

Capitolul 15. Scheme, obiecte și utilizatori

Până acum am lăsat să se înțeleagă că suntem singurii utilizatori ai bazei de date. În realitate, am creat obiecte ale bazei de date într-o (sub)schemă în care am avut voie să ne facem de cap. Noțiunea de subschemă din acest capitol vine în completarea și, pe alocuri, în contradicția noțiunii de schemă formulată în capitolul 2. Spuneam atunci că schema se referă la *structura* bazei (tabele, tabele virtuale, restricții etc). În cele ce urmează, schema este alcătuită din subscheme, fiecare subschemă fiind accesibilă unuia sau mai multor utilizatori/grupuri.

Dat fiind că în aplicațiile integrate, bazele de date pot fi organizate în mai multe subscheme, utilizatorii și grupurile de utilizatori având drepturi diferite la aceste subscheme, ne interesează cum le putem crea, cum acordăm drepturi la obiecte ale subschemelor altor utilizatori/grupuri decât cei „titulari”, și cum răsfoim dicționarul de date pentru a afla informații despre toate obiectele dintr-o (sub) schemă.

15.1. Crearea de subscheme și acordarea de drepturi

Termenul de subschemă este, așadar, acum mai apropiat celui de *container* sau de *director* (cu semnificația din sistemele de operare, de „folder” de fișiere), fiind o colecție de obiecte. Începând cu SQL-92, sintaxa SQL pentru crearea schemei este:

```
CREATE SCHEMA <clauză de denumire>
```

```
    [<specificarea setului de caractere>] [<element>...]
```

unde:

```
<clauză de denumire> ::= <nume schemă> | AUTHORIZATION
```

```
<identificator de autorizare>
```

```
| <nume schemă> AUTHORIZATION <identificator de autorizare>
```

```
<specificarea setului de caractere> ::=
```

```
DEFAULT CHARACTER SET <specificatorul setului de caractere>
```

```
<element> ::= <definiție domeniu>
```

```
| <definiție tabelă> | <definiție tabelă virtuală>
```

```
| <comandă pentru acordare de drepturi> | <definiție aserțiune>
```

Sintaxa și regimul de funcționare a subschemelor, inclusiv mecanismul de creare a utilizatorilor și de acordare/revocare de drepturi la obiecte din fiecare subschemă, diferă în bună măsură de la SGBD la SGBD. Mai trebuie adăugat că diferențele de titulatură și organizare ale subschemelor țin de specificul serverelor, inclusiv de cel al instalării.

15.1.1. Utilizatori și subscheme în DB2

În DB2, la crearea unei scheme aceasta poate fi „repartizată” unui utilizator (titular). Suntem interesați să creăm un utilizator cu numele *salarizare* pentru care vom defini o schemă cu același nume. Din păcate, utilizatorii bazei pot fi creați doar interactiv, întrucât dialectul SQL nu conține comanda CREATE USER. Din meniul arborescent al *Control Center* – vezi figura 15.1 – după selectarea opțiunii *DB Users* din meniul arborescent (stânga figurii) se afișează meniul contextual din care se alege *Add User* (meniul contextual se afișează printr-un click pe butonul din dreapta al mouse-ului.)

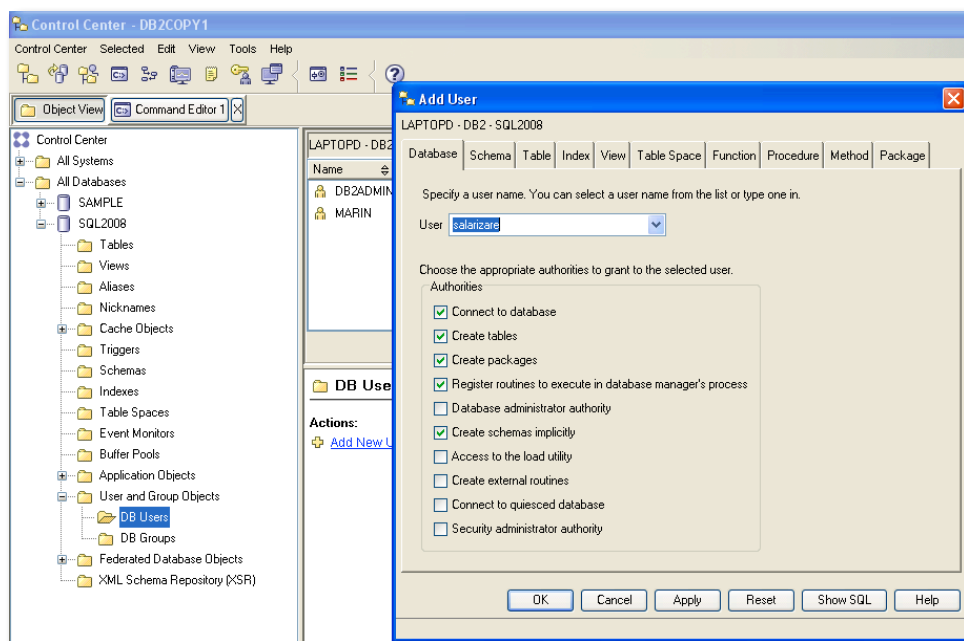


Figura 15.1. Crearea unui utilizator DB2 folosind Control Center-ul

După OK-ul de rigoare, noul utilizator apare în listă, așa că putem trece la crearea (sub)schemei. O întrebare persistă, totuși: cum de nu am fost întrebați nimic despre parolă? Vom vedea cât de curând că acesta este un semn rău.

Comanda de creare a schemei este destul de simplă. Adăugăm și o comandă COMMENT:

```
CREATE SCHEMA salarizare AUTHORIZATION salarizare ;
COMMENT ON Schema SALARIZARE IS 'fara comentarii';
```

Crearea se putea derula și grafic, cu același meniu arborescent de mai sus, din care alegem opțiunea *Schemas* – vezi figura 15.2. La rubrica *Authorization name* indicăm utilizatorul „titular” al schemei - *salarizare*.

După creare, încercăm să dăm drepturi utilizatorului *salarizare* la două dintre tabelele schemei curente (schemă curentă care se numește *MARIN*):

GRANT ALL ON TABLE personal2 TO USER salarizare ;

GRANT ALL ON TABLE sporuri TO USER salarizare ;

În continuare schimbăm subschema curentă, declarând-o pe cea nou creată - SALARIZARE:

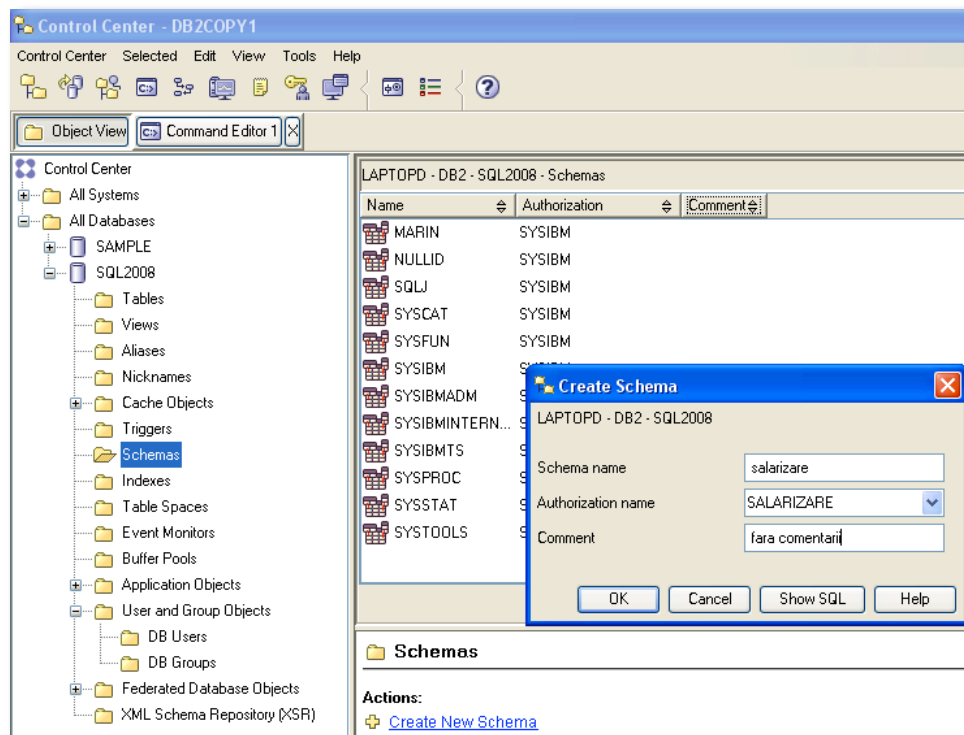


Figura 15.2. Crearea unei (sub)scheme DB2 folosind Control Center-ul

SET CURRENT_SCHEMA salarizare ;

