Adatbázisok II
Gyakorlat jegyzet

Készült Brányi László gyakorlatai
és Nikovits Tibor jegyzetei alapján

Sárközi Gergő, 2022-23-1. félév
Nincsen lektorálva!

Tartalomjegyzék

1. Bevezetés

- Oracle-t tanulunk (vannak esetek, amikor eltér pl. PostgreSQL-től)
- Hivatalos dokumentáció:
  https://docs.oracle.com/en/database/oracle/oracle-database/12.2/refrn/index.html
  - De rá Google-zni pl. rendszertáblára általában egyszerűbb, gyorsabb

2. Vegyes jegyzet, nem túl hasznos

2.1. Vegyes infók

- Továbbra is igaz: jó megoldás sokszor az összes lehetőség mínusz rosszak
- Table "elsődleges kulcs": tábla tulajdonosa + tábla neve
- Oracle minden szám típusból vagy `NUMBER`-t vagy `FLOAT`-ot csinál
  - `int = number(*,0)`, azaz `precision=NULL`, `scale=0`
- Ha PLSQL definíciók előtt és mögött `/` jel van, akkor az SqlDeveloper is felfogja, hogy hol van vége a definíciónak

2.2. Kis-, nagybetű érzékenység

- `'` között számít a kis- és nagybetű, egyébként nem
- Azaz lehet kisbetűs táblát létrehozni, erre vigyázni kell
- Oracle: alapból nagybetűs minden (talán 1-2 kivétellel)

2.3. NULL-nal kapcsolatos érdekességek

- Unique oszlopba több NULL kerülhet: NULL nem egyenlő önmagával
- `SELECT DISTINCT` csak egy NULL értéket hagy meg
- `intersect`, stb nem mindig működik (pl. `NULL` miatt), ezért először érdemes a szükséges oszlopokra szűrni

2.4. Oszlop default érték

- Beszúrás: NULL nem lesz felülírva DEFAULT érték által
- Oszlop: default 1+1 esetén `1+1` ki lesz értékelve minden default beszúrásnál
- `insert into test values (...)` esetén nem működik a default
  - Helyesen: `insert into test (...)  values (...)`

2.5. Nézettábla, view

- Készítés: `create (materialized) view <név> as <lekérdezés>`
- Készítésnél a `select *`-ban a `*` helyett oszlop felsorolás van mentve

2.6. Szinonima, synonym

- Készítés: `create synonym <név> for <tulaj>.<tábla>`

2.7. Adatbázis link

- `drop database link ullman_link;`
- `create database link ullman_link connect to username identified by password using 'ullman';`
- `select * from NIKOVITS.VILAG_ORSZAGAI@ullman_link;`

2.8. Virtuális oszlopok

- `level`: ha `select connect by` van használva (fabejárás)
- `rownum`: kimenetben sorszámozás
  - pl. `select rownum, emp.* from nikovits.emp`
- `rowid`: sor pozícióját adja meg
  - Base64: A-Z=0-25, a-z=26-51, 0-9=52-61, '+'=62, '/'=63
  - 18 karakteres a formátum: OOOOOOFFFBBBBBBRRR
  - OOOOOO: adatobjektum azonosító
    - `substr(rowid, 1, 6) objekt_azon`
    - `dbms_rowid.rowid_object(rowid) objekt_azon`
  - FFF: fájl azonosító (táblatéren belül relatív sorszám)
    - `substr(rowid, 7, 3) fajl_azon`
    - `dbms_rowid.rowid_relative_fno(rowid) fajl_azon`
  - BBBBBB: blokk azonosító (fájlon belüli sorszám)
    - `substr(rowid, 10, 6) blokk_azon`
    - `dbms_rowid.rowid_block_number(rowid) blokk_azon`
  - RRR: sor azonosító (blokkon belüli sorszám)
    - `substr(rowid, 16, 3) sor_azon`
    - `dbms_rowid.rowid_row_number(rowid) sor_azon`

2.9. Kimenet sorainak számának korlátozása

- Konstans sor szám akkor is, ha azonos "értékű" sorok vannak
  - Azaz ezek nem feltétlenül helyes megoldások min/max keresésre
  - A) `... FETCH FIRST n ROWS ONLY;`
  - B) Két query: belsőben `ORDER BY`, külsőben `WHERE rownum<n`
- Minimum/maximum: `WHERE score = (SELECT MAX(score) FROM ...)`
- Min/max példáján `=` helyett `IN` és `MAX` helyett N legnagyobb értékű sor visszaadásával "X legnagyobb értékű sor" típusú feladat megoldható

2.10. Fizikai és logikai tárolási egységek

- Logikai: séma, adatbázis, táblatér, szegmens, extens, adatblokk
  - Ez a tartalmazási sorrend is
  - A tábla is egy szegmens
  - Extens: fájlon belüli szomszédos adatblokkok
  - Szegmens: adatblokkokból álló tárolandó objektum
    - Pl. nézettáblához nem tartozik szegmens (sem object_id)
- Fizikai: fájl, fájlon belül blokkok
- Logikai táblaterek tartalmaznak fizikai fájlokat
- Fizikai fájlok tartalmaznak logikai extenseket és fizikai blokkokat
- Logikai adatblokk ekvivalens a fizikai adatblokkal

2.10.1. Szegmens, blokk, stb. tudnivalók

- Egy blokk lehet szegmenshez rendelve anélkül, hogy lenne benne adat

2.11. Új tábla létrehozás, tábla konfigurálás

- Ezek `create table <név> ...` vagy `alter table <név> ...` után
- `tablespace <név>`: hova tartozzon a tábla
- `storage (initial 128K)`: alapból 128Kb legyen lefoglalva
  - `allocate extent (size 128K)`: utólagos bővítés
- `pctfree 20`: fejléc és adatok közötti üres hely blokkokban; ez lesz kihasználva, ha pl. bejön új oszlop (fejléc az elejéről, adat a végéről van írva, így az üres hely középen van)
- `pctused 40`: blokkot legalább ennyire ki kéne használni

