



Sisteme de baze de date

Curs 6 – Limbajul SQL: prelucrarea datelor (3/3).
Organizarea logică a bazei de date

Sorina Preduț

sorina.predut@unibuc.ro

Universitatea din București



Operatorul EXISTS - cont.

- Alte exemple ce folosesc operatorul [NOT] EXISTS, inclusiv **operația de diviziune** menționată în Curs1, precum și **clauza WITH** se găsesc în fișierul DIVISION_clauzaWITH.pdf



Subinterogări - cont.

- Subinterogările mai pot apărea și în alte comenzi SQL cum ar fi: UPDATE, DELETE, INSERT și CREATE TABLE.
- Așa cum am văzut, există în principal 2 moduri de realizare a interogărilor ce folosesc date din mai multe tabele: joncțiuni și subinterogări.
- **Joncțiunile reprezintă forma de interogare relațională** (în care sarcina găsirii drumului de acces la informație revine SGBD-ului), iar **subinterogările forma procedurală** (în care trebuie indicat drumul de acces la informație).
Fiecare dintre aceste forme are avantajele sale, depinzând de cazul specific în care se aplică.



Operații pe tabele ce conțin informații de structură arborescentă

- O bază de date relațională nu poate stoca înregistrări în mod ierarhic, dar la nivelul înregistrării pot exista informații care determină o relație ierarhică între înregistrări.
- SQL permite afișarea rândurilor dintr-o tabelă ținând cont de relațiile ierarhice care apar între rândurile tabelului.
- Parcurgerea în mod ierarhic a informațiilor se poate face doar la nivelul unei singure tabele.
- Operația se realizează cu ajutorul clauzelor **START WITH** și **CONNECT BY** din comanda **SELECT** (Cererile se numesc în acest caz **ierarhice**).

Cereri ierarhice

- De exemplu, în tabela PROFESOR există o relație ierarhică între înregistrări datorată valorilor din coloanele cod și sef.

Fiecare înregistrare aferentă unui cadru didactic conține în coloana sef codul persoanei căreia îi este direct subordonat.

Pentru a obține o situație ce conține niveluri ierarhice, vom folosi următoarea interogare:

```
SELECT LEVEL, nume, prenume, grad  
FROM profesor  
START WITH sef IS NULL  
CONNECT BY PRIOR cod = sef;
```

LEVEL	NUME	PRENUME	GRAD
-----	-----	-----	----
1	GHEORGHIU	STEFAN	PROF
2	MARIN	VLAD	PROF
2	GEORGESCU	CRISTIANA	CONF
3	IONESCU	VERONICA	ASIST
4	STANESCU	MARIA	ASIST
2	ALBU	GHEORGHE	LECT
2	VOINEA	MIRCEA	ASIST



Cereri ierarhice - cont.

- Explicarea sintaxei și a regulilor de funcționare pentru exemplul anterior:
 - Clauza standard SELECT poate conține pseudo-coloana **LEVEL** ce indică nivelul înregistrării în arbore (cât de departe este de nodul rădăcină).
Astfel, nodul rădăcină are nivelul 1, fiii acestuia au nivelul 2, ș.a.m.d.;
 - În clauza FROM nu se poate specifica decât o tabelă;
 - Clauza WHERE poate apărea în interogare pentru a restricționa vizitarea nodurilor (înregistrărilor) din cadrul arborelui;



Cereri ierarhice - cont.

- Clauza CONNECT BY specifică coloanele prin care se realizează relația ierarhică; acesta este clauza cea mai importantă pentru parcurgerea arborelui și este obligatorie;
- Operatorul PRIOR stabilește direcția în care este parcurs arborele. Dacă clauza apare înainte de atributul cod arborele este parcurs de sus în jos, iar dacă apare înainte de atributul sef arborele este parcurs de jos în sus;
- Clauza START WITH specifică nodul (înregistrarea) de început al arborelui. Ca punct de start nu se poate specifica un anumit nivel (LEVEL), ci trebuie specificată valoarea; această clauză este opțională, dacă ea lipsește, pentru fiecare înregistrare se va parcurge arborele care are ca rădăcină această înregistrare.



Cereri ierarhice - cont.

- În sintaxa interogării anterioare, pentru a ordona înregistrările returnate, poate apărea clauza **ORDER BY**, dar este recomandabil să nu o folosim deoarece ordinea implicită de parcurgere a arborelui va fi distrusă.
- Pentru a elimina doar un anumit nod din arbore putem folosi clauza **WHERE**, iar pentru a elimina o întreagă ramură dintr-un arbore (o anumită înregistrare împreună cu fiii acesteia) folosim o condiție compusă în clauza **CONNECT BY**.

Cereri ierarhice - cont.

- Următorul exemplu elimină doar înregistrarea cu numele 'GEORGESCU', dar nu și copiii acesteia:

```
SELECT LEVEL, nume, prenume, grad
FROM profesor
WHERE nume != 'GEORGESCU'
START WITH sef IS NULL
CONNECT BY PRIOR cod = sef;
```

LEVEL	NUME	PRENUME	GRAD
-----	-----	-----	-----
1	GHEORGHIU	STEFAN	PROF
2	MARIN	VLAD	PROF
3	IONESCU	VERONICA	ASIST
4	STANESCU	MARIA	ASIST
2	ALBU	GHEORGHE	LECT
2	VOINEA	MIRCEA	ASIST

Cereri ierarhice - cont.

- Pentru a elimina toată ramura care conține înregistrarea cu numele 'GEORGESCU' și înregistrările pentru subordonații acesteia se folosește următoarea interogare:

```
SELECT LEVEL, nume, prenume, grad
FROM profesor
START WITH sef IS NULL
CONNECT BY PRIOR cod = sef AND nume != 'GEORGESCU' ;
```

LEVEL	NUME	PRENUME	GRAD
1	GHEORGHIU	STEFAN	PROF
2	MARIN	VLAD	PROF
2	ALBU	GHEORGHE	LECT
2	VOINEA	MIRCEA	ASIST



Comanda INSERT

- Această comandă este utilizată pentru adăugarea unor rânduri noi într-o tabelă creată anterior sau în tabelele de bază ale unei vederi.
- Comanda INSERT poate fi utilizată în 2 moduri:

1. Pentru introducerea datelor într-un tabel, câte o înregistrare la un moment dat.

În acest caz sintaxa este următoarea:

```
INSERT INTO tabela [(coloana1, coloana 2, ....)]  
VALUES (valoare1, valoare2, ...);
```



Comanda INSERT - cont.

- În momentul inserării datelor, trebuie respectate următoarele reguli:
 - coloanele pot fi specificate în orice ordine, însă trebuie asigurată corespondența între coloane și valorile furnizate (coloanei 1 îi corespunde valoarea 1, coloanei 2 îi corespunde valoarea 2, ș.a.m.d.) iar coloanelor nespecificate le va fi atașată valoarea Null;
 - în cazul în care coloanele nu sunt specificate explicit, se impune ca ordinea în care apar valorile în comanda INSERT să coincidă cu cea în care coloanele au fost definite la crearea tabelului (dacă nu se cunoaște ordinea de declarare a coloanelor se poate folosi comanda SQL* Plus `DESCRIBE nume_tabela` care va afișa lista coloanelor definite pentru tabela respectivă, tipul, lungimea și restricțiile de integritate);



Comanda INSERT - cont.

- valorile trebuie să aibă același tip de dată ca și câmpurile în care sunt adăugate;
 - dimensiunea valorilor introduse trebuie să fie mai mică sau cel mult egală cu dimensiunea coloanei (un șir de 20 de caractere nu poate fi adăugat într-o coloană cu dimensiunea de 15 caractere);
 - valorile introduse trebuie să respecte restricțiile de integritate definite la crearea tabeli (de exemplu, câmpuri definite ca NOT NULL sau UNIQUE).
-
- Atunci când se inserează valori de tip dată calendaristică în format predefinit (DD-MON-YY), sistemul presupune în mod automat secolul 21, ora 00:00:00 (miezul nopții).



Comanda INSERT - cont.

- Următoarea instrucțiune exemplifică introducerea unei noi înregistrări în tabela profesor:

```
INSERT INTO profesor (cod, nume, prenume, data_nast, sef,  
salariu, cod_catedra)  
VALUES (107, 'POPESCU', 'SERGIU', '09-DEC-71', 100, 1200, 20);
```

Se poate observa că valorile coloanelor grad și primă, care nu au fost specificate, vor fi NULL.



Comanda INSERT - cont.

- În cazul în care nu se specifică implicit numele coloanelor, valorile trebuie introduse în ordinea în care au fost definite și nu se poate omite valoarea nici unei coloane. Următoarea instrucțiune va produce același efect ca cea de mai sus:

```
INSERT INTO profesor  
VALUES (107, 'POPESCU', 'SERGIU', '09-DEC-71', NULL, 100, 1200,  
NULL, 20);
```



Comanda INSERT - cont.

- 2. Pentru introducerea datelor într-un tabel, prin copierea mai multor înregistrări dintr-un alt tabel sau grup de tabele; aceste înregistrări sunt rezultatul unei comenzi SELECT.

În acest caz sintaxa este următoarea:

```
INSERT INTO tabela [(coloana1, coloana 2, ....)] comanda_select;
```

- Și în acest caz trebuie respectate regulile de inserare, singura diferență fiind faptul că valorile nou-introduse sunt extrase cu ajutorul unei interogări, acest lucru creând posibilitatea de inserare a mai multor înregistrări în funcție de anumite condiții.



Comanda INSERT - cont.

- De exemplu, pentru a insera în tabela nou_profesor, având coloanele cod, nume, prenume și data_nastere, înregistrările din tabela profesor care au gradul didactic de asistent se poate folosi următoarea instrucțiune:

```
INSERT INTO nou_profesor (cod, nume, prenume, data_nastere)
SELECT cod, nume, prenume, data_nast
FROM profesor
WHERE grad='ASIST';
```



Comanda UPDATE

- Comanda UPDATE este folosită pentru a modifica valorile datelor existente într-un tabel sau în tabelele de bază ale unei vederi și are următoarea sintaxă generală:

```
UPDATE tabela [alias]  
SET atribuire_coloane, [atribuire_coloane, ...]  
[WHERE condiție];
```

unde atribuire_coloane poate avea una dintre următoarele forme:

coloana = {expresie | (subinterogare)} sau
(coloana [,coloana] ...) = (subinterogare)



Comanda UPDATE - cont.