Chiar dacă subschema curentă este acum SALARIZARE, faptul că utilizatorul curent este *marin*, tabelele (sub)schemei MARIN pot fi interogare, indiferent dacă le prefixăm sau nu:

SELECT * FROM marin.personal2 sau SELECT * FROM personal2

Necazurile apar dacă încercăm să ne conectăm (la serverul care se numește, în cazul nostru, *SQL2008*) cu numele noului utilizator:

CONNECT TO sql2008 USER salarizare

pentru că ni se solicită parola utilizatorului (vezi figura 15.3), iar noi nu ne amintim să fi precizat așa ceva. Indiferent ce am tasta la rubrica *Password*, vom primi un mesaj de eroare: *SQL30082N Security processing failed with reason "24" ("USERNAME AND/OR PASSWORD INVALID")*.

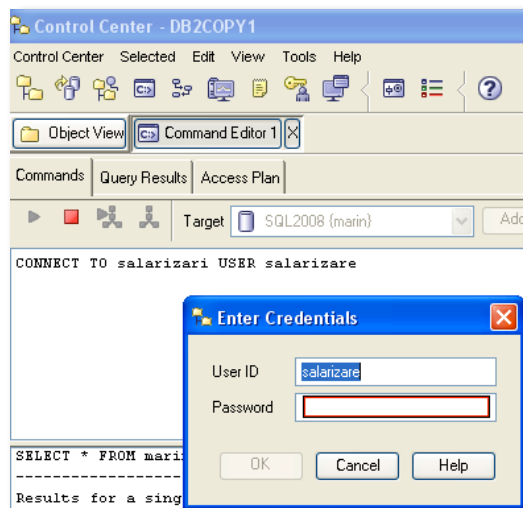


Figura 15.3. Solicitarea parolei pentru utilizatorul declarat odată cu subschema

Explicația fenomenului paranormal este că utilizatorii DB2 sunt preluați din sistemul de operare, așa că degeaba îi declarăm noi în premieră în comenzi CREATE SCHEMA. Așadar, trebuie să declarăm în Windows XP (acesta este sistemul de operare de pe calculatorul meu) un cont pentru utilizatorul *salarizare*, pe filiera *Control Panel* → *User Account*. Pentru aceasta, aveți nevoie de drepturi serioase în Windows. Contul utilizatorului nou creat nu trebuie să fie neapărat înzestrat cu drepturi de administrator.

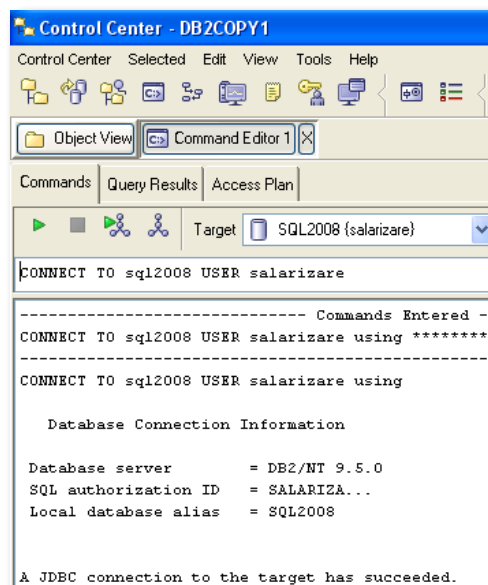


Figura 15.4. Conectarea ca utilizator salarizare, după crearea contului în Windows

Parola din Windows va fi furnizată acum la re-conectarea cu serverul DB2 (re-execuția comenzii CONNECT de mai sus). Dacă ținem minte parola indicată la crearea contului în sistemul de operare, sunt șanse mari să avem un mesaj plăcut, de genul celui din figura 15.4. După conectare însă, accesul la tabelele din subschema *MARIN* este condiționat, pe de o parte, de drepturile de acces ale utilizatorului *salarizare* și, pe de altă parte, de prefixarea numelui tabelului cu denumirea subschemei în care se află. Astfel, interogarea:

```
SELECT * FROM personal2
```

nu funcționează, tabela *PERSONAL2* nefiind recunoscută (mesajul de eroare este: *SQL0204N "SALARIZARE.PERSONAL2" is an undefined name*). Varianta corectă a interogării este:

```
SELECT * FROM marin.personal2
```

După cum vă amintiți, prin două comenzi GRANT am dat drepturi depline utilizatorului *salarizare* doar la două tabele din schema *MARIN*, *PERSONAL2* și *SPORURI*. Orice încercare de a interoga alte tabele din schema *MARIN* sunt sortite eșecului – vezi figura 15.5:

```
SELECT * FROM marin.facturi
```

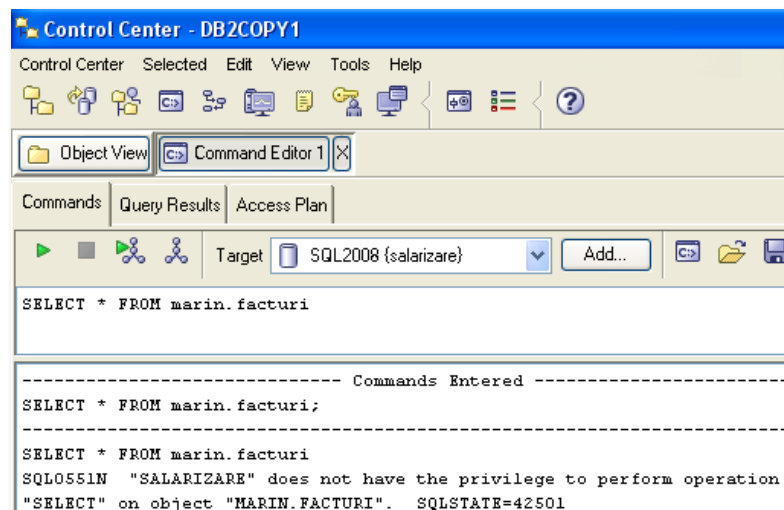


Figura 15.5. Acces interzis la o tabelă la care utilizatorul curent nu are drepturi (deloc)

15.1.2. Utilizatori și subscheme în Oracle

În Oracle legătura dintre schemă și utilizator este foarte strânsă. Utilizatorii pot fi creați mult mai simplu decât în DB2, prin comanda *CREATE USER* sau interactiv. La declarare, fiecărui *USER* i se alocă implicit o sub-schemă proprie în care vor fi grupate toate obiectele pe care le va crea. Pot crea noi conturi-utilizator numai cei care au acest drept. Astfel, după conectarea ca super-utilizator (*SYS*), sau orice alt utilizator cu drepturi corespunzătoare, putem crea contul dedicat

utilizatorului SALARIZARE și apoi să acordăm câteva drepturi (minimale) prin secvența de comenzi:

```
CREATE USER salarizare IDENTIFIED BY salarizare ;
```

```
GRANT CONNECT, RESOURCE, CREATE SESSION TO salarizare ;
```

```
GRANT CREATE SESSION TO salarizare ;
```

```
GRANT ALL ON sql2008.personal2 TO salarizare;
```

```
GRANT ALL ON sql2008.sporuri TO salarizare;
```

Iată cum arată dialogul în figura 15.6. Crearea utilizatorilor poate fi făcută și interactiv, alegând din meniul arborescent din stânga (jos) a figurii opțiunea *Other Users*. Privilegiul (dreptul) CREATE SESSION a fost acordat de două ori utilizatorului *salarizare*, fără ca repetarea să deranjeze.