2.12. Indexek

- Létrehozás: `CREATE [UNIQUE/BITMAP] INDEX emp_ind on emp(empno);`
  - Több oszlop, függvény: ...`on emp (job, SUBSTR(ename, 2, 2))`
- Unique index is "constraint nem teljesül" hibaüzenetet adhat
- Index típusa
  - Több oszlop $\Longrightarrow$ lehet még normál
  - Több oszlop, de az egyik DESC $\Longrightarrow$ function
    - Ezért láthatunk mezőt `dba_ind_expressions`-ben
  - function van indexben $\Longrightarrow$ function
    - Ilyenkor az index oszlop neve ilyesmi: `NC000####`

2.12.1. Index szervezett tábla (Index Organized Table, IOT)

- ```
  CREATE TABLE cikk_iot (
      ckod integer,
      cnev varchar2(20),
      szin varchar2(15),
      suly float,
      CONSTRAINT cikk_iot_pk PRIMARY KEY (ckod)
  ) ORGANIZATION INDEX
  PCTTHRESHOLD 20 INCLUDING cnev --Opcionális
  OVERFLOW TABLESPACE users; --Opcionális
  ```
- Új index hozzáadása meglévő IOT táblához:
  `CREATE INDEX CIKK_IOT_SZIN on cikk_iot(szin);`
- Táblatér: IOT esetén NULL, IOT overflow esetén viszont van
- A táblához nem tartozik szegmens, csak az indexhez
  ```

2.13. Particionált táblák

- `rowid` alapján meg lehet mondani, hogy melyik partícióban van az adat
- `SUBPARTITION` alatt már nincsen réteg
- A `OWNER`, `SEGMENT_NAME` több szegmenst is azonosíthat, ezért egyedi azonosítóhoz hozzájuk kell venni `PARTITION_NAME`-t is
  - `SELECT partition_name, bytes FROM dba_segments WHERE owner='NIKOVITS' AND segment_name='ELADASOK';`
  - Azaz szegmens neve a tábla neve, de adatokat a partíciók tárolják
- ```
  --Particionálás tartományok alapján
  CREATE TABLE eladasok (
      szla_szam NUMBER(5), szla_nev CHAR(30),
      mennyiseg NUMBER(6), het INTEGER
  ) PARTITION BY RANGE (het) (
    PARTITION negyedev1  VALUES LESS THAN (13) SEGMENT CREATION IMMEDIATE
      STORAGE(INITIAL 8K NEXT 8K) TABLESPACE users,
    PARTITION negyedev2  VALUES LESS THAN (26) SEGMENT CREATION IMMEDIATE
      STORAGE(INITIAL 8K NEXT 8K) TABLESPACE example,
    PARTITION negyedev3  VALUES LESS THAN (39) SEGMENT CREATION IMMEDIATE
      STORAGE(INITIAL 8K NEXT 8K) TABLESPACE users,
    PARTITION negyedev4  VALUES LESS THAN (MAXVALUE) SEGMENT CREATION
      IMMEDIATE STORAGE(INITIAL 8K NEXT 8K) TABLESPACE users
  );
  ```
- ```
  --Particionálás hash kulcs alapján és hash alapú alparticionálás
  CREATE TABLE eladasok2 (
    szla_szam NUMBER(5), szla_nev CHAR(30),
    mennyiseg NUMBER(6), het INTEGER
  ) PARTITION BY HASH ( het )
      SUBPARTITION BY HASH (mennyiseg) SUBPARTITIONS 3 --opcionális
  ( PARTITION part1 SEGMENT CREATION IMMEDIATE TABLESPACE users,
    PARTITION part2 SEGMENT CREATION IMMEDIATE TABLESPACE example,
    PARTITION part3 SEGMENT CREATION IMMEDIATE TABLESPACE users
  );
  ```

- --Particionálás lista alapján és minta list alapú alparticionálás
```
CREATE TABLE eladasok3 (
  szla_szam NUMBER(5), szla_nev CHAR(30),
  mennyiseg NUMBER(6), het INTEGER
) PARTITION BY LIST ( het )
    SUBPARTITION BY LIST (het) SUBPARTITION TEMPLATE
    ( SUBPARTITION lista VALUES(1,2,3,4,5),
      SUBPARTITION other VALUES(DEFAULT)    )
( PARTITION part1 VALUES(1,2,3,4,5)   SEGMENT CREATION IMMEDIATE
  STORAGE(INITIAL 8K NEXT 8K) TABLESPACE users,
  PARTITION part2 VALUES(6,7,8,9)     SEGMENT CREATION IMMEDIATE
  STORAGE(INITIAL 8K NEXT 8K) TABLESPACE example,
  PARTITION part3 VALUES(10,11,12,13) SEGMENT CREATION IMMEDIATE
  STORAGE(INITIAL 8K NEXT 8K) TABLESPACE users
);
```

2.14. Klaszter táblák

- Eltérő tábláknak egyes sorainak azonos `rowid`-je lesz (klaszter lényege)
- Eltérő `object_id`-hez tartozhat megegyező `data_object_id`
- Csak klaszternek van szegmense (tábláknak nincs)
- Index klaszter létrehozása: (index létrehozása nélkül nem lehet beszúrni)
```
CREATE CLUSTER perspnnel_cl (dep_num NUMBER(2)) SIZE 4K;
CREATE INDEX personnel_cl_idx ON CLUSTER personnel_cl;
```
- Hash klaszter: `CREATE CLUSTER cikk_hcl (ckod NUMBER) [...]`
  - Automatikus hash: ...`[SIZE 4K] [SINGLE TABLE] HASHKEYS 30;`
  - Saját hash: ...`HASHKEYS 30 HASH IS MOD(ckod, 53);`
- Táblák létrehozása klaszteren:

```
CREATE TABLE emp_clt (
  empno NUMBER PRIMARY KEY, ename VARCHAR2(30), job VARCHAR2(27),
  mgr NUMBER(4), hiredate DATE, sal NUMBER(7,2), comm NUMBER(7,2),
  deptno NUMBER(2) NOT NULL
) CLUSTER personnel_cl (deptno);
CREATE TABLE dept_clt (
  deptno NUMBER(2), dname VARCHAR2(42), loc VARCHAR2(39)
) CLUSTER personnel_cl (deptno);
```

3. Nevezetes rendszertáblák

- `DBA_...` helyett írható más prefix is:
  - `DBA`: rendszergazda által elérhető valamik
  - `ALL`: mindenki által elérhető valamik
    - Dokumentáció szerint: általam elérhető (gyakon mást mondtak)
  - `USER`: általam birtokolt valamik, ilyenkor nincs `owner` oszlop
- `DBA_OBJECTS`: objektumok (amik akár 0 helyet foglalnak)
  - `OWNER`: adatobjektum (szegmens) birtokosa
  - `DATA_OBJECT_ID`: adatobjektum (szegmens) azonosítója
    - Pl. nézettábla esetén NULL
  - `OBJECT_ID`: egyedi azonosító; `OBJECT_NAME`: név
  - `SUBOBJECT_NAME`: összetett objektumnál (pl. particionált tábla)
  - `OBJECT_TYPE`: pl.: TABLE, INDEX, VIEW, stb.
  - `CREATED`: létrehozás dátuma
- `DBA_TABLES`
  - `OWNER`, `TABLE_NAME`, `TABLESPACE_NAME`
  - `CLUSTER_OWNER`, `CLUSTER_NAME`: tábla melyik klaszteren van / NULL
  - `NUM_ROWS`: táblának sorainak BECSÜLT száma
  - `BLOCKS`: tábla által foglalt blokkok BECSÜLT száma
  - `IOT_TYPE`: IOT vagy IOT_OVERFLOW vagy NULL
  - `IOT_NAME`: `IOT_OVERFLOW` esetén IOT neve
  - `PARTITIONED`: YES vagy NO

- DBA_TAB_COLUMNS
  - OWNER, COLUMN_NAME
  - COLUMN_NAME, COLUMN_ID: oszlop név és sorsszám
  - TABLE_NAME: tábla/nézet/klaszter neve
  - DATA_TYPE: oszlop adattípusa
  - DATA_LENGTH: adattípus hossza, pl. CHAR(10) esetén 10
  - DATA_PRECISION, DATA_SCALE: NUMBER(10,2) esetén 10 és 2
    - Azaz a precision az értékes jegyek, a scale a tizedesek száma
  - NULLABLE: NULL engedélyezett-e, érték: Y/N
  - DATA_DEFAULT: DEFAULT érték, ha van
  - NUM_DISTINCT: oszlopban lévő különböző értékek BECSÜLT száma
- DBA_SYNONYMS
  - OWNER, SYNONYM_NAME
  - TABLE_OWNER, TABLE_NAME: tábla/nézet, amire a szinonima mutat
  - DB_LINK: lokális tábla: NULL, remote: adatbázis-kapcsoló neve
- DBA_VIEWS
  - OWNER, VIEW_NAME
  - TEXT: nézet lekérdezésének a szövege
- DBA_SEQUENCES
  - SEQUENCE_OWNER, SEQUENCE_NAME
  - MIN_VALUE, MAX_VALUE, INCREMENT_BY
- DBA_DB_LINKS
  - OWNER, DB_LINK
  - HOST, USERNAME

- DBA_DATA_FILES
  - FILE_NAME: operációs rendszerbeli fájlnév
  - FILE_ID: adatbázis rendszerben fájl azonosítója
  - RELATIVE_FNO: táblatéren belüli egyedi fájl azonosító
  - TABLESPACE_NAME: melyik táblatérhez tartozik a fájl
  - BYTES, BLOCKS: fájl jelenlegi mérete
  - AUTOEXTENSIBLE: automatikusan növelheti-e a fájlt az OS
  - MAXBYTES, MAXBLOCKS: maximális méret, amire növekedhet a fájl
- DBA_TEMP_FILES: temporális táblatérhez tartozó adatfájlok
  - Szerkezet: azonos DBA_DATA_FILES-zal
- DBA_TABLESPACES: táblatér az adatfájlok logikai csoportja
  - TABLESPACE_NAME: táblatér neve
  - BLOCK_SIZE: adatblokkok mérete a táblatéren
  - STATUS: ONLINE vagy OFFLINE
  - CONTENTS: PERMANENT vagy UNDO vagy TEMPORARY
- DBA_SEGMENTS: szegmens egy adatblokkokból álló tárolandó objektum
  - OWNER, SEGMENT_NAME (pl. tábla tulajdonosa és neve)
  - SEGMENT_TYPE: pl. TABLE, INDEX, TABLE_PARTITION, CLUSTER
  - TABLESPACE_NAME: melyik táblatéren van a szegmens tárolva
  - PARTITION_NAME: particionált tábla esetén, egyébként NULL
  - BYTES, BLOCKS: szegmens mérete
  - EXTENTS: szegmens extenseinek száma
- DBA_EXTENTS: extens a fájlon belüli szomszédos adatblokkokból álló tárolási egység
  - OWNER, SEGMENT_NAME, SEGMENT_TYPE: mint szegmensnél
  - TABLESPACE_NAME, FILE_ID: hol van az extens tárolva
