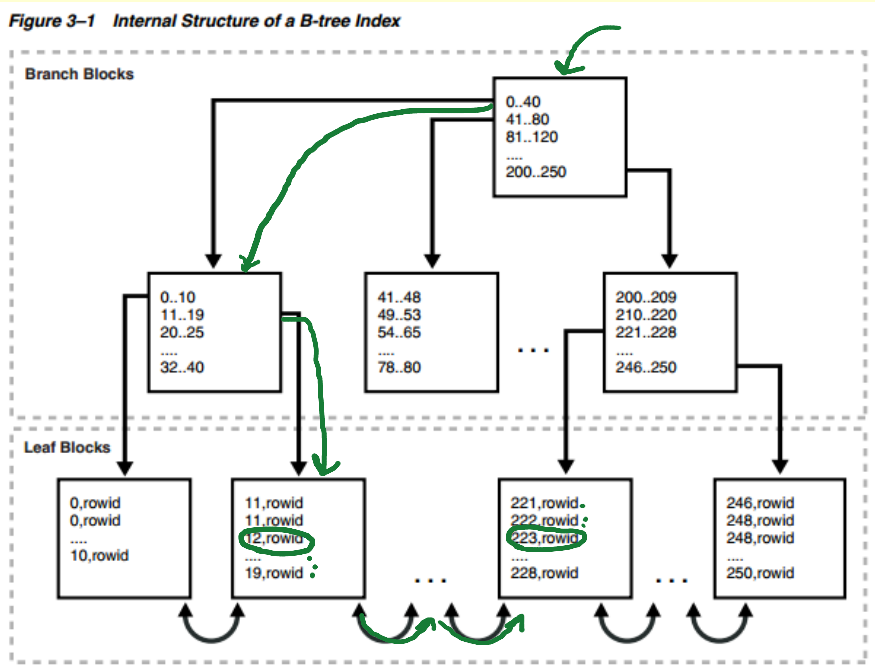
Index Unique Scan

```
SELECT *
  FROM employees
  WHERE employee_id = 5
```

```
Execution Plan
----------------------------------------------------------
Plan hash value: 1833546154

-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |               |     1 |    69 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMPLOYEES     |     1 |    69 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN         | EMP_EMP_ID_PK |     1 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("EMPLOYEE_ID"=5)
```

Index Range Scan

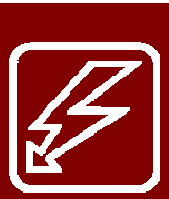
- 0, 1, vagy több sort ad vissza
- Használjuk, ha:
  - A Where feltétel valamilyen összehasonlító operátort használ



Index Range Scan

```
SELECT *
  FROM employees
  WHERE last_name LIKE 'A%'
```

```
Execution Plan
----------------------------------------------------------
Plan hash value: 2077747057

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |     3 |   207 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMPLOYEES   |     3 |   207 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | EMP_NAME_IX |     3 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("LAST_NAME" LIKE 'A%')
       filter("LAST_NAME" LIKE 'A%')
```



Index Skip Scan

Composite index – „Csak akkor használható, ha az első néhány oszlop szerepel a query where feltételében*”

- Fel tudjuk használni az indexet akkor is, ha nem tartalmazza a lekérdezés az első néhány oszlopot
- Csak akkor hatékony, ha a vezető oszlop kardinalitása alacsony

Index Skip Scan

- Elérhető egy (*cust_gender*, *cust_email*) oszlopokból álló composite index.

```
SELECT *
     FROM sh.customers
     WHERE cust_email = 'Abbey@company.com'
```

```
Execution Plan
----------------------------------------------------------
Plan hash value: 1287876719

-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                         |    33 |  5973 |    10   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| CUSTOMERS               |    33 |  5973 |    10   (0)| 00:00:01 |
|*  2 |   INDEX SKIP SCAN            | CUSTOMERS_GENDER_EMAIL |    33 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("CUST_EMAIL"='Abbey@company.com')
       filter("CUST_EMAIL"='Abbey@company.com')
```

Index Skip Scan

```
SELECT *
  FROM sh.customers
  WHERE cust_email = 'Abbey@company.com'
```

```
SELECT *
  FROM sh.customers
  WHERE cust_gender = 'F'
    AND cust_email = 'Abbey@company.com'
UNION ALL
SELECT *
  FROM sh.customers
  WHERE cust_gender = 'M'
    AND cust_email = 'Abbey@company.com'
```

```
Execution Plan
----------------------------
Plan hash value: 128787671

----------------------------
| Id  | Operation
----------------------------
|   0 | SELECT STATEMENT
|   1 |  TABLE ACCESS BY I
|*  2 |   INDEX SKIP SCAN
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("CUST_EMAIL"='Abbey@company.com')
       filter("CUST_EMAIL"='Abbey@company.com')
```

Összefoglalás

- Cél: Az adatbázis teljesítményének növelése (mindegy, hogy milyen eszközökkel)
  - Lekérdezések gyorsítása
  - Karbantartási költség minimalizálása
- Sok index / Kevés index – „Csak ésszel”
  - Ökölszabály: 1 table insert – 2 index insert
    - Pl: 10 index/tábla –> (1 + 20) egységnyi költség

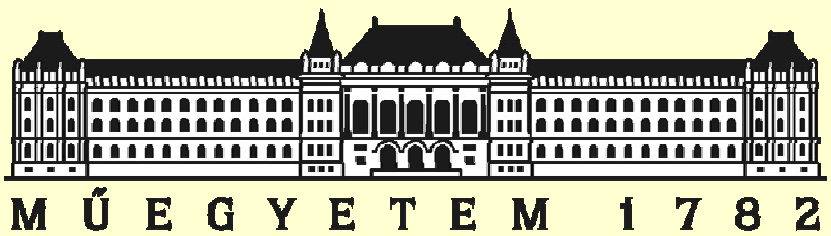
Kérdések…

- …válaszok (?)

Indexek @ Adatb haladóknak

Köszönöm a figyelmet!

Ableda Péter
abledapeter@gmail.com



Adatbázisok haladóknak 2012.

2012. szeptember 18.