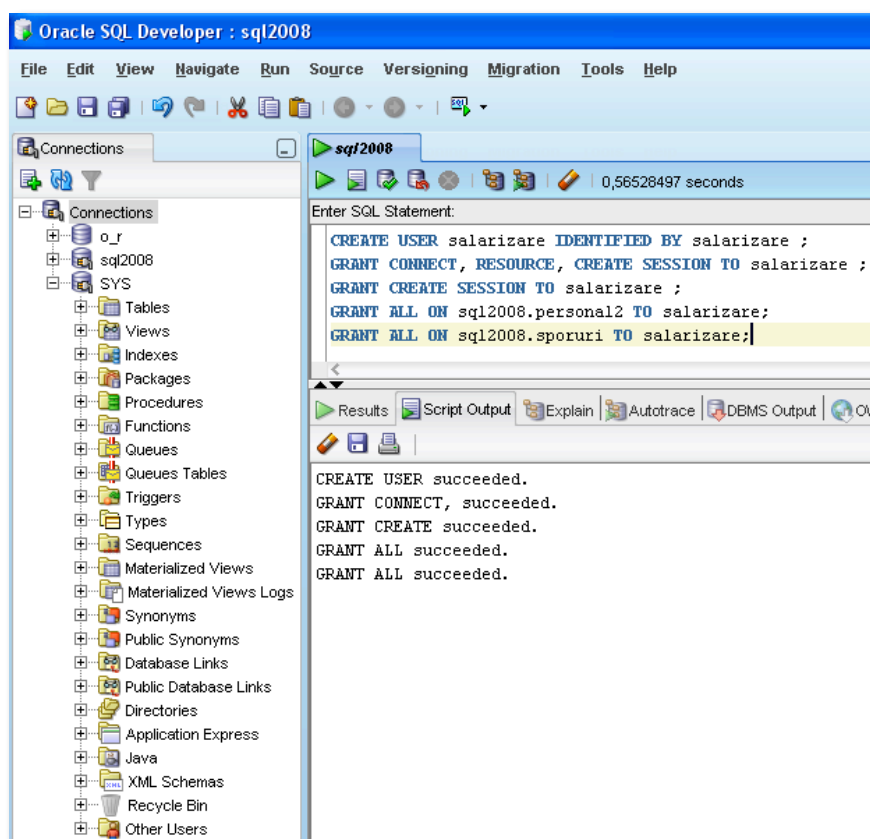


Figura 15.6. Crearea unui utilizator și acordarea de drepturi în Oracle SQL Developer

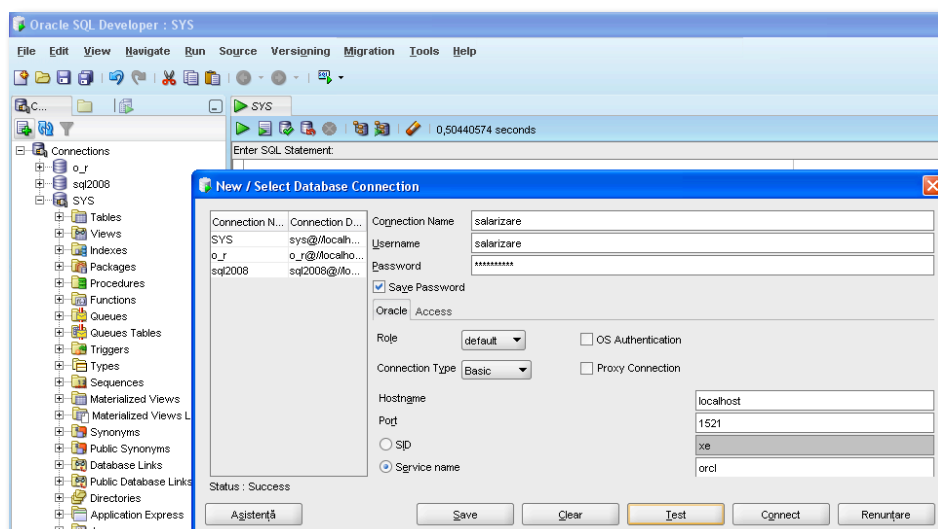


Figura 15.7. Crearea unei conexiuni pentru utilizatorul/schema salarizare

Pentru a lucra efectiv în această (sub)schemă, definim o conexiune nouă. Pentru simplitate, numele conexiunii va fi același ca numele subschemei și utilizatorului. Cum eu lucrez cu serverul Oracle (la fel și cu celelalte trei servere BD) instalat pe un laptop pentru această carte, la rubrica *Host name* apare *localhost* iar numele serviciului (*Service name*) este cel implicit de la instalare (*orcl*). Formularul complet pentru declararea conexiunii este cel din figura 15.7. În practică serverul BD este instalat pe o „mașină” specială, iar rubricile *Host name*, *Service name* etc. pot prezenta diferențe sensibile față de situația noastră.

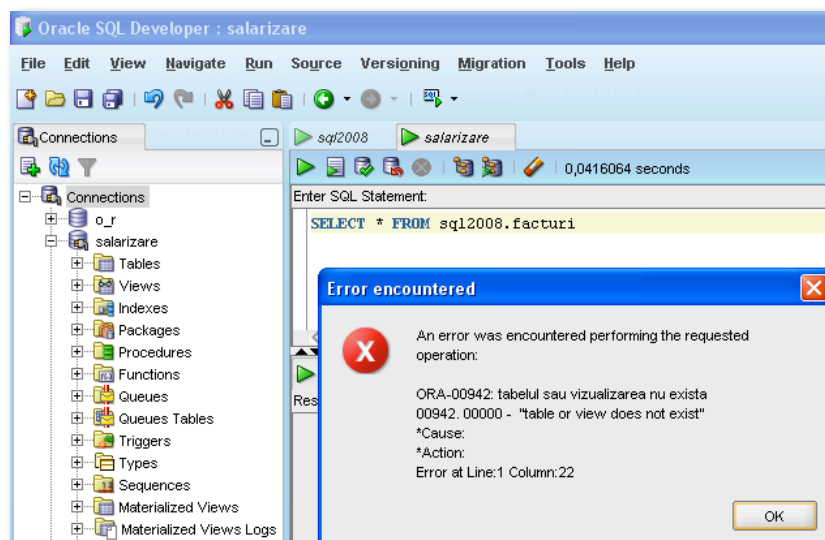


Figura 15.8. Interzicerea accesului la o tabelă neautorizată

După crearea conexiunii și deschiderea sa, accesul se poate face doar la tabelele la care utilizatorul *salarizare* are acces, prefixând numele obiectelor cu schema din care fac parte. Dacă în DB2 numele subschemei în care am lucrat până în acest capitol era *MARIN*, în Oracle subschema se numește *SQL2008*.

```
SELECT * FROM sql2008.personal2;
```

Ca și în DB2, încercarea de a consulta alte tabele decât cele două la care au fost definite drepturi este descurajată – vezi figura 15.8.