  - EXTENT_ID: extens sorszáma a szegmensen belül
  - BLOCK_ID: extens első blokkjának sorszáma fájlon belül
  - BYTES, BLOCKS: extens mérete

- DBA_FREE_SPACE: fájlon belüli összefüggő szabad terület
  - TABLESPACE_NAME, FILE_ID
  - BLOCK_ID: szabad terület első blokkjának sorszáma fájlon belül
  - BYTES, BLOCKS: szabad terület mérete
- DBA_INDEXES
  - TABLE_OWNER, TABLE_NAME: melyik táblához való az index
  - OWNER: index tulajdonosa (pl. tábla owner vagy adatbázis rendszer)
  - INDEX_NAME: index neve (ez alapján megtalálható a szegmens)
  - INDEX_TYPE: NORMAL, NORMAL/REV, BITMAP, CLUSTER, FUNCTION-BASED NORMAL, IOT - TOP
  - TABLESPACE_NAME: index melyik táblatéren van tárolva
  - UNIQUENESS: előfordulhatnak-e azonos értékek
  - COMPRESSION: ismétlődő értékek hatékonyabb tárolását szolgálja
  - PREFIX_LENGTH: tömörítés hány oszlopnyi legyen
- DBA_IND_COLUMNS
  - TABLE_OWNER, TABLE_NAME, INDEX_OWNER, INDEX_NAME
  - COLUMN_NAME: indexelt oszlop neve
  - COLUMN_POSITION: indexelt oszlopok közül ez az oszlop hányadik
  - DESCEND: csökkenő sorrendben van-e az indexelés
- DBA_IND_EXPRESSIONS: függvényszerű index oszlopok (vagy DESC mező)
  - TABLE_OWNER, TABLE_NAME, INDEX_OWNER, INDEX_NAME
  - COLUMN_EXPRESSION: kifejezés, ami alapján épül fel az index
  - COLUMN_POSITION: kifejezés hányadik az index bejegyzések sorrendjében

- DBA_CLUSTERS
  - CLUSTER_NAME, OWNER
  - CLUSTER_TYPE: INDEX vagy HASH
  - FUNCTION: NULL (index típus), DEFAULT2 vagy "hash expression"
  - HASHKEYS: nem hash klaszter típus esetén 0
  - SINGLE_TABLE: Y vagy N
- DBA_CLU_COLUMNS: táblák oszlopainak megfeleltetése klaszter kulcsnak
  - CLUSTER_NAME, OWNER
  - TABLE_NAME: tábla neve, OWNER azonos (közös)
  - CLU_COLUMN_NAME: klaszter kulcs neve
  - TAB_COLUMN_NAME: klaszter kulccsal egyeztetett tábla oszlop neve
- DBA_CLUSTER_HASH_EXPRESSIONS: hash klaszterek hash függvényei
  - CLUSTER_NAME, OWNER
  - HASH_EXPRESSION: szöveges kifejezés, ami alapján a hash megy (Soha nem NULL: olyankor csak nincsen bejegyzés ebben a táblában)

- DBA_PART_TABLES
  - TABLE_NAME, OWNER
  - PARTITIONING_TYPE: RANGE vagy HASH vagy LIST
  - SUBPARTITIONING_TYPE: NONE vagy fentiek közül egy
  - PARTITION_COUNT, DEF_SUBPARTITION_COUNT: legalább 0
- DBA_PART_INDEXES
- DBA_TAB_PARTITIONS
  - TABLE_NAME, TABLE_OWNER
  - PARTITION_NAME: partíció neve
  - PARTITION_POSITION: hányadik partíciója a táblának
  - COMPOSITE: van-e alpartició
  - SUBPARTITION_COUNT: legalább 0
  - HIGH_VALUE: mi a particionálás határa: szám, lista, MAXVALUE, NULL (hash esetén)
- DBA_IND_PARTITIONS
  - Nem néztük meg, nem volt róla szó sehol
- DBA_TAB_SUBPARTITIONS
  - Szerkezet nagy része: lásd DBA_TAB_PARTITIONS
  - Extra mezők: SUBPARTITION_NAME, SUBPARTITION_POSITION
- DBA_IND_SUBPARTITIONS
  - Nem néztük meg, nem volt róla szó sehol
- DBA_PART_KEY_COLUMNS: mi alapján történik a particionálás
  - NAME, OWNER
  - COLUMN_POSITION, COLUMN_NAME: particionálás X-edik szempontja
- DBA_SUBPART_KEY_COLUMNS: mi alapján történik az alparticionálás
  - Szerkezete megegyezik DBA_PART_KEY_COLUMNS-zal

4. Lekérdezés tervek, hintek

4.1. Lekérdezési terv lekérdezése

- A lekérdezést `EXPLAIN PLAN FOR ...`-ral kell kezdeni
- A lekérdezést végre kell hajtani
- Utolsó terv lekérdezése 'serial' módon:
  `select * from table(dbms_xplan.display());`
  - Extra paraméterek megadhatóak, de jó ez így ZH-ra
- Összes eddigi terv lekérdezése:
  `select * from plan_table order by plan_id, id;`
- `EXPLAIN PLAN` nélkül is le lehet kérdezni a tervet: fejlesztőkörnyezetekben van erre egy dedikált gomb

4.2. Tábla összekapcsolás típusok

