



Baze de date

Curs 10 – Organizarea logică a bazei de date

Sorina Preduț

sorina.predut@fmi.unibuc.ro

Universitatea din București



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, indecși) 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 tabel, clusterelor sau indecșilor 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 s-a discutat în cursul 4.

În continuare ne vom referi doar la tabele obișnuite.



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),  
    PARTITIONED 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]
| CHECK (condiție) }
```

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

```
[CONSTRAINT nume constrângere]
{UNIQUE | PRIMARY KEY
| {FOREIGN KEY (coloana [,coloana] ...) REFERENCES tabel (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 în tabelul din slide-ul 26.
- 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 salariat.
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.

- **Notă:** Comanda SQL ALTER TABLE nu permite ștergerea dintr-un tabel a unei coloane existente.

Dacă totuși se dorește acest lucru, se poate folosi comanda CREATE TABLE cu clauza AS, în care se selectează coloanele dorite.

De asemenea, comanda SQL ALTER TABLE nu permite modificarea definiției unei constrângeri existente.



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.



Bibliografie

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