15.1.3. Utilizatori și subscheme în PostgreSQL

Specific PostgreSQL este echivalența dintre comenzile CREATE USER și CREATE ROLE. Pot crea conturi pentru noi utilizatori numai *superuser*-ii. Utilizatorul în numele căruia am lucrat până în acest paragraf se numește *postgres*. Vom crea noul user – *salarizare* – prin comanda CREATE USER, deși puteam lucra și interactiv (opțiunea *Login Roles* din meniul arborescent pgAdmin – vezi figura 15.9):

```
CREATE USER salarizare LOGIN PASSWORD 'salarizare';
```

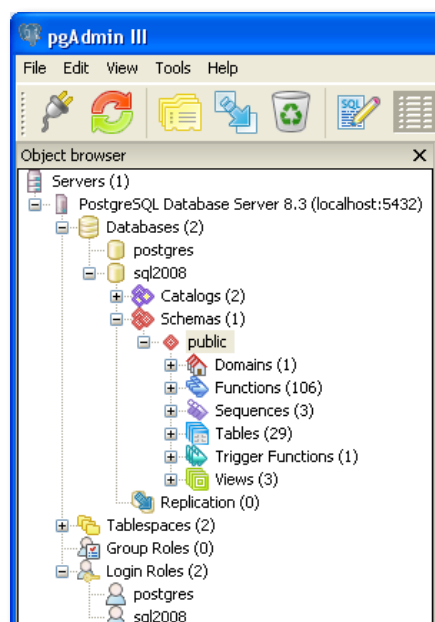


Figura 15.9. Meniul arborescent pgAdmin

Tot din figură aflăm numele schemei în care am lucrat până acum – *public* – din baza de date *sql2008*. Din paragraful 6.6 (și figura 6.44) știm că avem câteva funcții PostgreSQL prin care aflăm, printre altele, numele bazei de date și schemei curente (CURRENT_DATABASE() și CURRENT_SCHEMA()).

Creăm în BD SQL2008, alături de *public*, subschema *salarizare*:

```
CREATE SCHEMA salarizare AUTHORIZATION salarizare ;
```

Drepturile depline la cele două tabele din schema *public* trebuie precedate de acordarea dreptului la nivel de (sub)schemă:

```
GRANT USAGE ON SCHEMA public TO salarizare ;
```

```
GRANT ALL PRIVILEGES ON public.personal2 TO salarizare;
```

```
GRANT ALL PRIVILEGES ON public.sporuri TO salarizare;
```

Pentru a testa modul în care funcționează mecanismul de acordare a drepturilor, creăm o conexiune dedicată utilizatorului *salarizare* – vezi figura 15.10. Numele conexiunii este același cu al utilizatorului. Similar Oracle, la rubrica *Host* se introduce *localhost*. Numele serviciului este opțional, iar bifarea opțiunii *Connect now* asigură activarea imediată a conexiunii.

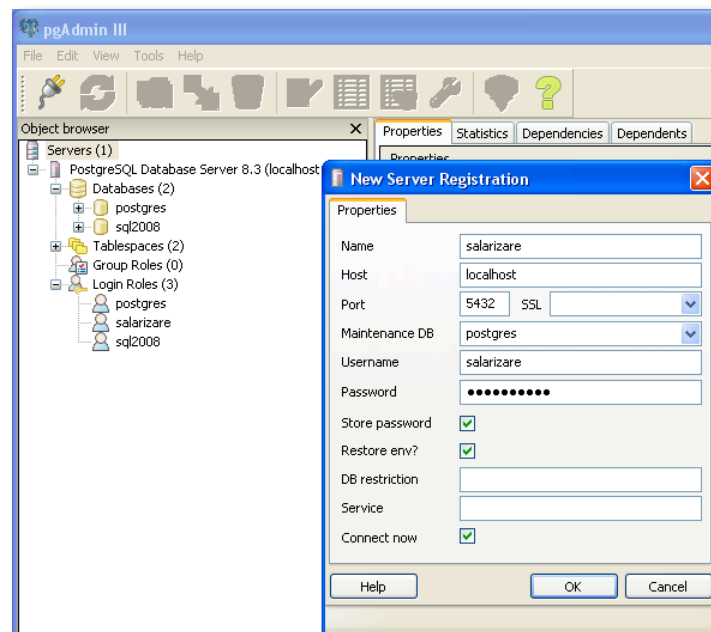


Figura 15.10. Crearea unei conexiuni (pgAdmin) pentru utilizatorul *salarizare*

După activarea conexiunii și selectarea schemei curente (vezi figura 15.11), știm că suntem conectați ca utilizator *salarizare* și vom avea drepturile de acces și creare/editare ale acestuia. Interogăm tabelele din subschema *public* fie prin oricare dintre variantele următoare:

```
SELECT * FROM sql2008.public.personal2 ;
```

```
SELECT * FROM public.personal2 ;
```

```
SELECT * FROM personal2 ;
```

Dacă însă încercăm să accesăm oricare altă tabelă decât cele două (PERSONAL2 și SPORURI) la care utilizatorul *salarizare* are drepturi, ne procopsim, cum era de așteptat, cu un mesaj de eroare de genul celui din figura 15.11.

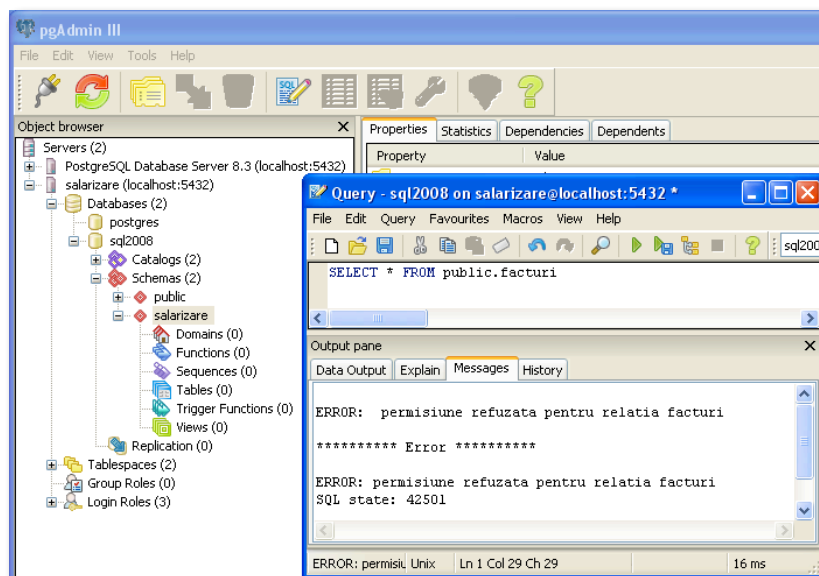


Figura 15.11. Blocarea accesului la o tabelă neautorizată în PostgreSQL

15.1.4. Utilizatori și subscheme în SQL Server

Și SQL Server are câteva particularități în materie de utilizatori, sub-scheme și drepturi. Mai întâi, trebuie să ținem cont de faptul că la definirea conexiunii folosite până acum am folosit autentificarea sistemului de operare. Aflarea bazei de date, schemei și utilizatorului curent(ă) presupune folosirea funcțiilor descrise în paragraful 6.6 – vezi și figura 15.12.

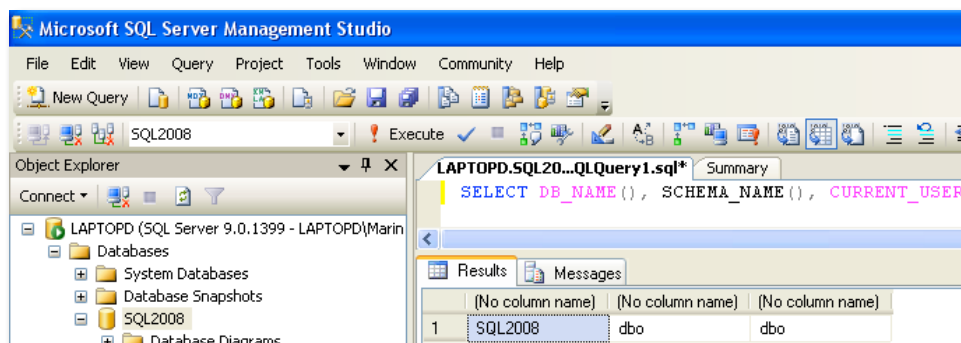


Figura 15.12. Aflarea bazei de date, schemei și utilizatorului curente/curent

O particularitate a SQL Server-ului ține de etapele parcurse pentru crearea unui cont-utilizator. Prima etapă constă în folosirea comenzii CREATE LOGIN:

```
CREATE LOGIN salarizare WITH PASSWORD = 'salarizare2008' ;
```

Numai după aceasta se poate lansa CREATE USER:

```
CREATE USER salarizare FOR LOGIN salarizare
```

Noului utilizator îi vom “repartiza” o schemă cu același nume, după formatul de acum cunoscut:

```
CREATE SCHEMA salarizare AUTHORIZATION salarizare ;
```

Și în privința acordării drepturilor (privilegiilor) SQL Server are câteva diferențe. Vom folosi în comandă opțiunea ALL, care, în ciuda numelui, se referă doar la operațiunile: DELETE, INSERT, REFERENCES, SELECT și UPDATE:

```
GRANT ALL PRIVILEGES ON dbo.personal2 TO salarizare;
```

```
GRANT ALL PRIVILEGES ON dbo.sporuri TO salarizare;
```

Pentru a verifica modul de respectare a drepturilor acordate, vom crea o nouă conexiune în care vom folosi metoda de autentificare a serverului BD – vezi figura 15.13.