- Descartes szorzat, `INNER JOIN ON`: duplikált oszlopokot megtartja
- `NATURAL JOIN`, `INNER JOIN USING`: duplikált oszlopok elvetése
  - `USING` lényege: ne az összes azonos nevű oszlop legyen használva az összekapcsoláshoz

4.3. Hintek megadása

- Hintek befolyásolják, hogy egy művelet hogyan legyen megvalósítva
- `SELECT`, `UPDATE`, `DELETE`, `INSERT` után állhat, belső lekérdezésben is
- Szintaxis: `<utasítás> /*+ tipp1 tipp2 stb */ ...`
  - A `+` jel előtt nincsen szóköz
  - Tippek és tipp paraméterek elválasztója: szóköz
- Hibás hintek figyelmen kívül vannak hagyva (nincs hibaüzenet)
- Tábla megadása hint paraméterül
  - Ha van alias, akkor kötelező az alias-t használni
  - A sémát (tábla tulajdonosát) tilos kiírni

4.4. Fontosabb tippek (hintek) listája

4.4.1. Egyéb, más kategóriába nem illő beállítások

- Optimalizálás típusa: `all_rows` vagy `first_rows(count)`
- Összekapcsolások sorrendje:
  - `ordered`: SQL utasításban található sorrend használata
  - `leading(táblalista)`: listában szereplők legyenek elsők

4.4.2. Elérési útvonal konfigurálása

- `[indexlista]` jelentése: 0 vagy több index felsorolása
- `full(tábla)`: indexek használata helyett full table scan
- `cluster(tábla)`, `hash(tábla)`: csak index/hash klaszteren értelmezett
- `index(tábla [indexlista])`: (felsoroltak közül legolcsóbb) index segítségével legyen a tábla elérve (lista lehet üres)
  - `no_index(tábla [indexlista])`: ne legyenek az indexek használva
- `index_asc(tábla [indexlista])` vagy `index_desc(tábla [indexlista])`: normál vagy fordított sorrendben legyen az index bejárva
- `index_combine(tábla [indexlista])`: bitmap indexek használata
- `index_ffs(tábla [indexlista])`: fast full index scan használata
  - `no_index_ffs(tábla [indexlista])`: ennek a megtiltása
- `index_join(tábla [indexlista])`: több index használata, a sorazonosítók join-olásával legyen elérve a tábla
- `index_equal(tábla [indexlista])`: több index használata, a sorazonosítók metszetével legyen elérve a tábla

4.4.3. SQL transzformációk konfigurálása

- `no_query_transformation`: minden tiltása
- `no_expand`: lekérdezésben található `OR` vagy `IN` mentén lekérdezés több darabra darabolásának megtiltása
- `use_concat`: `OR` feltételekből unió létrehozása (előző ellentéte)

4.4.4. Tábla összekapcsolás konfigurálása

- `táblalista` jelentése: 2 vagy több tábla felsorolása
- Alábbiakhoz létezik értelemszerű `no_` prefixű változat is
- `use_hash(táblalista)`: hash join használata
- `use_nl(táblalista)`: nested loop join használata
- `use_merge(táblalista)`: merge sort join használata
- Belső lekérdezésekben, táblalista nélkül használható:
  - `nl_sj`, `hash_sj`, `merge_sj`: semi join használata
  - `nl_aj`, `hash_aj`, `merge_aj`: anti join használata

4.5. Műveletek (operations), amit a tervben olvashatunk

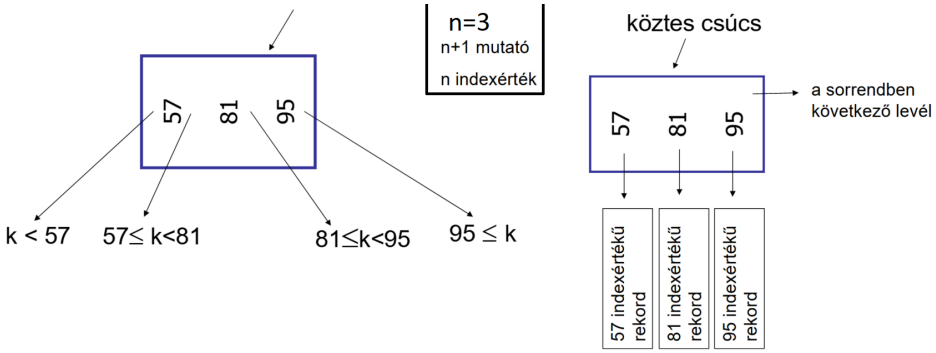
- dr. Nikovits Tibor honlapján található `tervekN.txt` sokat segít: megmutatja, hogy milyen lekérdezés eredményez adott elemeket tartalmazó tervet
- `INLIST ITERATOR`: műveletek ismétlése ciklusban
- `TABLE ACCESS FULL/HASH/CLUSTER`: lásd `full(tábla)` vagy `hash(tábla)` vagy `cluster(tábla)` hint
- `TABLE ACCESS BY INDEX ROWID`: index segítségével tábla elérés
- `PARTITION RANGE ALL/SINGLE/ITERATOR`: összes/1/több partíció olvasása
- `SORT AGGREGATE/UNIQUE/GROUP BY/JOIN/ORDER BY`: adott művelet rendezés segítségével történő megvalósítása
  - Ha `SORT UNIQUE NOSORT` szükséges: próbálkozzunk szöveg oszloppal
- `HASH UNIQUE/GROUP BY`: adott művelet hasítás segítségével történő megvalósítása

  - Ha SORT kéne de a tervben HASH van: csináljuk egy ORDER BY-t