- Se observă că \exists 2 posibilități de modificare:
 - furnizarea în mod explicit a fiecărei valori sau expresii pentru câmpurile ce trebuie modificate;
 - obținerea valorilor cu ajutorul unei subinterogări.
- Comanda UPDATE modifică valorile înregistrărilor în funcție de condiția clauzei WHERE. În lipsa clauzei WHERE, vor fi actualizate toate înregistrările din tabelul dat.
- Expresia furnizată ca nouă valoare a unei coloane poate cuprinde valorile curente ale câmpurilor din înregistrarea care este actualizată.

Comanda UPDATE - cont.

- De exemplu, pentru a mări salariul cu 20% și prima cu 100 pentru cadrele didactice ce au gradul de asistent, se va folosi următoarea comandă:

```
UPDATE profesor
SET salariu = salariu * 1.2,
    prima = prima + 100
WHERE grad = 'ASIST';
```

- Pentru a exemplifica actualizarea datelor utilizând subinterogări presupunem că mai avem o tabelă numită PRIMA ce conține sumele de bani primite suplimentar de unele cadre didactice:

COD	PRIMA
102	100
103	200
102	50



Comanda UPDATE - cont.

- Pentru a modifica datele din tabela PROFESOR pe baza datelor din tabela PRIMA se poate folosi următoarea comandă care conține o subinterogare corelată și o subinterogare imbricată:

```
UPDATE profesor
SET prima = (SELECT SUM(prima)
              FROM prima a
              WHERE a.cod = profesor.cod)
WHERE cod IN (SELECT cod
              FROM prima);
```



Comanda UPDATE - cont.

- O altă posibilitate este ca sumele suplimentare conținute în tabela PRIMA să fie adăugate la prima existentă în tabela PROFESOR:

```
UPDATE profesor
SET prima = (SELECT SUM (prima) + profesor.prima
              FROM prima a
              WHERE a.cod = profesor.cod)
WHERE cod IN (SELECT cod
              FROM prima);
```

- Să presupunem acum că toți asistenții sunt transferați la catedra din care face parte cadrul didactic cu codul 104 și vor primi același salariu cu acesta:

```
UPDATE profesor  
SET (cod_catedra, salariu) = (SELECT cod_catedra, salariu  
                                FROM profesor  
                                WHERE cod = 104)  
  
WHERE grad = 'ASIST';
```



Comanda DELETE

- Comanda DELETE realizează ștergerea înregistrărilor dintr-o tabelă sau din tabelele de bază ale unei vederi în funcție de o anumită condiție și are următoarea sintaxă generală:

```
DELETE FROM tabela  
[WHERE condiție];
```

- Similar comenzii UPDATE, comanda DELETE șterge anumite înregistrări în funcție de condiția din clauza WHERE.
În lipsa clauzei WHERE vor fi șterse toate înregistrările din tabelul dat.
În această clauză pot fi incluse și subinterogări.



Comanda DELETE - cont.

- De exemplu următoarea comandă șterge toate înregistrările pentru care gradul didactic este asistent

```
DELETE FROM profesor  
WHERE grad = 'ASIST';
```

- **Observații:** Comanda DELETE nu poate fi folosită pentru ștergerea valorii unui câmp individual (pentru aceasta folosiți comanda UPDATE) ci șterge înregistrări complete dintr-un singur tabel.

În plus, comanda DELETE șterge numai înregistrări din tabel nu și tabelul.
Pentru a șterge un tabel se folosește comanda DROP TABLE.



Comanda DELETE - cont.

- Un alt aspect important este faptul că, similar comenzilor INSERT și UPDATE, ștergerea înregistrărilor dintr-un tabel poate determina apariția unor probleme legate de integritatea referențială.

Pentru a evita aceste probleme se pot defini constrângeri de integritate care împiedică operațiile de inserare, actualizare sau ștergere care ar distruge integritatea referențială a datelor.



Comanda TRUNCATE

- Pentru a șterge în mod rapid toate înregistrările dintr-o tabelă sau dintr-un cluster se poate folosi comanda TRUNCATE.
- Comanda TRUNCATE este mult mai rapidă decât comanda DELETE din următoarele motive:
 - Comanda TRUNCATE este o comandă DDL, prin urmare se execută dintr-o singură tranzacție și deci nu folosește segmentul de revenire.
Comanda trebuie folosită cu precauție deoarece nu mai poate fi derulată înapoi.
 - Comanda TRUNCATE nu declanșează triggerul DELETE.



Comanda TRUNCATE - cont.

- Comanda are următoarea sintaxă generală:

```
TRUNCATE {TABLE tabel | CLUSTER cluster} [ {DROP | REUSE}  
STORAGE]
```

unde:

- Clauza TABLE specifică numele unei tabele iar clauza CLUSTER specifică numele unui cluster.

După cum se observă din sintaxă, aceste 2 opțiuni sunt alternative, deci nu se poate specifica într-o comandă TRUNCATE ștergerea rândurilor dintr-o tabelă și dintr-un cluster în același timp.



Comanda TRUNCATE - cont.

În cazul în care se specifică clauza TABLE, tabela la care se referă această clauză nu poate face parte dintr-un cluster.

Comanda TRUNCATE se poate executa și asupra tabelelor organizate pe index.

La trunchierea unei table, Oracle șterge automat datele din indecșii tablei.

În cazul în care se specifică clauza CLUSTER, clusterul la care se referă această clauză nu poate fi un cluster hash ci numai un cluster de index.

De asemenea, la trunchierea unui cluster, Oracle șterge automat datele din indecșii tabelor clusterului.



Comanda TRUNCATE - cont.

- Clauza DROP STORAGE eliberează spațiul alocat înregistrărilor șterse din tabel sau cluster.

Clauza REUSE STORAGE păstrează spațiul alocat înregistrărilor șterse din tabel sau cluster.

Acest spațiu care nu a fost dealocat poate fi reutilizat doar la operații de inserare sau modificare asupra tabelii sau clusterului.

Aceste 2 opțiuni nu modifică efectul pe care îl are comanda TRUNCATE asupra spațiului eliberat de datele șterse din indecșii asociați.

Opțiunea implicită este DROP STORAGE.



Comanda TRUNCATE - cont.

- Ștergerea înregistrărilor cu ajutorul comenzii TRUNCATE este mult mai avantajoasă decât eliminarea tabelului și recrearea lui ulterioară deoarece:
 - Eliminarea tabelului face ca obiectele dependente de acesta să devină invalide, pe când în cazul folosirii comenzii TRUNCATE nu se întâmplă acest lucru;
 - Comanda TRUNCATE nu necesită re acordarea de drepturi asupra tabelului așa cum se întâmplă dacă acesta a fost eliminat și apoi recreat;
 - Eliminarea tabelului necesită recrearea indecșilor, constrângerilor de integritate, declanșatorilor, precum și specificarea parametrilor de stocare.
- **De exemplu, dacă un utilizator execută comanda**
`SELECT COUNT (*) FROM nume_tabel` , iar această interogare returnează după un interval destul de îndelungat valoarea 0, se recomandă trunchierea tabelului.



Organizarea logică a bazei de date



Scheme

- Dezvoltatorul de aplicații trebuie să fie profund conștient de organizarea logică a bazei de date.
- La nivel logic, baza de date este alcătuită din scheme.
- **O schemă este o colecție de structuri logice de date, numite și obiecte ale schemei.**
- O schemă este proprietatea unui utilizator al bazei de date și are același nume cu acesta. De aceea se mai spune că obiectele schemei sunt proprietatea utilizatorului respectiv.



Scheme - cont.

- Definițiile tuturor obiectelor schemei sunt păstrate în dicționarul bazei de date.
- Unele dintre obiectele schemei (tabele, cluster, indici) conțin date, pentru care este necesar un spațiu de stocare.

Datele din fiecare astfel de obiect sunt stocate dpdv **logic** într-un **spațiu tabel**.
Dpdv **fizic**, aceste date sunt stocate într-unul sau mai multe din **fișierele de date asociate acelui spațiu tabel**.

În general, într-un spațiu tabel sunt stocate mai multe obiecte.
La crearea tabelului, clusterului sau indicilor se poate specifica spațiul tabel corespunzător și spațiul alocat obiectului creat (prin intermediul parametrilor de stocare).



Scheme - cont.

- Obiectele schemei pot fi create și manipulate folosind comenzi SQL.

Principalele obiecte ale schemei sunt următoarele:

- tabele (tables),
- vederi (views),
- indecși (indexes),
- clustere (clusters) și clustere hash (hash cluster),
- secvențe (sequences),
- sinonime (synonyms),



Scheme - cont.

- proceduri și funcții stocate/rezidente (stored procedures and functions),
- pachete stocate (stored packages),
- declanșatoare ale bazei de date (database triggers),
- instantanee (snapshots),
- legături ale bazei de date (database link).



Tabele

- Tabelul este principala structură logică de stocare a datelor.
- După cum am văzut în Cursul 1, un tabel este o structură bidimensională formată din **coloane și rânduri**.
Coloanele mai sunt numite și **câmpuri**, iar rândurile **înregistrări**.
- În general, fiecare tabel este stocat într-un spațiu tabel.
- Porțiunea dintr-un spațiu tabel folosită pentru stocarea datelor unui tabel se numește **segment tabel**.
Cu alte cuvinte, segmentul tabel este omologul fizic al unui tabel.



Tabele - cont.