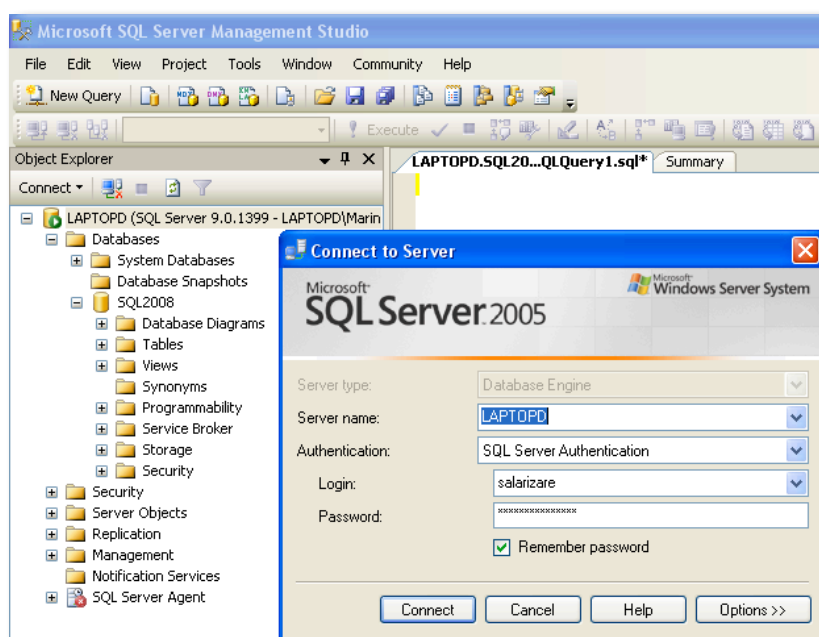


Figura 15.13. Crearea unei (alte) conexiuni SQL Server

După activarea noii conexiuni, o să vedem (figura 15.14) că nu există schemă distinctă salarizare, așa cum ne-am fi așteptat. În schimb, meniul arborescent afișează pentru baza de date (SQL2008) numai cele două tabele la care utilizatorul *salarizare* are acces. Încercarea de a interoga orice altă tabelă decât cele două este sortită eșecului.

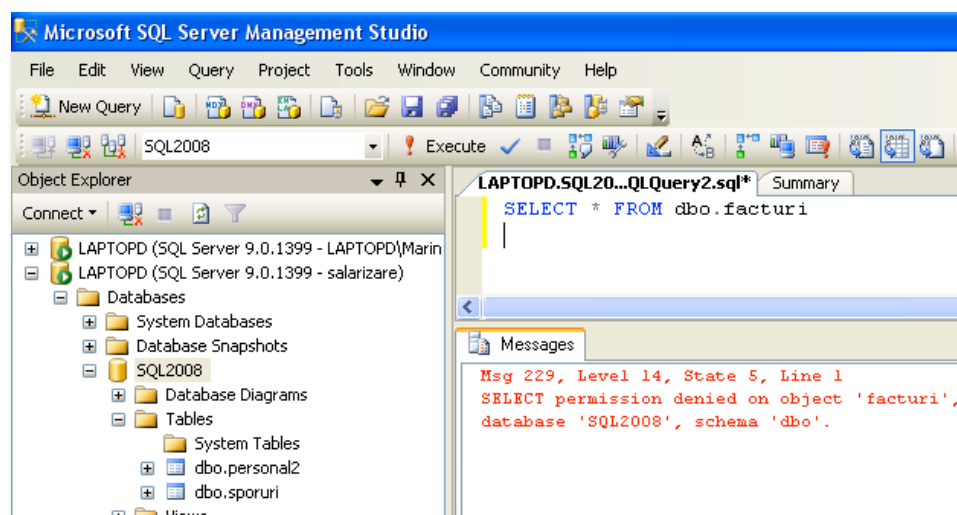


Figura 15.14. Afișarea tabelelor la care are acces utilizatorul *salarizare*

15.2. Informații despre obiectele din (sub)scheme

Programele-client cu ajutorul cărora ne putem conecta la bazele de date, cum sunt SQL Developer și pgAdmin, sunt destul de generoase în afișarea obiectelor dintr-o (subschemă) a bazei. Ne amintim din primul capitol că întreaga structură a unei BD se păstrează în *dicționarul de date*, numit și *catalog-sistem* sau *container de date* etc. Structura și conținutul acestui dicționar prezintă diferențe sensibile de la produs la produs, însă organizarea sa este tot tabelară. Deși vorbim de *dicționar de date*, în practică acesta este fragmentat într-o puzderie de tabele și tabele virtuale care, firește, pot fi interogate prin SELECT-uri și uneori chiar modificate.

Standardele SQL (începând cu SQL-92)¹ fac trimitere la două (sub)scheme speciale din orice bază de date, INFORMATION_SCHEMA și DEFINITION_SCHEMA². Prima este compusă din tabele virtuale (*views*) ce furnizează informații despre obiectele persistente ale bazei, în timp ce a doua este alcătuită din tabele „obișnuite” ce conțin informații despre modul de organizare al primeia. Dicționarul de date este deci denumit în SQL *catalog*, catalog ce reprezintă o colecție de (sub)scheme printre care se regăsesc și cele două menționate.

Fiecare tabelă virtuală sau tabelă din catalog este specializată în anumite categorii de informații³. Iată câteva exemple:

¹ Nu uitați că, în afara primelor două părți generale, există o parte a standardului SQL numită *SQL/Schemata*.

² [Kriegel & Trukhnov 2008], p.495

³ Vezi [Kriegel & Trukhnov 2008], pp.496-500

- TABLES – informații generale despre tabele;
- COLUMNS – informații despre atribute;
- TABLE_CONSTRAINTS – informații despre restricțiile definite în tabele;
- CHECK_CONSTRAINTS – informații numai despre restricțiile definite prin clauza CHECK (reguli de validare la nivel de atribut și înregistrare);
- REFERENTIAL_CONSTRAINTS – informații numai despre restricțiile referențiale;
- SCHEMATA – subschemele dintr-o bază de date;
- VIEWS – informații despre tabelele virtuale.

Din păcate denumirile lor sunt cât de poate de eterogene de la SGBD la SGBD, după cum vom vedea în continuare.