  - Ha HASH kéne de SORT van: csináljunk explicit GROUP BY-t
- UNION-ALL: duplikátumokat megtartó unió
  - UNION megvalósítása: UNION-ALL után DISTINCT
- MINUS, INTERSECTION, CONCATENATION: sorhalmazok műveletei
- AND-EQUAL: sorazonosító halmazok metszetének képezése
- VIEW: ???
- FILTER: sorhalmaz szűrése: WHERE vagy HAVING
- Összekapcsolások
  - NESTED LOOPS
  - MERGE JOIN: rendezett (rész)táblák összefuttatása
  - HASH JOIN [OUTER/ANTI [NA]/SEMI]: hasításos összekapcsolás
    * ANTI: NOT EXISTS-hez vagy NOT IN-hez használható
    * NA: null aware, azaz NULL is előfordulhat
    * SEMI: összekapcsolás után csak egyik tábla adatai kellenek
- Indexekkel kapcsolatos műveletek
  - INDEX FULL SCAN [DESCENDING]: lásd index_desc(tábla [indexlista])
  - INDEX FAST FULL SCAN: lásd index_ffs(tábla [lista]) hint
  - INDEX RANGE SCAN: intervallumos keresés
  - INDEX SKIP SCAN: több oszlopos index olvasása, az első oszlopok ismerete nélkül
  - INDEX UNIQUE SCAN: egyedi értékek keresése index segítségével
- Bitmappel kapcsolatos műveletek
  - BITMAP INDEX SINGLE VALUE: egyetlen bitvektor visszaadása
  - BITMAP AND/OR: bitmapek közötti logikai műveletek
  - BITMAP CONVERSION TO/FROM ROWIDS: sorazonosító-bitmap konverzió
  - BITMAP CONVERSION COUNT: bitmapben az igaz értékek száma

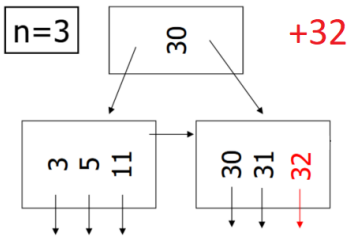
5. Papíros rész

5.1. B+ fa

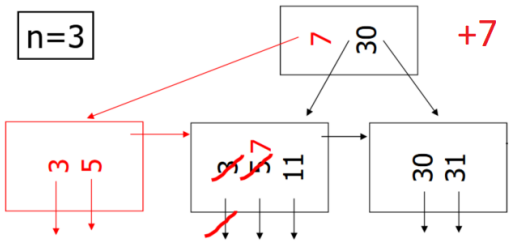
- Minden csúcs legalább 50%-ban telített
  - Leveleknél: legalább $\lfloor \frac{n+1}{2} \rfloor$ mutató blokkra (ugyan ennyi kulcs)
    - Példa: $n = 3 \vee n = 4 \implies \min 2$ kulcs
  - Belső csúcsoknál: legalább $\lfloor \frac{n+1}{2} \rfloor$ mutató csúcsra ($\lfloor \frac{n+1}{2} \rfloor - 1$ kulcs)
    - Példa: $n = 3 \implies \min 1$ kulcs, $n = 4 \implies \min 2$ kulcs
  - Kivétel: gyökér csak legalább 1 mutatót tartalmaz
- Belső csúcsban megtalálható kulcs megtalálható levélben is
  - De egy szám maximum kétszer szerepelhet (különben valami hibás)
- $n = 3$ jelentése: 3 adattag, 4 mutató (levél esetén a 4.: következő levél)
- Szomszédos cellák egyesítése, összevonása:
  - Beszúrásnál nem csinálunk ilyet
  - Törlésnél lehetséges



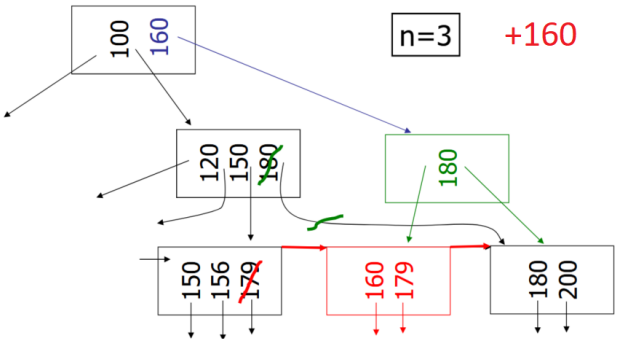
5.1.1. Beszúrás



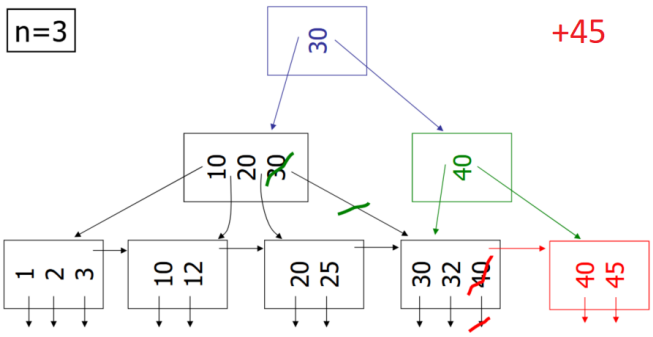
Beszúrás: egyszerű eset



Beszúrás: új levél

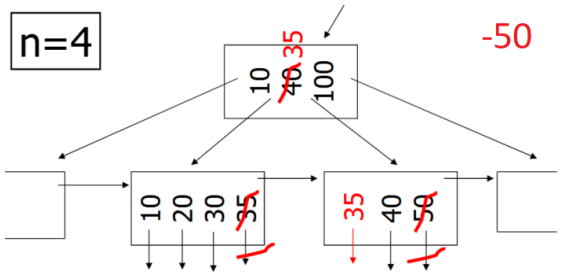


Beszúrás: új gyökér

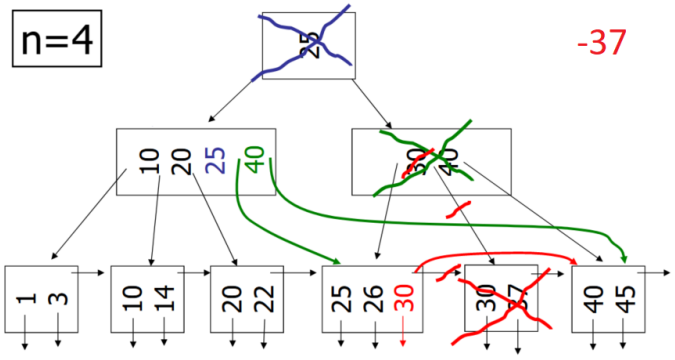


Beszúrás: új gyökér

5.1.2. Törlés



Törlés: egyszerűbb eset

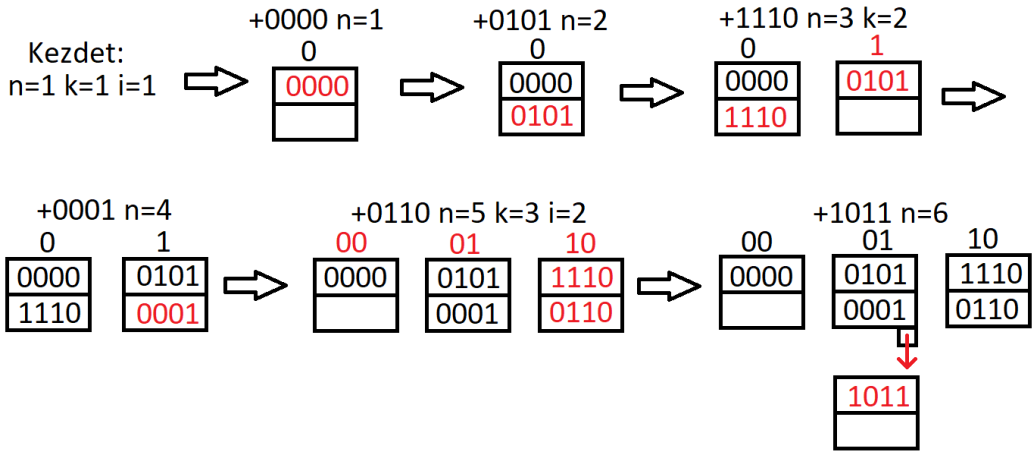


Törlés: bonyolult eset

5.2. Lineáris hasítás

- Változók:
  - n: adatok száma (kezdetben: 0)
  - k: kosarak száma (kezdetben: 1)
  - i: vizsgált bitek száma (kezdetben: 1)
  - m: max kosár sorszám eddig, nem használjuk (kezdetben: 0)
- Konstans: küszöbszám $= 2,4$

- Változók változása
  - beszúrás $\implies n := n + 1$
  - $n/k >$ küszöb $\implies k := k + 1$
  - $k > 2^i \implies i := i + 1$
  - A változást okozó beszúrásban már a változók új értékével dolgozunk
    - pl. ha van új kosár: az új kosarat is figyelembe vesszük
- Kosaraknak van sorszámuk és 2 férőhelyük
  - Megtelnek $\implies$ láncolni kell túlcsordulási blokkot (+2 férőhely)
- Adathoz tartozó kosár: sorszáma a hasító érték utolsó i bitjével egyezik
  - Ha nincs ilyen kosár, akkor hátulról i-edik bitben eltérés megengedett
    - Hátulról i-edik bit a vizsgált bitsorozat első bitje
- Áthelyezés lehet, hogy kell, ha:
  - a) k nőtt: hátulról i-edik bitben eltérő elemeket kell megnézni
  - b) i nőtt: minden kosarat végig kell nézni
  - Először történik az áthelyezés, utána az új érték beszúrása



5.3. Bitmap index