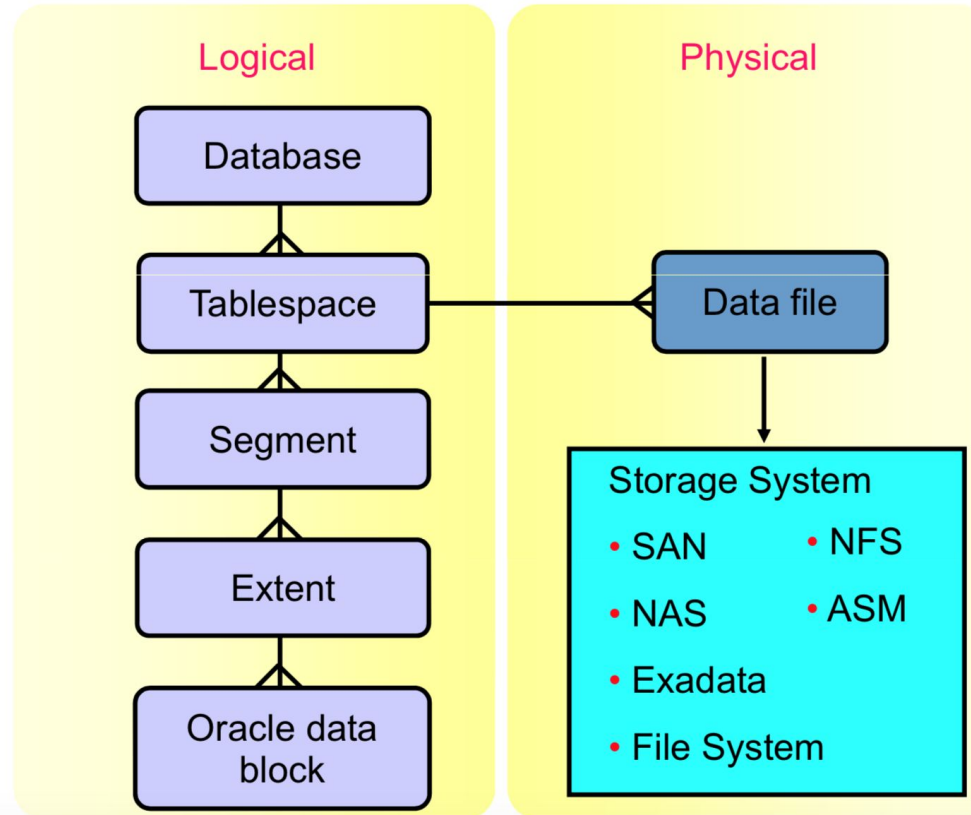
- În anumite situații, pentru a mări eficiența operațiilor de scriere/citire a datelor, mai multe tabele pot fi stocate împreună, formând clustere (grupuri de tabele).
- **O noutate adusă de versiunea Oracle8 este posibilitatea de crea un tabel pe baza unui index, reducând astfel timpul de acces la date prin interogări care folosesc ca termen de comparație coloanele indexate.**

Despre acestea vom discuta ulterior.

În continuare ne vom referi doar la tabele obișnuite.

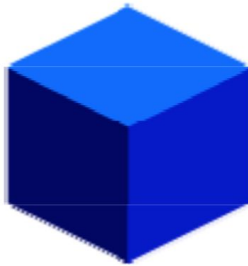
Logical and Physical Database Structures

■



Segments, Extents, and Blocks

- Segments exist in a tablespace.
- Segments are collections of extents.
- Extents are collections of data blocks.
- Data blocks are mapped to disk blocks.



Segment



Extents

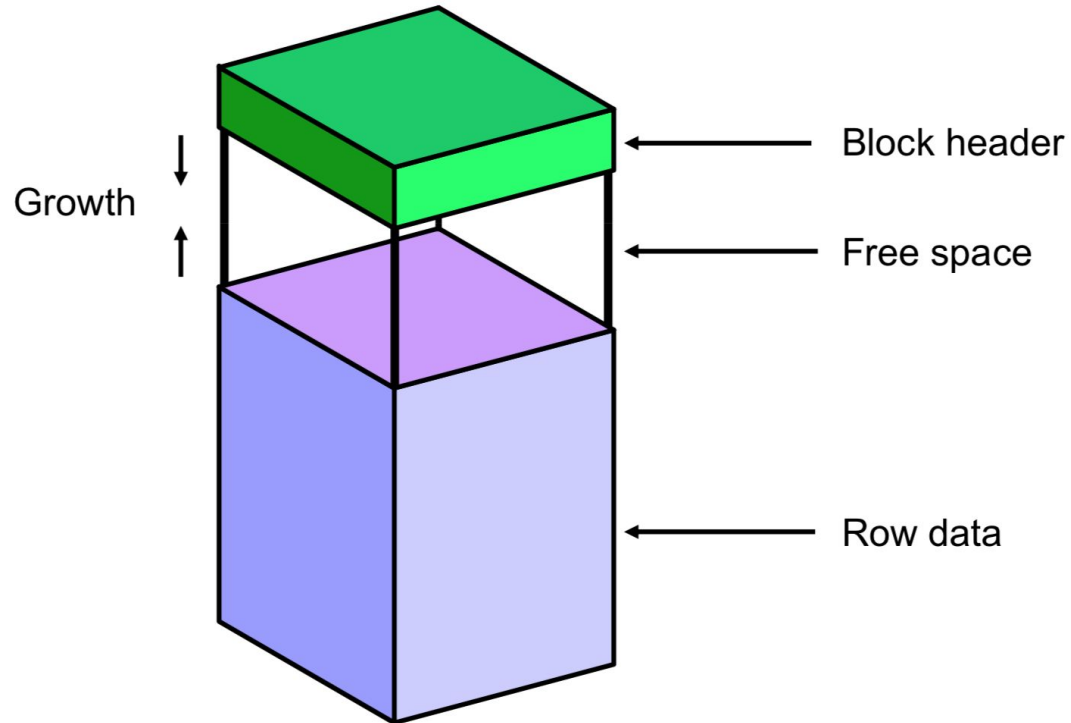


Data
blocks

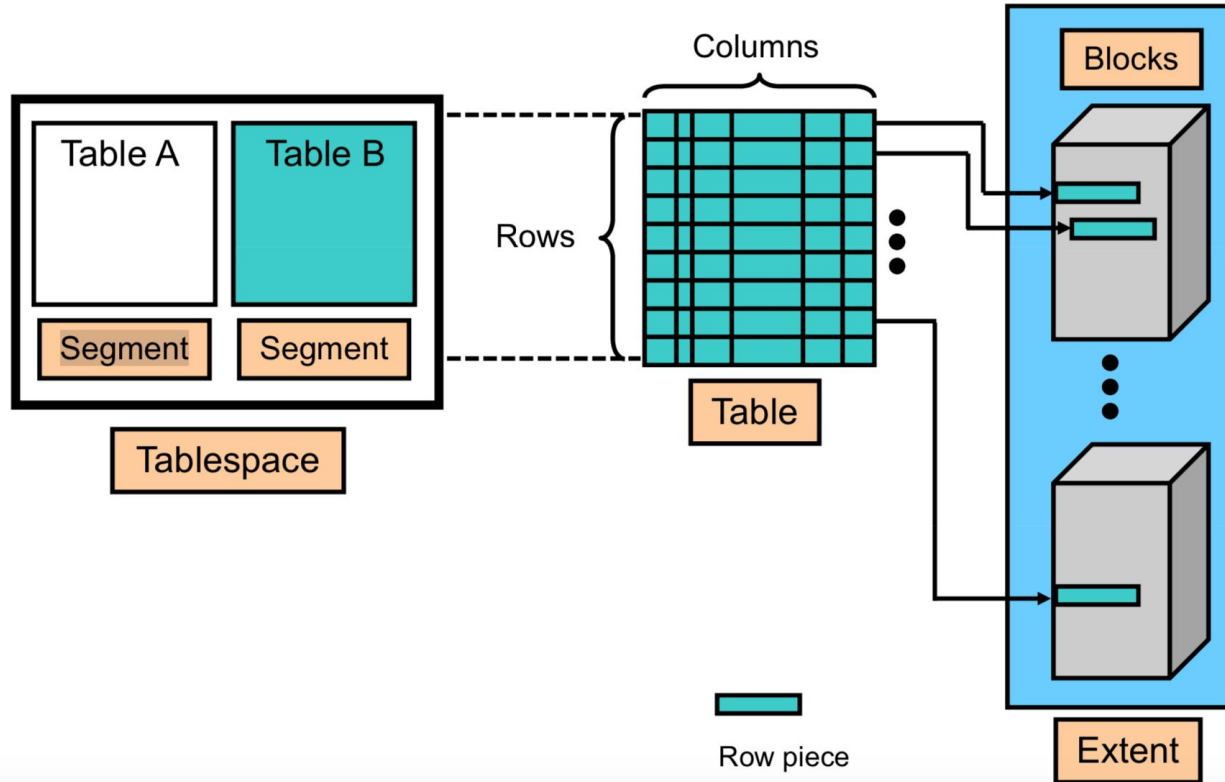


Disk blocks
(File System
Storage)

Database Block: Contents



How Table Data Is Stored





Crearea tabelelor

- Un tabel poate fi creat prin comanda SQL `CREATE TABLE`, în care trebuie specificat numele și tipul de date pentru fiecare coloană a tabelului. De exemplu:

```
CREATE TABLE salariat(  
    cod_salariat NUMBER(10),  
    nume VARCHAR2(10),  
    prenume VARCHAR2(10),  
    data_nastere DATE,  
    salariu NUMBER(10),  
    manager NUMBER(10),  
    cod_dept NUMBER(10),  
    cod_tara NUMBER(10)  
);
```



Crearea tabelelor - cont.

- O sintaxă simplificată a comenzii CREATE TABLE este prezentată în continuare:

```
CREATE TABLE nume_tabel
    (nume_coloana tip_data [DEFAULT expresie]
      [, nume_coloana tip_data [DEFAULT expresie] ... )
    [PCTFREE întreg] [PCTUSED întreg]
    [TABLESPACE spațiu_tabel]
    [STORAGE parametri_de_stocare]
```

unde:



Crearea tabelelor - cont.

- DEFAULT desemnează o valoare implicită pentru coloană, folosită în cazul în care la inserarea unui rând în tabel nu este specificată o valoare explicită pentru coloana în cauză.
- TABLESPACE specifică spațiul tabel în care va fi stocat tabelul.
Dacă acesta nu este menționat explicit, se va folosi spațiul tabel implicit (default) al utilizatorului care este proprietarul schemei din care face parte tabelul.
- Valorile parametrilor PCTFREE și PCTUSED determină gradul de utilizare al blocurilor din extinderile segmentului tabel.
- Clauza STORAGE este folosită pentru setarea parametrilor de stocare (INITIAL, NEXT, PCTINCREASE, MINEXTENTS, MAXEXTENTS) prin intermediul cărora se specifică mărimea și modul de alocare a extinderilor segmentului tabel.



Crearea tabelelor - cont.

```
CREATE TABLE salariat(  
    cod_salariat NUMBER(10),  
    nume VARCHAR2(10),  
    prenume VARCHAR2(10),  
    data_nastere DATE,  
    salariu NUMBER(10),  
    manager NUMBER(10),  
    cod_dept NUMBER(10),  
    cod_tara NUMBER(10) DEFAULT 40)  
PCTFREE 20 PCTUSED 70  
TABLESPACE ts_alfa  
STORAGE (INITIAL 100K NEXT 100K);
```



Crearea tabelelor - cont.