15.2.1. Obiecte din schema unei baze de date în PostgreSQL

Începem cu descrierea câtorva informații din dicționarul de date PostgreSQL. Punctul de plecare în aflarea pe cale “interogațională” a informațiilor din schema unei baze de date PostgreSQL este tabela-sistem PG_TABLES:

SELECT * FROM pg_tables

schemaname name	tablename name	tableowner name	tablespace name	hasindexes boolean	hasrules boolean	hastriggers boolean
information_schema	sql_features	postgres		f	f	f
information_schema	sql_implementation_info	postgres		f	f	f
pg_catalog	pg_statistic	postgres		t	f	f
information_schema	sql_languages	postgres		f	f	f
information_schema	sql_packages	postgres		f	f	f
information_schema	sql_parts	postgres		f	f	f
information_schema	sql_sizing	postgres		f	f	f
pg_catalog	pg_authid	postgres	pg_global	t	f	t
information_schema	sql_sizing_profiles	postgres		f	f	f
pg_catalog	pg_database	postgres	pg_global	t	f	t
pg_catalog	pg_tablespace	postgres	pg_global	t	f	f
pg_catalog	pg_pltemplate	postgres	pg_global	t	f	f
pg_catalog	pg_shdepend	postgres	pg_global	t	f	f
pg_catalog	pg_shdescription	postgres	pg_global	t	f	f
pg_catalog	pg_ts_config	postgres		t	f	f
pg_catalog	pg_ts_config_map	postgres		t	f	f
pg_catalog	pg_ts_dict	postgres		t	f	f
pg_catalog	pg_ts_parser	postgres		t	f	f
pg_catalog	pg_ts_template	postgres		t	f	f
pg_catalog	pg_auth_members	postgres	pg_global	t	f	t
pg_catalog	pg_type	postgres		t	f	f
pg_catalog	pg_attribute	postgres		t	f	f
pg_catalog	pg_proc	postgres		t	f	f

Figura 15.15. Tabela centrală a dicționarului de date în PostgreSQL

O parte din conținutul său este afișată în figura 15.15. Informațiile sunt dispuse în trei subscheme, INFORMATION_SCHEMA, PG_CATALOG și PUBLIC (care nu a încăput în figură).

Tabelele-dicționar din (sub)schema INFORMATION_SCHEMA sunt de interes general, cum ar fi SQL_PACKAGES care informează despre modul de implementare în versiunea curentă de PostgreSQL a componentelor din standardul SQL (vezi figura 15.16):

```
SELECT * FROM information_schema.sql_packages
```

feature_id character varying	feature_name character varying	is_supported boolean	is_verified_by character varying	comments text
PKG000	Core	NO		
PKG001	Enhanced datetime facilities	YES		
PKG002	Enhanced integrity management	NO		
PKG003	OLAP facilities	NO		
PKG004	PSM	NO		PL/pgSQL is similar.
PKG005	CLI	NO		ODBC is similar.
PKG006	Basic object support	NO		
PKG007	Enhanced object support	NO		
PKG008	Active database	NO		
PKG010	OLAP	NO		NO

Figura 15.16. Aflarea modului de implementare în PostgreSQL a componentelor din standardul SQL

sau SQL_FEATURES care furnizează răspunsul la întrebarea *Care dintre opțiunile standardului SQL sunt implementate și care nu* etc. În subschema PUBLIC se află tabelele create de utilizator (adică de noi) până la momentul curent. Dacă le-ați uitat cumva, le putem obține foarte simplu (vezi figura 15.17):

```
SELECT * FROM pg_tables WHERE schemaname='public' ORDER BY tablename
```

schemaname name	tablename name	tableowner name	tablespace name	hasindexes boolean	hasrules boolean	hastriggers boolean
public	clienti	postgres		t	f	t
public	coduri_postale	postgres		t	f	t
public	curse	postgres		f	f	f
public	curse_statii	postgres		f	f	f
public	distante	postgres		t	f	f
public	doctori	postgres		t	f	t
public	facturi	postgres		t	f	t
public	facturi_2006	postgres		f	f	f
public	facturi_2007	postgres		f	f	f
public	garzi	postgres		t	f	t
public	incasari	postgres		t	f	t
public	incasfact	postgres		t	f	t
public	judete	postgres		t	f	t
public	liniifact	postgres		t	f	t
public	liniifact_2007	postgres		f	f	f
public	pacienti	postgres		t	f	t

Figura 15.17. O parte dintre tabelele create de utilizator (adică de noi)

Detaliile despre celelalte ingrediente ale structurii bazei – atribute, restricții, proceduri stocate, declanșatoare etc. se găsesc în tabele specializate ale dicționarului, tabele aflate în schema PG_CATALOG. Conținutul acestora este, de cele mai multe ori derutant. Spre exemplu, dacă dorim să aflăm ceva mai multe despre

atributele din tabelele create în capitolele anterioare, vom începe prin a scotoci în PG_ATTRIBUTE:

```
SELECT * FROM pg_catalog.pg_attribute
```

După cum se poate zări din figura 15.18, conținutul acestei tabele este devastator, aici zăcând, de-a valma, atributele „noastre” cu attribute din tabele sistem, greu numărabile și comprehensibile (sesizați, vă rog, termenul elevat!).

	attrelid oid	attname name	atttypid oid	attstattarget integer	attlen smallint	attnum smallint	attndims integer	attcacheoff integer	atttypm integer
2237	26117	tableoid	26	0	4	-7	0	-1	-1
2238	26119	chunk_id	26	-1	4	1	0	-1	-1
2239	26119	chunk_seq	23	-1	4	2	0	-1	-1
2240	17785	valtotala	1700	-1	-1	5	0	-1	655366
2241	17785	tva	1700	-1	-1	6	0	-1	655366
2242	17785	valincasata	1700	-1	-1	7	0	-1	655366
2243	17785	reduceri	1700	-1	-1	8	0	-1	589830
2244	17785	penalizari	1700	-1	-1	9	0	-1	589830
2245	17797	tvalinie	1700	-1	-1	6	0	-1	589830
2246	26178	jud	1042	-1	-1	1	0	-1	6
2247	26178	judet	1043	-1	-1	2	0	-1	29
2248	26178	regiune	1043	-1	-1	3	0	-1	19

Figura 15.18. O (mică) parte din tabela(virtuală)-dicționar PG_CATALOG.PG_ATTRIBUTE

Filtrarea atributelor după numele tabeli din care fac parte e ceva mai laborioasă. Ne bazăm pe coloana *Attrelid* (prima din figura 15.18) care conține identificatorul tabeli. Indicatorul și numele tabeli sunt conținute în tabela-dicționar PG_CLASS, așa că interogarea va avea forma:

```
SELECT * FROM pg_attribute
```

```
WHERE attrelid = (SELECT oid FROM pg_class WHERE relname='facturi')
```

Însă tot n-am scăpat de atributele-sistem ale tabeli (vezi figura 15.19). O soluție ar fi, totuși. Numărul fiecărui atribut din tabelă - *Attnum* - este pozitiv, dacă se numără printre cele definite de utilizator, și negativ, dacă este unul sistem:

	attrelid oid	attname name	atttypid oid	attstattarget integer	attlen smallint	attnum smallint	attndims integer	attcacheoff integer	atttypmod integer	attby boole
1	17785	tableoid	26	0	4	-7	0	-1	-1	t
2	17785	cmax	29	0	4	-6	0	-1	-1	t
3	17785	xmax	28	0	4	-5	0	-1	-1	t
4	17785	cmin	29	0	4	-4	0	-1	-1	t
5	17785	xmin	28	0	4	-3	0	-1	-1	t
6	17785	ctid	27	0	6	-1	0	-1	-1	f
7	17785	nrfact	23	-1	4	1	0	-1	-1	t
8	17785	datafact	1082	-1	4	2	0	-1	-1	t
9	17785	codcl	21	-1	2	3	0	-1	-1	t
10	17785	obs	1043	-1	-1	4	0	-1	54	f
11	17785	valtotala	1700	-1	-1	5	0	-1	655366	f
12	17785	tva	1700	-1	-1	6	0	-1	655366	f
13	17785	valincasata	1700	-1	-1	7	0	-1	655366	f
14	17785	reduceri	1700	-1	-1	8	0	-1	589830	f
15	17785	penalizari	1700	-1	-1	9	0	-1	589830	f

Figura 15.19. Atributele tabeli FACTURI

```
SELECT * FROM pg_attribute
```

```

WHERE attrelid = (SELECT oid FROM pg_class WHERE relname='facturi')
AND attnum > 0
ORDER BY attnum

```

Pentru a afișa numele, tipul și valoarea implicită a fiecărui atribut din tabela FACTURI (figura 15.20), este necesar să recurgem și la tabelele virtuale-dicționar PG_TYPE și PG_ATTRDEF:

```

SELECT attname AS NumeAtribut, typname AS tip,
      (SELECT adsr FROM pg_attrdef WHERE (adrelid,adnum) = (attrelid,attnum))
      AS val_implicita
FROM pg_attribute INNER JOIN pg_type ON pg_attribute.atttypid = pg_type.typelem
WHERE attrelid = (
      SELECT oid FROM pg_class WHERE relname='facturi'
    ) AND attnum>0
ORDER BY attnum