- Olyan oszlopokhoz hatékony, amelyekben kevés különböző érték található
- Bitmap minden sorában pontosan egy '1' szerepel

Empno	**Status**	**Region**	**Gender**
101	single	east	male
102	married	central	female
103	married	west	female
104	divorced	west	male

Tábla, amihez bitmap index lesz

Region= east	Region= central	Region= west
1	0	0
0	1	0
0	0	1
0	0	1

Bitmap index *Region* oszlopra

5.3.1. Lekérdezés bitmap segítségével

- Lekérdezés feltételek logikai műveletekké alakíthatóak
- Lekérdezés: `SELECT COUNT(*) FROM CUSTOMER WHERE STATUS='married' AND REGION IN ('central', 'west');`

status= married		region= central		region= west						
0		0		0		0		0		0
1	AND	1	OR	0	=	1	AND	1	=	1
1		0		1		1		1		1
0		0		1		0		1		0

- Az eredmény az utolsó oszlopban az egyesek száma.

5.4. Szakaszhossz kódolás

- Cél: bitmap-ek tömörítése (ahol ritka az 1-es érték)

5.4.1. Enkódolás, tömörítés

- 1. lépés: minden '1' után vágás, ez alkot egy szakaszt
- 2. lépés: szakaszban lévő '0'-k megszámolása
- 3. lépés: $C :=$ előző érték felírása 2-es számrendszerbe
- 4. lépés: $D :=$ bináris számban helyi értékek megszámolása
- Részeredmény: $D - 1$ db '1' + 1 db '0' + C
  - Teljes eredményhez a többi szakaszt is kódoljuk
  - Részeredményeket írjuk egymás után
- Példa: ...|000001|... $\rightarrow$ 5db '0' $\rightarrow 101_2 \rightarrow$ 3 számjegy $\rightarrow$ 11|0|101
- Ha az utolsó szakasz nem '1'-re végződik: nem tároljuk a záró '0'-kat
- Két '1' egymás után: nem gond, 00-ként van kódolva

5.4.2. Dekódolás, visszafejtés

- 1. lépés: $A :=$ elején lévő '1' karakterek száma
- 2. lépés: következő karakter ('0') kiolvasása
- 3. lépés: $C := A + 1$ db karakter bináris számként értelmezése
- Részeredmény: C db '0'-t követ egy '1'-es
  - Teljes eredményhez a többi szakaszt is dekódoljuk
  - Részeredményeket írjuk egymás után
- Példa: $in = 110101... \rightarrow$ 2db '1' $\wedge\ in = 0|101|... \rightarrow 5_{10} \rightarrow 000001$

5.4.3. Hosszabb példa

- Eredeti: 0000000000000110001
  - 13, 0, illetve 3 darab '0' található szakaszonként
- Kódolt: 1110**1101**00**1**0**11**
  - '0'-k bináris darabszáma vastag betűvel van szedve

5.5. Naplózás: napló alapján helyreállítás

5.5.1. UNDO naplózás

- Mely tranzakciókat kell vizsgálni, ha ilyen bejegyezéssel találkozunk?
  - <END CKPT> $\Longrightarrow$ <START CKPT(...)> után kezdődőekkel
  - <START CKPT(T1,...)> $\Longrightarrow$ T1 és utána kezdődőekkel
  - Egyik sincs a naplóban $\Longrightarrow$ összes
- Kezelendő tranzakciók: nincsen <T COMMIT>
- Helyreállítás: naplón visszafelé haladunk
  - <T,X,v> $\Longrightarrow$ <T,X,v>
  - <T START> $\Longrightarrow$ <T ABORT>
- Magyarázat:
  - <T COMMIT> a módosítások lemezre írása után egyből kiíródik
  - CKPT(...) során a felsorolt tranzakciók <T COMMIT>-jára várunk
  - <START CKPT(...)>-ban fel nem soroltakhoz már van <T COMMIT>

5.5.2. REDO naplózás

- Mely tranzakciókat kell vizsgálni, ha ilyen bejegyezéssel találkozunk?
  - <END CKPT> $\Longrightarrow$ START-ban felsoroltak közül legkorábbitól
  - Nincs a naplóban $\Longrightarrow$ összes
- Kezelendő tranzakciók: van <T COMMIT> de nincs <T END>
  - Ha nincs <T COMMIT> se, akkor csak <T ABORT> kell
- Helyreállítás:
  - Naplón előrefelé haladunk; <T,X,v> $\Longrightarrow$ <T,X,v>
  - Napló végén minden ilyen T-hez: <T END>
- Magyarázat:
  - <T COMMIT> a módosítások lemezre írása előtt kerül a lemezre
  - CKPT(...) során a fel nem soroltak lemezre írására várunk
    - Felsoroltakhoz még nincsen <T COMMIT>, többihez van

5.5.3. UNDO/REDO naplózás

- Az UNDO és a REDO lépés független tudtommal: nincs sorrend köztük
- UNDO:
  - Elég ezeket a tranzakciókat vizsgálni:
    - <END CKPT> $\Longrightarrow$ <START CKPT(...)> után kezdődőekkel
    - <START CKPT(T1,...)> $\Longrightarrow$ T1 és utána kezdődőekkel
    - Nincs a naplóban $\Longrightarrow$ összes
  - Kezelendő tranzakciók: nincs <T COMMIT>
  - Naplón visszafelé haladunk
    - <T,X,v,w> $\Longrightarrow$ <T,X,v>
    - <T START> $\Longrightarrow$ <T ABORT>
- REDO:
  - Elég ezeket a tranzakciókat vizsgálni:
    - <END CKPT> $\Longrightarrow$ START-ban felsoroltak közül legkorábbitól
    - Nincs a naplóban $\Longrightarrow$ összes
  - Kezelendő tranzakciók: van <T COMMIT>, nincs <T END>
  - Naplón előrefelé haladunk:
    - <T,X,v,w> $\Longrightarrow$ <T,X,w>
  - Napló végén minden ilyen T-hez: <T END>
- Magyarázat:
  - <T COMMIT> kiadható lemezre írás előtt és után is
  - CKPT(...) során az összes módosítás lemezre írására várunk
  - CKPT(...) felsoroltjaihoz nincsen <T COMMIT>, többihez van

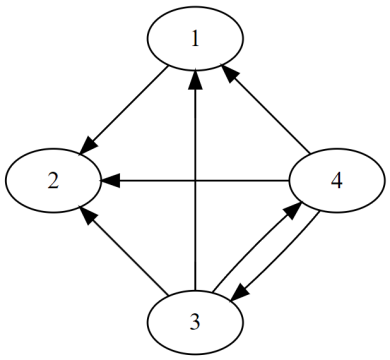
5.5.4. Appendix

Amikor ez a jegyzet azt mondja, hogy `<T,X,v>`-t kell a ZH-n a lapra írni, akkor a valóságban ezt kéne csinálni: `WRITE(X,v); OUTPUT(X)`. Legalábbis az előadás diák szerint.

5.6. Megelőzési gráf

- T_i-ből mutat nyíl T_j-be, ha T_i-nek meg kell előzni T_j-t
  - Konfliktus-sorbarendezhető az ütemezés, ha a gráf körmentes
- Táblázatot készítünk segítségül:
  - Motiváció: jobban olvasható, mintha egy sorban lenne minden
  - Oszlopok: tranzakciók
  - Soronként egy cella van kitöltve: egy tranzakcióhoz tartozó művelet
  - Sorok fentről lefelé vannak feltöltve sorban az ütemezés műveleteivel
- Táblázat használata:
  - Csak lefelé vizsgálódunk, és csak eltérő oszlopokban
  - Ha T_i egy művelete konfliktusban van egy tőle lejjebb található T_j egy műveletével, akkor T_i-ből mutasson egy nyíl T_j-be
- Konfliktuspárok: $w_i(X); w_j(X)$ és $r_i(X); w_j(X)$

1	2	3	4
$r_1(A)$			
	$r_2(A)$		
		$r_3(B)$	
			$r_4(B)$
		$w_3(B)$	
			$w_4(B)$
$w_1(B)$			
	$w_2(B)$		



- Az eredeti ütemezés:
  $r_1(A); r_2(A); r_3(B); r_4(B); w_3(B); w_4(B); w_1(A); w_2(B)$

5.7. Tábla összekapcsolás költségbecslés

- Memória és táblák méretei: M, B_R, B_S blokk
- Táblákban található rekordok száma: N_R, N_S darab

5.7.1. Blokk-skatulyázott ciklusos (block-nested loop)

- Költség: $\lceil \frac{B_R}{M-1} \rceil * B_S + B_R$
  - Akkor lesz kevesebb, ha $B_R < B_S$
- Algoritmus: beolvasunk $M-1$ blokkot az egyik táblából, a másikat pedig blokkonként végigolvassuk. Utána a következő $M-1$ blokkot olvassuk be az első táblából, stb.

5.7.2. Indexelt skatulyázott ciklusos

- Költség: $B_R + N_R * c$
  - Ahol c az S táblából az index szerinti kiválasztás költsége
    - c értéke állítólag az S tábla vizsgált oszlopán található különböző értékek száma, azaz $c = SC(S, A) = \frac{N_S}{V(S,A)}$
  - Akkor lesz kevesebb a költség, ha $N_R < N_S$
- Algoritmus: beolvasunk R-ből egy blokkot, index segítségével megtaláljuk a megfelelő S rekordokat, majd beolvassuk a következő blokkot R-ből.

5.7.3. Rendezéses-összefésüléses

- Költség: $2*B_R*(1+\lceil \log_{M-1}(\frac{B_R}{M}) \rceil)+2*B_S*(1+\lceil \log_{M-1}(\frac{B_S}{M}) \rceil)+B_R+B_S$
- Rendező algoritmus: először a táblát memóriában elférő rendezett partíciókra osztjuk, azután futamonként $M-1$ darab ilyen partíciót rendezünk, ezzel egy újabb partíciót készítve. Legvégén fájlba beírjuk.
- Összekapcsoló algoritmus: két rendezett táblát párhuzamosan olvassuk.

5.7.4. Hasításos

- Költség: $3 * B_R + 3 * B_S$
- Algoritmus: rekordokat beolvassuk, hash függvényt alkalmazunk az összekapcsolási mezőre és így kosarakba (partíciókba) írjuk őket. Majd a megfelelő kosár párokat kiolvassuk és összekapcsoljuk.