- La crearea unui tabel este necesară specificarea tipului de dată pentru fiecare coloană a tabelului.
- Următorul tabel arată tipurile de date scalare cel mai des folosite.

Tip de dată	Descriere
VARCHAR2(n)	Șiruri de caractere de lungime variabilă având lungimea maximă n bytes. Lungimea maximă n trebuie neapărat specificată. În versiunea Oracle 8, valoarea maximă a lungimii n de 4000.
CHAR(n)	Șiruri de caractere de lungime fixă n bytes. Valoarea implicită pentru n este 1. Dacă într-o coloană având acest tip se inserează șiruri de caractere mai scurte decât lungimea specificată, atunci Oracle inserează la dreapta numărul necesar de spații libere (blank-uri) pentru atingerea lungimii specificate. În versiunea Oracle 8, valoarea maximă a lungimii n este de 2000.
NUMBER(n, m)	Numere cu precizia n și scala m . Precizia reprezintă numărul maxim de digiți permis, care nu poate depăși 38. Scala reprezintă numărul de zecimale pe care le va avea numărul și poate avea valori între -84 și 127.
NUMBER(n)	Numere întregi având precizia maximă n . Valoarea maximă pentru n este de 38.

Tip de dată	Descriere
NUMBER	Numere în virgulă mobilă având o precizie (număr maxim de digiți) de 38 de digiți.
DATE	<p>Date calendaristice având valori între 1 Ianuarie 4712 î.e. n și 31 Decembrie 4712 e.n. Pentru fiecare dată calendaristică sunt înregistrate următoarele informații: secolul, anul, luna, ziua, ora, minutul și secunda.</p> <p>Pentru a specifica o valoare de tip dată calendaristică, este necesară convertirea unui șir de caractere sau a unui număr folosind funcția TO_DATE. Atunci când sunt folosite în expresii de tip dată calendaristică, Oracle convertește automat șiruri de caractere care au formatul implicit de dată calendaristică. Formatul implicit de dată calendaristică este specificat de parametrul de inițializare NLS_DATE_FORMAT și este un șir de caractere. De exemplu, acesta poate fi 'DD-MON-YY', care cuprinde un număr de doi digiți pentru ziua din lună, o abreviație a numelui lunii și ultimii doi digiți ai anului – de exemplu '01-JAN-99' reprezintă 1 Ianuarie 1999.</p>

Tip de dată	Descriere
LONG	Șiruri de caractere de dimensiune variabilă până la 2 Gbytes sau $2^{31}-1$ bytes. O singură coloană de tip LONG este admisă în cadrul unui tabel.
RAW(<i>n</i>)	Se folosește pentru a stoca date binare (șiruri de biți) de lungime variabilă, având lungimea maximă <i>n</i> bytes. Valoarea lui <i>n</i> trebuie specificată și trebuie să nu depășească 2000. Se poate folosi pentru stocarea imaginilor grafice sau a sunetului digital. Este similar cu VARCHAR2, cu excepția dimensiunii maxime și a faptului că pentru tipul de dată RAW nu se pot interpreta datele.
LONG RAW	Se folosește pentru a stoca date binare (șiruri de biți) de lungime variabilă de până la 2Gbytes. Tipul de dată LONG RAW este similar tipului de dată LONG, excepție făcând faptul că pentru tipul de dată LONG RAW nu se pot interpreta datele.



Crearea tabelelor - cont.

- Tipul de dată poate fi urmat de unul sau mai multe numere în paranteză care furnizează informații despre dimensiunea coloanei.
- Dimensiunea coloanei determină dimensiunea maximă a oricărei valori pe care o poate avea coloana.
- Coloanele de tip VARCHAR2 trebuie să aibă specificată o mărime.
- Coloanele NUMBER și CHAR pot avea o mărime specificată, dar în lipsa acestora se folosește o valoare implicită.



Crearea tabelelor - cont.

- Alte date tipuri de date scalare furnizate de Oracle SQL sunt NCHAR și NVARCHAR2, folosite pentru reprezentarea caracterelor limbilor naționale.
Pentru o descriere mai detaliată a acestor tipuri de date se poate consulta [Datatypes](#) care cuprinde toate tipurile de date din Oracle SQL.
- În Oracle, alături de aceste tipuri de date scalare, există și **tipuri de date LOB** (Large Objects), care specifică locația unor obiecte de dimensiuni mari.
În plus, opțiunea **obiect** din Oracle permite definirea de către utilizator a unor tipuri de date.



Crearea tabelelor - cont.

- În Oracle, tabelele pot fi create sau modificate în orice moment, chiar dacă în momentul respectiv există utilizatori care folosesc baza de date.
- La crearea unui tabel nu este nevoie să se specifice dimensiunea maximă a acestuia, ea fiind determinată până la urmă de cât de mult spațiu a fost alocat spațiului tabel în care este creat tabelul.
- Unui tabel îi poate fi repartizat mai mult spațiu în mod automat, în cazul în care spațiul alocat inițial a fost umplut.



Tabele partiționate

- O noutate introdusă în Oracle8 este **posibilitatea de a partiționa tabele**, adică de a împărți tabelul în mai multe părți independente, fiecare cu parametri de stocare potențial diferiți și cu posibilitatea ca părți diferite ale tabelului să se găsească pe spații tabel diferite. Fiecare partiție a tabelului conține înregistrări ce au valoarea cheii într-un anumit interval specificat. În acest sens, partiționarea este foarte folositoare în cazul tabelelor de dimensiuni foarte mari.



Tabele partiționate - cont.

- Partiționarea este transparentă pentru utilizatori și aplicații.
Utilizarea tabelelor partiționate oferă câteva avantaje.
Dacă o parte a tabelului este inaccesibilă, celelalte părți sunt disponibile pentru inserare, selecție, modificare și ștergere; numai acele înregistrări care sunt în acea partiție nu vor fi accesibile.
De asemenea, se poate bloca accesul la o parte a tabelului în timp ce restul înregistrărilor sunt disponibile.



Tabele partiționate - cont.

- Fiecare partiție poate avea proprii săi parametri de stocare PCTFREE și PCTUSED, INITIAL, NEXT, PCTINCREASE, MINEXTENTS, MAXEXTENTS.

Acest lucru este important deoarece o parte a tabelului poate să conțină un nr. mult mai mare de înregistrări decât alta, necesitând, pentru o funcționare eficientă, parametri diferiți de stocare.

Posibilitatea de a atribui în mod individual parametri de stocare fiecărei părți oferă o mai mare flexibilitate în stocarea datelor.

De asemenea, fiecare parte poate fi stocată în spații tabel diferite.

Acest lucru este avantajos în cazul în care unul dintre spațiile tabel este inaccesibil.

- Sintaxa comenzii CREATE TABLE în cazul partiționării tabelului este:




Crearea tabelelor partiționate

```
CREATE TABLE nume_tabel
  (nume_coloană tip_dată [DEFAULT expresie]
  [, nume_coloană tip_dată [DEFAULT expresie] ... )
PARTITION BY RANGE (listă_coloane)
  (PARTITION nume_partiție VALUES [LESS|GREATER] THAN (listă_valori)
  [PCTFREE întreg] [PCTUSED întreg]
  [TABLESPACE spațiu_tabel]
  [STORAGE parametri_de_stocare]
  [, PARTITION nume_partiție VALUES [LESS|GREATER] THAN (listă_valori)
  [PCTFREE întreg] [PCTUSED întreg]
  [TABLESPACE spațiu_tabel]
  [STORAGE parametri_de_stocare]]...)
```



Crearea tabelelor partiționate - cont.

- unde:
 - listă_coloane este o listă ordonată de coloane care determină partiția,
 - listă_valori este o listă ordonată de valori pentru coloanele din listă_coloane.



```
CREATE TABLE salariat_part(  
    cod_salariat NUMBER(10),  
    nume VARCHAR2(10),  
    prenume VARCHAR2(10),  
    data_nastere DATE,  
    salariu NUMBER(10),  
    manager NUMBER(10),  
    cod_dept NUMBER(10),  
    cod_tara NUMBER(10))  
PARTITION BY RANGE(salariu)  
    (PARTITION salariu_mic VALUES LESS THAN (1000)  
    TABLESPACE ts_alfa  
    STORAGE (initial 50K next 50K),  
    PARTITION salariu_mediu VALUES LESS THAN (10000)  
    TABLESPACE ts_beta  
    STORAGE (initial 100K next 100K),  
    PARTITION salariu_mare VALUES LESS THAN (9999999999)  
    TABLESPACE ts_alfa  
    STORAGE (initial 50K next 50K));
```

➤ Notă: MAXVALUE are practic semnificația de “infinit”, ultima parte a tabelului cuprinzând valorile de peste 10000.



Constrângeri

- Alături de numele și tipurile de date ale coloanelor, **la definirea unui tabel se pot specifica și constrângeri (restricții) de integritate (constraints).**
- În Oracle, constrângerile sunt folosite pentru a impune anumite restricții asupra datelor tabelului sau pentru a păstra integritatea referențială a bazei de date.
- Constrângerile se pot defini la nivel de coloană sau la nivel de tabel, după cum ele se referă la datele unei singure coloane sau la datele mai multor coloane.
- În Oracle există următoarele tipuri de constrângeri:



Constrângeri - cont.

Constrângere	Nivel de definire	Funcționalitate
NOT NULL	Coloană	Impune ca valorile coloanei să fie diferite de Null.
UNIQUE	Coloană, tabel	Impune unicitatea valorilor unei coloane sau a unei combinații de coloane.
PRIMARY KEY	Coloană, tabel	Impune unicitatea valorilor unei coloane sau a unei combinații de coloane. În plus, valorile Null nu sunt permise în coloanele care fac parte din PRIMARY KEY. Într-un tabel poate exista o singură cheie primară.
[FOREIGN KEY] REFERENCES	Coloană, tabel	Impune regula de integritate referențială în cadrul aceluiași tabel sau între tabele diferite. O cheie străină este folosită în relație cu o coloană sau combinație de coloane definite ca UNIQUE sau PRIMARY KEY
CHECK	Coloană	Definește explicit o condiție pe care trebuie să o satisfacă datele din fiecare rând al tabelului



Constrângeri - cont.