```

numeatribut name	tip name	val_implicita text
nrifact	_int4	
datafact	_date	('now'::text)::date
codcl	int2vector	
codcl	_int2	
obs	_varchar	
valtotala	_numeric	0
tva	_numeric	0
valincasata	_numeric	0
reduceri	_numeric	0
penalizari	_numeric	0

Figura 15.20. Informații despre atributele tabeli FACTURI aflate din view-urile-dicționar *pg_attribute*, *pg_type* și *pg_attrdef*

Mult mai simplu este, însă, să folosim tabela virtuală *COLUMNS* din (sub)schema *INFORMATION_SCHEMA*:

```

SELECT column_name, data_type,
      CASE
        WHEN data_type LIKE 'char%' THEN character_maximum_length || ' '
        WHEN data_type='numeric' THEN numeric_precision
        || '(' || numeric_scale || ')' END AS Lungime,
      is_nullable
FROM information_schema.columns
WHERE table_catalog='sql2008' AND table_schema='public' AND table_name='facturi'
ORDER BY ordinal_position

```

Cu această versiune nu numai că suntem mai aproape de sintaxa standardului, dar și modul în care se prezintă informațiile este parcă mai arătos – vezi figura 15.21.

column_name character varying	data_type character var	lungime text	is_nullable character va
nrfact	integer		NO
datafact	date		YES
codcl	smallint		YES
obs	character varyin	50	YES
valtotala	numeric	10(2)	YES
tva	numeric	10(2)	YES
valincasata	numeric	10(2)	YES
reduceri	numeric	9(2)	YES
penalizari	numeric	9(2)	YES

Figura 15.21. Informații despre atribute aflate din view-ul *information_schema.columns*

Afișarea regulilor de validare (clauzele CHECK) dintr-o tabelă - vezi figura 15.22 - reclamă folosirea a două noi tabele virtuale dicționar: CHECK_CONSTRAINTS și CONSTRAINT_COLUMN_USAGE.

SELECT DISTINCT

```
(SELECT ordinal_position FROM information_schema.columns
WHERE table_catalog='sql2008' AND table_schema='public' AND
table_name='facturi' AND column_name=ccu.column_name)
AS NrCrt,
```

column_name,

```
(SELECT check_clause FROM information_schema.check_constraints
WHERE constraint_name=ccu.constraint_name) AS clauza_CHECK
```

FROM information_schema.constraint_column_usage ccu

WHERE table_catalog='sql2008' AND table_schema='public' AND table_name='facturi'

ORDER BY 1

nrcrt integer	column_name character vary	clauza_check character varying
1	nrfact	
2	datafact	(((datafact >= to_date('2007-01-01'::text, 'YYYY-MM-DD'::text)) AND (datafact <= '2015-12-31'::date)))

Figura 15.22. Restricțiile de tip CHECK definite în tabela FACTURI

Cea mai simplă modalitate de a obține toate restricțiile definite într-o tabelă presupune interogarea tabeli virtuale TABLE_CONSTRAINTS. Iată restricțiile tabeli LINIIFACT (vezi figura 15.23):

SELECT *

FROM information_schema.table_constraints

WHERE table_name='liniifact'

O parte dintre cheile primare (PRIMARY KEY) și alternative (UNIQUE) sunt compuse, așa că, pentru a afla expresiile (atributele din componența cheilor) apelăm la joncțiunea externă repetată a tabeli virtuale TABLE_CONSTRAINTS cu KEY_COLUMN_USAGE:

constraint_catalog character varying	constraint_schema character varying	constraint_name character varying	table_catalog character varying	table_schema character varying	table_name character varying	constraint_type character varying	is_deferrable character varying	initially_deferred character varying
sql2008	public	2200_17797_1_not_null	sql2008	public	liniifact	CHECK	NO	NO
sql2008	public	2200_17797_2_not_null	sql2008	public	liniifact	CHECK	NO	NO
sql2008	public	2200_17797_3_not_null	sql2008	public	liniifact	CHECK	NO	NO
sql2008	public	2200_17797_4_not_null	sql2008	public	liniifact	CHECK	NO	NO
sql2008	public	2200_17797_5_not_null	sql2008	public	liniifact	CHECK	NO	NO
sql2008	public	ck_linie	sql2008	public	liniifact	CHECK	NO	NO
sql2008	public	fk_liniifact_facturi	sql2008	public	liniifact	FOREIGN KEY	NO	NO
sql2008	public	fk_liniifact_produce	sql2008	public	liniifact	FOREIGN KEY	NO	NO
sql2008	public	pk_liniifact	sql2008	public	liniifact	PRIMARY KEY	NO	NO
sql2008	public	un_liniifact	sql2008	public	liniifact	UNIQUE	NO	NO

Figura 15.23. Restricții definite în tabela LINIIFACT

```

SELECT tc.constraint_name, '(' || ku1.column_name ||
CASE WHEN ku2.column_name IS NULL THEN "
ELSE ',' || ku2.column_name END
|| CASE WHEN ku3.column_name IS NULL THEN "
ELSE ',' || ku3.column_name END
|| CASE WHEN ku4.column_name IS NULL THEN "
ELSE ',' || ku4.column_name END
|| ')' AS expresie_restricție
FROM information_schema.table_constraints tc
INNER JOIN information_schema.key_column_usage ku1
ON tc.constraint_name=ku1.constraint_name
AND ku1.ordinal_position = 1
LEFT OUTER JOIN information_schema.key_column_usage ku2
ON tc.constraint_name=ku2.constraint_name
AND ku2.ordinal_position = 2
LEFT OUTER JOIN information_schema.key_column_usage ku3
ON tc.constraint_name=ku3.constraint_name
AND ku3.ordinal_position = 3
LEFT OUTER JOIN information_schema.key_column_usage ku4
ON tc.constraint_name=ku4.constraint_name
AND ku4.ordinal_position = 4
WHERE tc.table_name='liniifact'

```

Interogarea de mai sus se bazează pe faptul că în tabelele bazei de date VÎNZĂRI orice cheie primară, alternativă sau străină nu are mai mult de patru atribute (de fapt, numărul maxim este trei, dar parcă ar fi prea simplu). Rezultatul este cel din figura 15.24.

constraint_name character varying	expresie_restricție text
fk_liniifact_produce	(codpr)
fk_liniifact_facturi	(nrfact)
un_liniifact	(nrfact,codpr)
pk_liniifact	(nrfact,linie)

Figura 15.24. Expresiile cheii primare, alternative și ale celor străine din tabela LINIIFACT

Expresia cheilor străine este incompletă, deoarece lipsesc numele tabelului părinte și atributele din cheia primară/alternativă a tabelului părinte. Așa că intră în scenă o altă tabelă virtuală din INFORMATION_SCHEMA, și anume REFERENTIAL_CONSTRAINTS (vezi figura 15.25):