- În cazul folosirii constrângerilor la definirea unui tabel, sintaxa comenzii SQL CREATE TABLE se completează în modul următor:

```
CREATE TABLE nume_tabel
    (nume_coloana tip_data [DEFAULT expresie] [constr_coloana
        [constr_coloana] ... ],
    [nume_coloana tip_data [DEFAULT expresie] [constr_coloana
        [constr_coloana] ... ]], ...
    constrângere_tabel [, constrângere_tabel] ... )
...
```



Constrângeri - cont.

- **Sintaxa unei constrângeri la nivel de coloană este:**

```
[CONSTRAINT nume constrângere]
{NOT NULL | UNIQUE | PRIMARY KEY
| REFERENCES tabel (coloana) [ON DELETE {CASCADE | SET NULL}]
| CHECK (condiție) }
```

- **iar sintaxa unei constrângeri la nivel de tabel este:**

```
[CONSTRAINT nume constrângere]
{UNIQUE | PRIMARY KEY
| {FOREIGN KEY (coloana [,coloana] ...) REFERENCES
tabel (coloana[,coloana] ...) [ON DELETE {CASCADE | SET NULL}}]}
```



Constrângeri - cont.

- CONSTRAINT permite specificarea unui nume pentru integritatea definită. Dacă această opțiune este omisă, Oracle va genera în mod automat un nume, de forma SYS_Cn, unde n reprezintă un nr. care face ca numele constrângerii să fie unic.
- NOT NULL, UNIQUE, PRIMARY KEY, [FOREIGN KEY] REFERENCES, CHECK sunt tipurile de constrângeri definite anterior.
- ON DELETE {CASCADE | SET NULL} este o clauză care se poate folosi la definirea unei restricții de integritate referențială; în acest caz, în cazul ștergerii unei înregistrări care conține cheia primară sau unică la care face referire cheia străină, integritatea referențială este menținută prin ștergerea tuturor înregistrărilor ce conțin chei străine dependente | modificarea automată a valorilor cheii străine la valoarea NULL.



Constrângeri - cont.

- În exemplul următor sunt create 2 tabele departament și salariat, fiind impuse următoarele constrângeri:
 - combinația (cod_dept, cod_tara) este cheia primară a tabelului departament.
În acest caz, constrângerea este definită la nivel de tabel.
 - cod_salariat este cheia primară a tabelului salariat.
În acest caz, constrângerea este definită la nivel de coloană.
 - nume este o coloană a tabelului salariat care nu admite valori Null.
 - manager este o cheie străină a tabelului salariat care face referință la cheia primară cod_salariat a aceluiași tabel.
În acest caz, constrângerea este definită la nivel de coloană.



Constrângeri - cont.


- valorile pentru coloana salariu din tabelul salariat trebuie să fie mai mari ca 0.
- combinația de coloane (nume, prenume, data_nastere) din tabelul salariat trebuie să aibă valori unice.

În acest caz, constrângerea este definită la nivel de tabel.

- combinația (cod_dept, cod_tara) este o cheie străină a tabelului salariat care face referință la cheia primară a tabelului departament.

În acest caz, constrângerea este definită la nivel de tabel.

Remarcați că în cazul constrângerii de cheie străină, sintaxa diferă în cazul definirii la nivel de coloană față de cel al definirii la nivel de tabel, în prima situație lipsind cuvintele "FOREIGN KEY".




```
CREATE TABLE departament(  
    cod_dept NUMBER(10),  
    cod_tara NUMBER(10),  
    nume_dept NUMBER(10),  
    CONSTRAINT dept_pk PRIMARY KEY(cod_dept, cod_tara));
```

```
CREATE TABLE salariat(  
    cod_salariat NUMBER(10) constraint sal_pk PRIMARY KEY,  
    nume VARCHAR2(10) NOT NULL,  
    prenume VARCHAR2(10),  
    data_nastere DATE,  
    manager NUMBER(10)  
        CONSTRAINT sal_sal_fk REFERENCES salariat(cod_salariat),  
    salariu NUMBER(10)  
        CONSTRAINT sal_ck CHECK(salariu > 0),  
    cod_dept NUMBER(10),  
    cod_tara NUMBER(10),  
    UNIQUE(nume, prenume, data_nastere),  
    CONSTRAINT sal_dept_fk FOREIGN KEY(cod_dept, cod_tara)  
        REFERENCES departament(cod_dept, cod_tara));
```



Constrângeri - cont.

- Constrângerea FOREIGN KEY impune integritatea referențială între tabelul master (departament) și tabelul detaliu (salariat).
De exemplu, aceasta înseamnă că un salariat nu poate fi adăugat decât dacă departamentul corespunzător este fie NULL sau există în tabelul departament.
La fel, nu poate fi șters un departament dacă există angajați în acel departament.
Există însă și posibilitatea de a permite ștergerea unui departament în care există salariați; în acest caz, pentru menținerea integrității, este necesară și ștergerea tuturor angajaților dependenți.
Acest lucru se poate face prin adăugarea clauzei ON DELETE CASCADE pentru constrângerea FOREIGN KEY:



```
CREATE TABLE salariat(  
    cod_salariat NUMBER(10) constraint sal_pk PRIMARY KEY,  
    nume VARCHAR2(10) NOT NULL,  
    prenume VARCHAR2(10),  
    data_nastere DATE,  
    manager NUMBER(10)  
        CONSTRAINT sal_sal_fk REFERENCES salariat(cod_salariat),  
    salariu NUMBER(10)  
        CONSTRAINT sal_ck CHECK(salariu > 0),  
    cod_dept NUMBER(10),  
    cod_tara NUMBER(10),  
    UNIQUE(nume, prenume, data_nastere),  
    CONSTRAINT sal_dept_fk FOREIGN KEY(cod_dept, cod_tara)  
        REFERENCES departament(cod_dept, cod_tara)  
        ON DELETE CASCADE);
```

Constrângeri - cont.

- Toate detaliile despre constrângeri sunt stocate în dicționarul de date Oracle.

De exemplu, pentru a vizualiza toate constrângerile definite pentru tabelele de mai sus putem executa următoarea interogare asupra vederii ALL_CONSTRAINTS:

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE,  
TABLE_NAME  
FROM ALL_CONSTRAINTS  
WHERE TABLE_NAME IN ('SALARIAT',  
    'DEPARTAMENT') ;  
care va produce rezultatul:
```

CONSTRAINT_NAME	C	TABLE_NAME
-----	-	-----
SYS_C002725	C	SALARIAT
SAL_PK	P	SALARIAT
SAL_CK	C	SALARIAT
SYS_C002728	U	SALARIAT
SAL_SAL_FK	R	SALARIAT
SAL_DEPT_FK	R	SALARIAT
DEPT_PK	P	DEPARTAMENT



Constrângeri - cont.

- Fiecare constrângere are asociat un nume.
În general este convenabil ca acesta să fie dat în mod explicit de cel care creează tabelul (cum este cazul constrângerilor CHECK, PRIMARY KEY și FOREIGN KEY din exemplul anterior) pentru că în acest mod constrângerea poate fi referită mai ușor după aceea.
În caz contrar (de exemplu constrângerile UNIQUE și NOT NULL din exemplul anterior) numele este generat automat și are forma "SYS_C...".



Constrângeri amânate

- În Oracle fiecare constrângere este verificată de fiecare dată când este executată o instrucțiune DML (inserare, actualizare sau ștergere).
Există, de asemenea, posibilitatea ca o constrângere să fie amânată (**DEFERRED**).
În cazul acesta, mai multe comenzi SQL pot fi executate fără a se verifica restricția, acesta fiind verificată numai la sfârșitul tranzacției, atunci când este executată instrucțiunea COMMIT.
Dacă vreuna dintre comenzile DML ale tranzacției încalcă restricția, atunci întreaga tranzacție este derulată înapoi și este returnată o eroare.



Constrângeri amânate - cont.

- În Oracle, orice constrângere pe tabelă sau pe coloană poate fi definită ca amânabilă folosind cuvântul cheie **DEFERABLE**;

opțiunea contrară este NOT DEFERABLE care este și opțiunea implicită:

```
{constrângere_tabel | constrângere_coloana} [NOT DEFERABLE |  
DEFERABLE [INITIALLY IMMEDIATE | INITIALLY DEFERRED]]
```



Constrângeri amânate - cont.

- Când o constrângere este specificată ca fiind DEFERRABLE, se poate specifica în plus starea inițială a constrângerii, care poate fi INITIALLY DEFERRED sau INITIALLY IMMEDIATE, setarea implicită fiind INITIALLY IMMEDIATE.

Dacă o constrângere are starea inițială INITIALLY IMMEDIATE, ea este pornită în modul fără amânare, fiind verificată imediat după fiecare instrucțiune executată.

Dacă starea inițială este INITIALLY DEFERRABLE, atunci constrângerea este verificată la executarea unei comenzi COMMIT sau la schimbarea stării constrângerii în IMMEDIATE.



Constrângeri amânate - cont.

- Schimbarea stării unei constrângeri se poate face folosind comanda SQL SET CONSTRAINT:
`SET CONSTRAINT [DEFERRED | IMMEDIATE]`
- Posibilitatea de a amâna verificarea unei constrângeri este folositoare în special în cazul unor restricții de integritate referențială, în acest mod fiind posibilă inserarea unor rânduri în tabela copil (detaliu), care conține cheia străină, înaintea rândului corespunzător din tabela părinte (master), care conține cheia primară.
- În exemplul următor, cele 2 restricții de integritate referențială au fost definite ca amânabile.

```
CREATE TABLE departament(  
    cod_dept NUMBER(10),  
    cod_tara NUMBER(10),  
    nume_dept NUMBER(10),  
    CONSTRAINT dept_pk PRIMARY KEY(cod_dept, cod_tara));
```

```
CREATE TABLE salariat(  
    cod_salariat NUMBER(10) constraint sal_pk PRIMARY KEY,  
    nume VARCHAR2(10) NOT NULL,  
    prenume VARCHAR2(10),  
    data_nastere DATE,  
    manager NUMBER(10)  
        CONSTRAINT sal_sal_fk REFERENCES salariat(cod_salariat)DEFERRABLE,  
    salariu NUMBER(10)  
        CONSTRAINT sal_ck CHECK(salariu > 0),  
    cod_dept NUMBER(10),  
    cod_tara NUMBER(10),  
    UNIQUE(nume, prenume, data_nastere),  
    CONSTRAINT sal_dept_fk FOREIGN KEY(cod_dept, cod_tara)  
        REFERENCES departament(cod_dept, cod_tara)DEFERRABLE);
```



Constrângeri amânate - cont.

- Deoarece starea inițială a restricțiilor nu a fost precizată, ea va fi implicit INITIALLY IMMEDIATE.

Pentru a trece restricțiile în starea DEFERRED se folosește instrucțiunea SET CONSTRAINT:

```
SET CONSTRAINT sal_sal_fk DEFERRED;  
SET CONSTRAINT sal_dept_fk DEFERRED;
```



Constrângeri amânate - cont.

- De exemplu, următoarea secvență de instrucțiuni SQL se execută cu succes dacă restricția referențială sal_dept_fk este DEFERRED, dar eșuează în caz contrar.

```
INSERT INTO salariat(cod_salariat, nume, cod_dept, cod_tara)
    VALUES(100, 'Popescu', 1, 40);
INSERT INTO departament(cod_dept, cod_tara, nume_dept)
    VALUES(1, 40, 'IT');
COMMIT;
```



Crearea și popularea simultană a tabelelor

- Atunci când se creează un tabel există posibilitatea ca în același timp tabelul să fie și populat. Pentru aceasta, în cadrul comenzii SQL CREATE TABLE se va utiliza clauza AS urmată de o interogare pe unul sau mai multe tabele.
În mod evident, nr. coloanelor din definiția tabelului trebuie să coincidă cu acela din interogare.
- Exemplul următor creează un tabel care conține toate înregistrările din tabelul salariat având țara cu codul 100:



Crearea și popularea simultană a tabelelor

```
CREATE TABLE salariat_100
    (cod_salariat, nume, prenume, data_nastere, manager, salariu,
     cod_dept)
AS
    SELECT cod_salariat, nume, prenume, data_nastere, manager, salariu,
           cod_dept
    FROM salariat
    WHERE cod_tara = 100;
```




Crearea și popularea simultană a tabelelor

- Atunci când la crearea unui tabel se folosește clauza AS nu este permisă specificarea tipurilor de date ale coloanelor, acestea fiind preluate automat de la tabelul de bază.
Pe de altă parte însă, restricțiile definite pentru tabelul de bază, cu excepția celor NOT NULL nu sunt preluate automat de noul tabel.
De exemplu, tabelul salariat_100 va avea o singură restricție de integritate, NOT NULL pentru coloana nume.
Folosind clauza AS se pot crea tabele și din mai multe tabele de bază, de exemplu un tabel care conține codul, numele și prenumele salariaților precum și numele departamentului în care lucrează se poate crea în modul următor:



Crearea și popularea simultană a tabelelor

```
CREATE TABLE sal_dept_temp
    (cod_salariat, nume, prenume, nume_dept)
AS
    SELECT s.cod_salariat, s.nume, s.prenume, d.nume_dept
    FROM salariat s, departament d
    WHERE s.cod_dept = d.cod_dept
    AND s.cod_tara = d.cod_tara;
```



Modificarea tabelelor

- Un tabel existent poate fi modificat folosind comanda SQL ALTER TABLE.
Se pot efectua următoarele tipuri de modificări:
 - Adăugarea de noi coloane (împreună cu eventualele constrângeri pentru aceste coloane):

```
ALTER TABLE departament  
ADD (localitate VARCHAR2(10) NOT NULL);
```
 - Modificarea tipului de date sau a mărimii unor coloane existente:

```
ALTER TABLE departament  
MODIFY (nume_dept VARCHAR2(20));
```



Modificarea tabelelor - cont.

- **Notă:** schimbarea tipului de date al unei coloane sau scăderea dimensiunii acesteia nu este posibilă decât dacă acea coloană este goală; în caz contrar, o astfel de operație ar putea duce la modificarea datelor din tabel.



Modificarea tabelelor - cont.

- Ștergerea unor constrângeri existente:

```
ALTER TABLE salariat  
DROP CONSTRAINT sal_ck;
```

- Trebuie remarcat că o constrângere PRIMARY KEY la care face referință o constrângere FOREIGN KEY nu poate fi ștearsă decât dacă împreună cu constrângerea PRIMARY KEY sunt șterse și toate constrângerile referențiale asociate.
Pentru acesta se folosește clauza CASCADE.

```
ALTER TABLE departament  
DROP CONSTRAINT dept_pk CASCADE;
```

- Comanda SQL de mai sus șterge atât constrângerea PRIMARY KEY dept_pk de pe tabelul departament, cât și constrângerea FOREIGN KEY de pe tabelul salariat.



Modificarea tabelelor - cont.

- Adăugarea de noi constrângeri:

```
ALTER TABLE salariat  
ADD (CONSTRAINT data_ck CHECK(data_nastere > '1-Jan-1900'));
```

- Activarea (**ENABLE**) sau dezactivarea (**DISABLE**) unor constrângeri existente;

```
ALTER TABLE salariat DISABLE CONSTRAINT sal_dept_fk;
```



Modificarea tabelelor - cont.

- La crearea unui tabel, toate constrângerile definite sunt implicit active dacă nu a fost folosită opțiunea DISABLE.

Dacă o constrângere este dezactivată, atunci asupra datelor pot fi executate operații care încalcă acea constrângere.

O constrângere care a fost dezactivată poate fi ulterior activată numai dacă datele care au fost introduse, actualizate sau șterse cât timp ea a fost dezactivată nu încalcă această constrângere.

De exemplu, constrângerea sal_dept_fk poate fi reactivată numai dacă după execuția comenzii anterioare nu au fost introduse date în tabelul salariat care încalcă integritatea referențială:

```
ALTER TABLE salariat ENABLE CONSTRAINT sal_dept_fk;
```



Modificarea tabelelor - cont.

- În Oracle, alături de starea activă (ENABLED) și inactivă (DISABLED), o constrângere poate avea o a treia stare: impusă (**ENFORCED**).

Atât restricțiile activate cât și cele dezactivate pot fi trecute în starea ENFORCED.

O restricție poate fi trecută în starea ENFORCED folosind comanda ALTER TABLE cu clauza ENFORCE CONSTRAINT:

```
ALTER TABLE salariat ENFORCE CONSTRAINT sal_dept_fk;
```

În cazul executării acestei comenzi, restricția este impusă după executarea comenzii.

Deci comanda ALTER TABLE ... ENFORCE CONSTRAINT nu va eșua dacă în tabel există înregistrări care încalcă restricția respectivă (cum se întâmplă în cazul executării unei comenzi ALTER TABLE ... ENABLE CONSTRAINT).



Modificarea tabelelor - cont.

Dar, după ce restricția a fost impusă, ea nu va mai permite inserarea sau actualizarea înregistrărilor care nu o respectă, cum s-ar fi întâmplat dacă restricția era dezactivată.

- Eliminarea unei coloane din structura tabelului:

```
ALTER TABLE nume_tabel  
DROP COLUMN coloana;
```

- Eliminare de constrângeri:

```
ALTER TABLE nume_tabel  
DROP PRIMARY KEY | UNIQUE(coloana[, coloana2[, ...]]) | CONSTRAINT  
nume_constr;
```



Modificarea tabelelor - cont.

- Dp dv fizic, comanda ALTER TABLE permite schimbarea parametrilor PCTFREE și PCTUSED și a parametrilor din clauza STORAGE folosind sintaxa:

```
ALTER TABLE nume_tabel  
[PCTFREE întreg] [PCTUSED întreg]  
[STORAGE parametri_de_stocare]
```



Modificarea tabelelor - cont.

- De asemenea, comanda ALTER TABLE permite alocarea și dealocarea manuală a spațiului utilizat de către un tabel.
- **Alocarea manuală a spațiului pentru un tabel se face prin adăugarea de noi extinderi.**
Alocarea manuală se poate face în general:
 - înainte de o încărcare masivă a datelor;
 - pentru a controla distribuția extinderilor unui tabel în cadrul fișierelor.
- Dealocarea spațiului asociat unui tabel reprezintă eliberarea spațiului nefolosit de acesta (care nu a fost niciodată folosit sau care a devenit între timp gol datorită ștergerii de rânduri).



Modificarea tabelelor - cont.

- Pentru a alocă sau dealoca spațiul utilizat de un tabel se folosește comanda ALTER TABLE cu următoarele sintaxe:

```
ALTER TABLE nume_tabel  
ALLOCATE EXTENT ([SIZE întreg [K|M]]  
[DATAFILE nume_fișier_de_date] )]
```

respectiv

```
ALTER TABLE nume_tabel  
DEALLOCATE UNUSED [KEEP întreg [K|M]]
```

unde:



Modificarea tabelelor - cont.

- DATAFILE specifică fișierul de date (din spațiul tabel asociat tabelului) care va cuprinde noua extindere.
Dacă această opțiune este omisă, fișierul este ales de către Oracle.
- SIZE specifică dimensiunea noii extinderi.
Dacă opțiunea SIZE este omisă atunci Oracle va stabili dimensiunea extinderii pe baza parametrilor de stocare ai tabelului.
- Cu ajutorul opțiunii KEEP se poate specifica un număr de bytes (Kbytes, Mbytes) din spațiul liber al tabelului ce nu vor fi dealocați.



Distrugerea tabelelor

- Pentru a distruge un tabel în Oracle se poate folosi comanda SQL:
`DROP TABLE salariat;`
- Pe de altă parte însă, dacă vom folosi o comandă similară pentru a distruge un tabel a cărui cheie primară face referință la o cheie străină a altui tabel, adică ultimul tabel are definită o constrângere FOREIGN KEY corespunzătoare, de exemplu
`DROP TABLE departament;`
atunci existența constrângerii de integritate referențială va împiedica distrugerea tabelului, astfel încât la încercarea de a executa comanda de mai sus se va genera un mesaj de eroare.



Distrugerea tabelelor - cont.

În astfel de situații, tabelul trebuie distrus împreună cu toate constrângerile FOREIGN KEY care fac referire la cheia primară a acestuia.

Acest lucru se poate face prin folosirea comenzii cu clauza CASCADE CONSTRAINTS:

```
DROP TABLE departament CASCADE CONSTRAINTS;
```

Execuția comenzii de mai sus va duce la distrugerea tabelului departament și a constrângerii referențiale sal_dept_fk.

- În momentul în care un tabel este distrus, vor fi șterse automat și toate datele din tabel cât și indecșii asociați lui.

Vederile și sinonimele asociate unui tabel care a fost distrus vor rămâne dar vor deveni invalide.



Indecși

- Un index este o **structură opțională a bazei de date care permite accesarea directă a unui rând dintr-un tabel.**
- Indecșii pot fi creați pentru una sau mai multe coloane a unui tabel, în acest ultim caz folosindu-se denumirea de indecși compuși sau indecși concatenați.
- Un index este **utilizat** de către baza de date **pentru a găsi rapid valori pentru coloana sau coloanele pentru care a fost creat indexul**, în acest mod furnizând o cale de acces directă la liniile asociate acestora **fără a mai fi necesară investigarea fiecărui rând din tabel.**



Indecși - cont.

- Practic, în momentul în care se dorește căutarea anumitor înregistrări ale căror valori îndeplinesc un anumit criteriu, în loc să se parcurgă tabelul, se parcurge indexul, acesta din urmă furnizând localizarea exactă a înregistrărilor ce îndeplinesc criteriul de căutare prin precizarea **ROWID**.
Așa cum am mai precizat anterior, identificatorul de rând ROWID reprezintă în primul rând cel mai rapid mod de a localiza o anumită înregistrare.
Prin urmare pentru a găsi o anumită înregistrare în modul cel mai rapid se determină ROWID-ul acesteia în loc să se parcurgă secvențial tabelul respectiv.
Fără indecși, operații precum determinarea unicității sau ordonarea valorilor unei coloane ar putea consuma multe resurse și timp.



Indecși - cont.

- Prezența indecșilor este transparentă pentru utilizator și aplicație, ei neputând fi referiți în mod direct prin interogări.

În plus, **sintaxa unei comenzi SQL nu este în nici un fel influențată de existența indecșilor, iar rezultatele \forall interogări vor fi aceleași indiferent dacă \exists indecși sau nu.**

Pe de altă parte însă, utilizarea corectă a acestora poate influența în cel mai mare grad eficiența unei interogări: **diferența va consta în rapiditatea cu care se execută comanda și nu în rezultatele acesteia.**

Foarte multe sisteme ce raportează probleme de performanță suferă din cauza lipsei unui index sau din cauza absenței unui index optim.

Informația conținută în indecși este redundantă, ea fiind derivată din informația care există în tabele.



Indecși - cont.

- Indecșii sunt independenți d.p.d.v. fizic și logic de datele din tabelul de bază.
- Un index poate fi creat și distrus fără ca datele din tabelul de bază sau ceilalți indecși să aibă de suferit.

Singurul lucru pe care indexul îl va modifica va fi durata accesului la datele tabelului, acesta devenind mai lent în absența indexului.

Evident, odată cu ștergerea unui tabel sunt șterși și indecșii asociați acestuia.



Indecși - cont.

- Indecșii pot să fie unici sau ne-unici.

Indecșii unici garantează faptul că nu va exista nici o pereche de linii care să aibă valori identice pentru coloana sau grupul de coloane pentru care a fost definit indexul.

Valorile Null nu sunt considerate pentru unicitate.

Un rând cu valoarea Null în coloana indexată nu va fi înregistrat în index, deci un index unic nu va împiedica stocarea mai multor rânduri cu o valoare Null în coloana indexată.

Un index ne-unic nu impune nici o restricție în legătură cu valorile din coloanele care îl definesc.

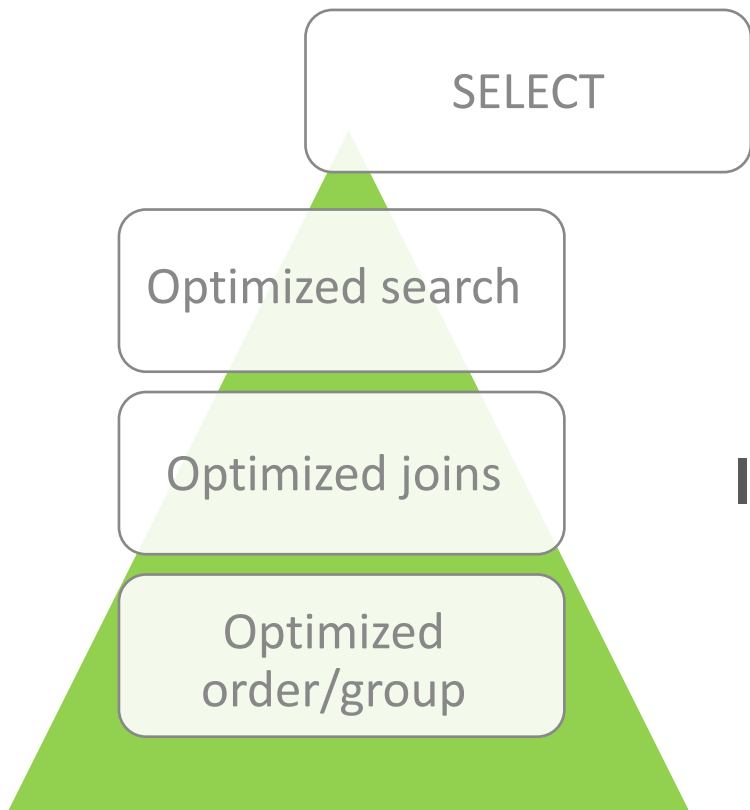


Indecși - cont.

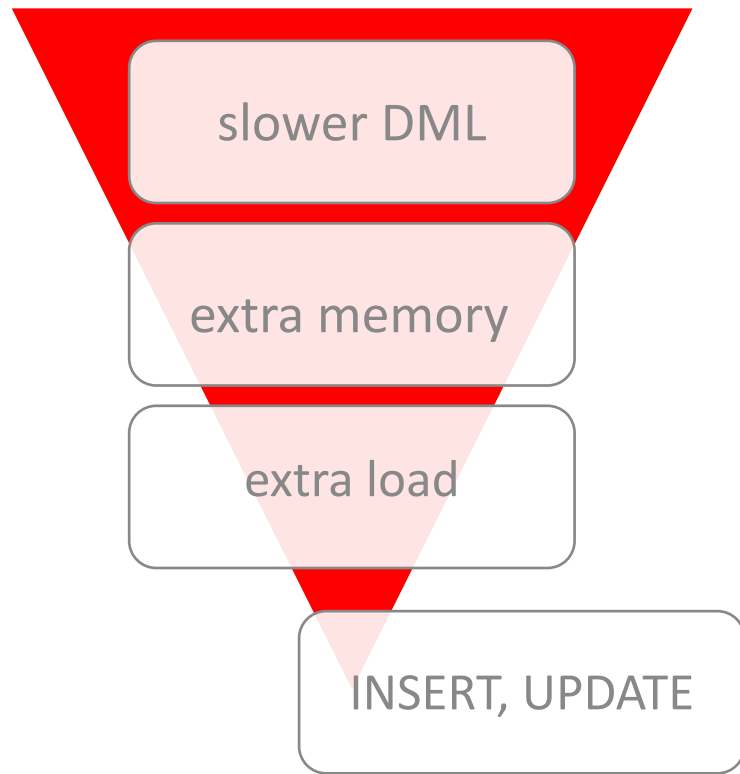
- Odată definit, un **index este actualizat de către baza de date ori de câte ori au loc modificări ale datelor tabelului.**

Aceasta înseamnă, că ori de câte ori au loc inserări, ștergeri sau modificări ale datelor unui tabel, toți indecșii acelui tabel trebuie actualizați în mod automat, acest lucru având ca efect încetinirea acestor operații asupra datelor tabelului.

Cu alte cuvinte, **existența indecșilor sporește viteza accesului la datele tabelului pe de o parte, dar în același timp încetinește operațiile de modificare ale acestora.**




Index





Indecși - cont.

- De aceea, **nu trebuie creat un index pentru fiecare coloană a unui tabel** (deși acest lucru este posibil), **ci pentru anumite coloane cheie ale acestuia**.
De exemplu, se recomandă indexarea coloanelor care conțin în majoritate valori unice sau un domeniu larg de valori sau coloane după care se fac dese căutări sau ordonări. Deoarece prezența indecșilor poate avea un impact semnificativ asupra eficienței aplicației în continuare prezentăm câteva sugestii privind folosirea acestora:



Indecși - cont.

- **Ce tabele trebuie indexate:**
 - Indexați tabelele pentru care majoritatea interogărilor selectează doar un nr. redus de rânduri (sub 5%). Interogările care selectează un nr. mare de rânduri nu folosesc în mod eficient indecși.
 - Nu indexați tabele ce conțin puține înregistrări deoarece în acest caz accesul secvențial va fi mai rapid.
 - Indexați tabelele care sunt interogate folosind clauze SQL simple. Clauzele SQL mai complexe nu folosesc cu aceeași eficiență indecși.



Indecși - cont.

- **Ce tabele trebuie indexate:**
 - Nu indexați tabelele care sunt actualizate frecvent.
Inserările, modificările și ștergerile sunt îngreunate de existența indecșilor.
Decizia de a indexa un tabel trebuie luată pe baza raportului dintre nr. de interogări și cel de actualizări efectuat asupra acestuia.
 - Indexați tabelele care nu au valori duplicate în coloanele care apar în clauza WHERE a celor mai frecvente interogări.



Indecși - cont.

➤ **Ce coloane trebuie indexate:**

- Folosiți coloanele cele mai frecvent folosite în clauza WHERE a interogărilor.
- Nu indexați coloane care nu au multe valori unice.

Totuși, începând cu Oracle8 se **pot indexa coloane care nu au multe valori distincte cu ajutorul indexului de tip bitmap** asupra căruia vom reveni mai târziu.

- Coloanele care au valori unice sunt candidate foarte bune pentru indexare. De altfel, **Oracle creează în mod automat indecși unici pentru coloanele definite ca PRIMARY KEY sau UNIQUE.**

În plus, d.p.d.v. logic, este preferabil ca indecșii unici să nu fie definiți în mod explicit, ci prin intermediul constrângerilor PRIMARY KEY și UNIQUE - acesta deoarece unicitatea este un concept logic și ar trebui să fie asociat cu definiția tabelului.



Indecși - cont.

➤ **Ce coloane trebuie indexate:**

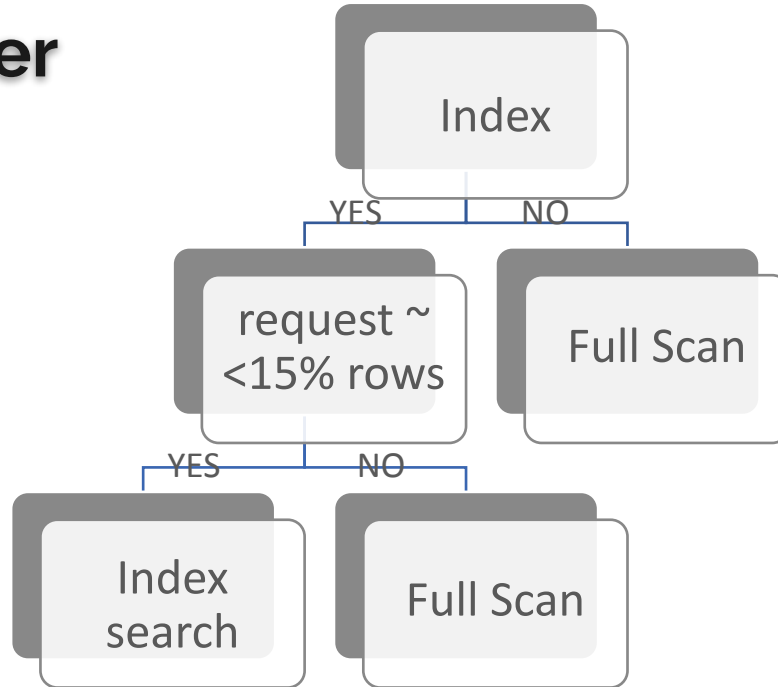
- Coloanele care sunt folosite pentru a face legătura dintre tabele sunt în general candidate pentru indexare.

În general, se recomandă **indexarea cheilor străine**.

- În anumite situații, folosirea indecșilor compuși poate fi mai eficientă decât a celor individuali.

De exemplu, crearea indecșilor compuși este recomandabilă când 2 coloane nu sunt unice fiecare în parte, dar combinația lor este unică sau are în majoritate valori unice. De asemenea, se recomandă crearea indecșilor compuși atunci când interogările uzuale ale tabelului conțin în clauza WHERE mai multe coloane individuale legate prin AND.

Sql Optimizer





Indecși - cont.

- Indecșii consumă spațiu în baza de date la fel ca și tabelele și \exists într-un spațiu tabel la fel ca și acestea.

Spațiul necesar indecșilor pentru tabele mari este de obicei semnificativ, așa că trebuie planificat din momentul în care se proiectează baza de date.



Crearea, modificarea și distrugerea indecșilor

- În SQL un index se creează folosind comanda CREATE INDEX.

O sintaxă simplificată a acestei comenzi este:

```
CREATE [UNIQUE] INDEX nume_index  
    ON tabel (coloana [,coloana] ...)  
    [PCTFREE întreg] [PCTUSED întreg]  
    [TABLESPACE spațiu_tabel]  
    [STORAGE parametrii_de_stocare]
```

unde:

- Dacă este specificată opțiunea UNIQUE, indexul creat este unic, altfel nu.
- Valorile parametrilor PCTFREE și PCTUSED determină gradul de utilizare a blocurilor din extinderile segmentului de index.



Crearea, modificarea și distrugerea indecșilor

- TABLESPACE specifică spațiul tabel în care va fi stocat indexul.
Dacă acesta nu este menționat explicit, se va folosi spațiul tabel implicit (default) al utilizatorului care este proprietarul schemei din care face parte indexul.
În general, se recomandă stocarea indexului într-un spațiu tabel diferit de cel în care este stocat tabelul său; în acest caz Oracle poate accesa tabelul și indexul în paralel, obținându-se astfel o creștere a performanțelor interogărilor.
- Clauza STORAGE este folosită pentru setarea parametrilor de stocare (INITIAL, NEXT, PCTINCREASE, MINEXTENTS, MAXEXTENTS) prin intermediul cărora se specifică mărimea și modul de alocare a extinderilor segmentului de index.



Crearea, modificarea și distrugerea indecșilor

- Următoarele comenzi SQL creează un index compus sal_dept_ind pentru coloanele cod_dept și cod_tara din tabelul salariat și respectiv un index unic pentru coloana nume_dept a tabelului departament.

```
CREATE INDEX sal_dept_ind ON salariat(cod_dept, cod_tara);  
CREATE UNIQUE INDEX nume_dept_ind ON departament(nume_dept);
```
- Crearea indecșilor unui tabel se recomandă a se face după ce tabelul a fost populat cu date - aceasta deoarece existența indecșilor încetinește în mod evident inserarea datelor.
- La definirea constrângerilor **PRIMARY KEY** sau **UNIQUE** sau la activarea acestora, Oracle creează în mod automat indecși unici pentru coloanele sau grupurile de coloane respective. În acest caz numele indexului coincide cu numele constrângerii.



Crearea, modificarea și distrugerea indecșilor

- La fel ca și în cazul tabelelor, parametrii de stocare a indecșilor pot fi modificați ulterior. Pentru aceasta se folosește comanda ALTER INDEX cu următoarea sintaxă:

```
ALTER INDEX nume_index  
STORAGE parametrii_de_stocare
```

De exemplu:

```
ALTER INDEX sal_dept_ind  
STORAGE (NEXT 400K MAXEXTENTS 100);
```

- De asemenea, Oracle8 permite alocarea și dealocarea manuală a spațiului utilizat de un index.



Crearea, modificarea și distrugerea indecșilor

- **Alocarea manuală a spațiului pentru un index înseamnă adăugarea manuală a unei noi extinderi.**

De exemplu, alocarea manuală se poate face înaintea unei perioade în care se va înregistra o activitate intensă de inserări în tabela pe care este bazat indexul.

În acest caz, alocarea manuală previne extinderea dinamică a indexului (alocarea dinamică a unei noi extinderi), împiedicând astfel creșterea timpului de execuție. Dealocarea spațiului asociat unui index reprezintă eliberarea spațiului nefolosit de acesta.



Crearea, modificarea și distrugerea indecșilor

- Pentru a alocă sau dealoca spațiul utilizat de un index se folosește comanda ALTER TABLE cu următoarele sintaxe:

```
ALTER INDEX nume_index  
ALLOCATE EXTENT ([SIZE întreg [K|M]]  
                [DATAFILE specificație_fișier_de_date] )
```

respectiv

```
ALTER INDEX nume_index  
DEALLOCATE UNUSED [KEEP întreg [K|M]]
```

- Semnificația parametrilor din aceste comenzi este aceeași ca și în cazul tabelelor.



Crearea, modificarea și distrugerea indecșilor

- Un index poate fi distrus folosind comanda SQL DROP INDEX cu următoarea sintaxă: `DROP INDEX nume_index;`
De exemplu:
`DROP INDEX sal_dept_ind;`
- **Un index nu poate fi șters dacă el a fost creat în mod automat**, ca parte a definirii sau a activării unei restricții PRIMARY KEY sau UNIQUE.
În acest caz indexul este șters automat la ștergerea sau dezactivarea constrângerii.



Crearea, modificarea și distrugerea indecșilor

- Ștergerea unui index se face de obicei dacă indexul nu mai este necesar, dacă se dorește reconstruirea sa (despre care vom discuta în continuare) sau înainte de încărcarea masivă a datelor într-un tabel;

ștergerea unui index înainte de încărcarea masivă a datelor și recrearea lui după terminarea încărcării va duce la îmbunătățirea performanței încărcării precum și la utilizarea spațiului alocat indexului în mod mai eficient.



Crearea, modificarea și distrugerea indecșilor

- Pentru a reconstrui un index se pot folosi două metode.
Prima este de a șterge indexul folosind comanda DROP INDEX și de a îl recrea folosind comanda CREATE INDEX.

A doua este folosirea ALTER INDEX cu opțiunea REBUILD folosind sintaxa:

```
ALTER INDEX nume_index REBUILD  
[PCTFREE întreg] [PCTUSED întreg]  
[TABLESPACE spațiu_tabel]  
[STORAGE parametrii_de_stocare]  
[REVERSE | NOREVERSE];
```



Crearea, modificarea și distrugerea indecșilor


- **Reconstruirea** unui index se face în următoarele situații:
 - **indexul existent trebuie mutat într-un spațiu tabel diferit** (de exemplu, dacă indexul a fost creat în mod automat de o constrângere de integritate, caz în care el se găsește în același spațiu tabel cu tabelul).
 - **un index normal trebuie convertit într-un index cu cheie inversă** (despre tipurile de indecși vom vorbi în continuare).

Pentru a converti un index obișnuit într-un index cu cheie inversă se folosește comanda ALTER INDEX ... REBUILD cu opțiunea REVERSE (opțiunea contrară este NOREVERSE, care este și opțiunea implicită)
 - **pentru a recupera spațiul de stocare sau pentru a schimba attributele fizice de stocare.**



Crearea, modificarea și distrugerea indecșilor

- În cazul reconstruirii indexului folosind comanda `ALTER INDEX ... REBUILD`, în timpul creării noului index este păstrat și indexul original, astfel că indexul poate fi folosit de interogări și în timpul acestei operații.
Principalele avantaje ale fiecărei dintre cele 2 metode sunt rezumate în tabelul următor:

Șterge și recreează	Folosește opțiunea REBUILD
<p>Poate redenumi indexul.</p> <p>Poate schimba între UNIQUE și non-UNIQUE.</p>  <p>Poate schimba între tipul de index bazat de arbore B* și indexul de tip bitmap (vezi paragrafele următoare).</p> <p>Are nevoie de spațiu doar pentru o copie a indexului.</p> <p>Necesită sortare.</p> <p>Index este indisponibil în perioada dintre ștergere și recreare.</p> <p>Nu se poate folosi această metodă dacă indexul a fost creat de o constrângere PRIMARY KEY sau UNIQUE. În acest caz ștergerea indexului se poate face prin ștergerea sau dezactivarea constrângerii respective.</p>	<p>Nu poate redenumi indexul.</p> <p>Nu poate schimba între UNIQUE și non-UNIQUE.</p> <p>Nu poate schimba între tipul de index bazat de arbore B* și indexul de tip bitmap.</p> <p>Are nevoie de spațiu suplimentar pentru a duplica indexul în mod temporar.</p> <p>Nu necesită sortare, folosindu-se indexul inițial.</p> <p>Indexul este disponibil interogărilor.</p> <p>Se poate folosi această metodă dacă indexul a fost creat de o constrângere PRIMARY KEY sau UNIQUE.</p>



Bibliografie

F. Ipate, M. Popescu, *Dezvoltarea aplicațiilor de baze de date în Oracle 8 și Oracle Forms 6*, Editura ALL, 2000.