SELECT * FROM information_schema.referential_constraints

constraint_character	constraint_character	constraint_name_character_varying	unique_character	unique_character	unique_constraint_character_varying	match_option_character_varying	update_rule_character_varying	delete_rule_character_varying
sql2008	public	fk_personal2	sql2008	public	pk_personal2	NONE	NO ACTION	NO ACTION
sql2008	public	fk_sporuri_pers2	sql2008	public	pk_personal2	NONE	NO ACTION	NO ACTION
sql2008	public	fk_coduri_post_jud	sql2008	public	pk_judete	NONE	CASCADE	RESTRICT
sql2008	public	fk_persoane_cp	sql2008	public	pk_coduri_post	NONE	CASCADE	RESTRICT
sql2008	public	fk_perscienti_clienti	sql2008	public	pk_clienti	NONE	CASCADE	RESTRICT
sql2008	public	fk_perscienti_persoane	sql2008	public	pk_persoane	NONE	CASCADE	RESTRICT
sql2008	public	garzi_iddoctor_fkey	sql2008	public	pk_doctori	NONE	CASCADE	NO ACTION
sql2008	public	fk_internari_pacienti	sql2008	public	pk_pacienti	NONE	CASCADE	NO ACTION
sql2008	public	fk_clienti_cp	sql2008	public	pk_coduri_post	NONE	CASCADE	RESTRICT
sql2008	public	fk_incasfact_facturi	sql2008	public	pk_facturi	NONE	CASCADE	RESTRICT
sql2008	public	fk_incasfact_incasari	sql2008	public	pk_incasari	NONE	CASCADE	RESTRICT
sql2008	public	fk_liniifact_produce	sql2008	public	pk_produce	NONE	CASCADE	RESTRICT
sql2008	public	fk_liniifact_facturi	sql2008	public	pk_facturi	NONE	CASCADE	RESTRICT
sql2008	public	fk_facturi_clienti	sql2008	public	pk_clienti	NONE	CASCADE	NO ACTION

Figura 15.25. Conținutul view-ului *information_schema.referential_constraints*

Deși în baza de date toate cheile străine sunt simple, formulăm o soluție care are funcționa chiar și dacă am avea chei străine alcătuite din patru atribute:

```
SELECT tc.table_name, constraint_type || ' (' || kcu1.column_name ||
CASE WHEN kcu2.column_name IS NULL THEN " ELSE ',' || kcu2.column_name
END ||
CASE WHEN kcu3.column_name IS NULL THEN " ELSE ',' || kcu3.column_name
END ||
CASE WHEN kcu4.column_name IS NULL THEN " ELSE ',' || kcu4.column_name
END ||') REFERENCES ' || (
SELECT tc2.table_name || ' (' || kcu11.column_name ||
CASE WHEN kcu12.column_name IS NULL THEN "
ELSE ',' || kcu12.column_name END ||
CASE WHEN kcu13.column_name IS NULL THEN "
ELSE ',' || kcu13.column_name END ||
CASE WHEN kcu14.column_name IS NULL THEN "
ELSE ',' || kcu14.column_name END ||')'
FROM information_schema.table_constraints tc2
INNER JOIN information_schema.key_column_usage kcu11
ON tc2.constraint_name=kcu11.constraint_name
AND kcu11.ordinal_position =1
LEFT OUTER JOIN information_schema.key_column_usage kcu12
ON tc2.constraint_name=kcu12.constraint_name
AND kcu12.ordinal_position = 2
```

```

        LEFT OUTER JOIN information_schema.key_column_usage kcu13
            ON tc2.constraint_name=kcu13.constraint_name
            AND kcu13.ordinal_position = 3
        LEFT OUTER JOIN information_schema.key_column_usage kcu14
            ON tc2.constraint_name=kcu14.constraint_name
            AND kcu14.ordinal_position = 4
    WHERE tc2.constraint_name = rc.unique_constraint_name
    ) || CASE update_rule WHEN 'NO ACTION' THEN "
    ELSE ' ON UPDATE ' || update_rule END ||
    CASE delete_rule WHEN 'NO ACTION' THEN "
    ELSE ' ON DELETE ' || delete_rule END
    AS "Expresie_completa_FOREIGN_KEY"
FROM (information_schema.table_constraints tc
    INNER JOIN information_schema.key_column_usage kcu1
        ON tc.constraint_name=kcu1.constraint_name
        AND kcu1.ordinal_position = 1
    LEFT OUTER JOIN information_schema.key_column_usage kcu2
        ON tc.constraint_name=kcu2.constraint_name
        AND kcu2.ordinal_position = 2
    LEFT OUTER JOIN information_schema.key_column_usage kcu3
        ON tc.constraint_name=kcu3.constraint_name
        AND kcu3.ordinal_position = 3
    LEFT OUTER JOIN information_schema.key_column_usage kcu4
        ON tc.constraint_name=kcu4.constraint_name
        AND kcu4.ordinal_position = 4)
    INNER JOIN information_schema.referential_constraints rc
        ON tc.constraint_name=rc.constraint_name
WHERE constraint_type='FOREIGN KEY'
ORDER BY tc.table_name

```

Sunt extrase toate cheile străine din schema *public* - vezi figura 15.26. Ca o confirmare a corectitudinii interogării (nu că s-ar fi îndoit careva!), pe linia 12 apare și restricția referențială în care cheia străină și cheia primară corespondentă se află în aceeași tabelă - PERSONAL2.

table_name character var	Expresie_completa_FOREIGN_KEY text
clienti	FOREIGN KEY (codpost) REFERENCES coduri_postale (codpost) ON UPDATE CASCADE ON DELETE RESTRICT
coduri_postale	FOREIGN KEY (jud) REFERENCES judete (jud) ON UPDATE CASCADE ON DELETE RESTRICT
facturi	FOREIGN KEY (codcl) REFERENCES clienti (codcl) ON UPDATE CASCADE
garzi	FOREIGN KEY (iddoctor) REFERENCES doctori (iddoctor) ON UPDATE CASCADE
incasfact	FOREIGN KEY (codinc) REFERENCES incasari (codinc) ON UPDATE CASCADE ON DELETE RESTRICT
incasfact	FOREIGN KEY (nrifact) REFERENCES facturi (nrifact) ON UPDATE CASCADE ON DELETE RESTRICT
liniifact	FOREIGN KEY (codpr) REFERENCES produse (codpr) ON UPDATE CASCADE ON DELETE RESTRICT
liniifact	FOREIGN KEY (nrifact) REFERENCES facturi (nrifact) ON UPDATE CASCADE ON DELETE RESTRICT
persclienti	FOREIGN KEY (cnp) REFERENCES persoane (cnp) ON UPDATE CASCADE ON DELETE RESTRICT
persclienti	FOREIGN KEY (codcl) REFERENCES clienti (codcl) ON UPDATE CASCADE ON DELETE RESTRICT
persoane	FOREIGN KEY (codpost) REFERENCES coduri_postale (codpost) ON UPDATE CASCADE ON DELETE RESTRICT
personal2	FOREIGN KEY (marcasef) REFERENCES personal2 (marca)
sporuri	FOREIGN KEY (marca) REFERENCES personal2 (marca)
triai	FOREIGN KEY (idpacient) REFERENCES pacienti (idpacient) ON UPDATE CASCADE

Figura 15.26. Expresiile complete ale cheilor străine

15.2.2. Obiecte din schema unei baze de date în Oracle

Modalitățile sub care se prezintă dicționarul de date în Oracle au fost tratate ceva mai pe larg în lucrarea [Fotache s.a. 2003]⁴. Toate tabele și tabelele virtuale din dicționar aparțin (super)utilizatorului SYS, iar pentru simplificarea accesului, fiecare are un sinonim. De exemplu, în loc de SYS.DBA_TABLES se poate folosi sinonimul DBA_TABLES. Dacă numele view-urilor-dicționar este prefixat de USER, atunci este vorba de obiecte ce aparțin utilizatorului curent; ALL desemnează, în plus, și obiectele la care utilizatorul curent are acces, dar proprietarul este un alt utilizator, iar prefixul DBA (Data Base Administrator) se referă la toate obiectele bazei, indiferent de "proprietar". Tabela principală a catalogului este numită *DICTIONARY* și are doar două coloane, *TABLE_NAME* și *COMMENTS* (vezi figura 15.27). Aflarea tuturor tabelelor virtuale ce oferă informații despre obiectele din schemă este posibilă prin interogarea:

```
SELECT * FROM dictionary
```

Este posibil să aflăm orice informație despre schema bazei de date, atât despre obiecte de tipul celor create până în prezent (tabele, tabele virtuale, restricții, indecși), cât și de genul celor create de aici încolo (funcții, proceduri, pachete, declanșatoare, tipuri etc.). Totul e să știm (sau să nimerim) tabela virtuală corespunzătoare din dicționar. Pentru non-administratori, inventarierea obiectelor din schema proprie (tabele, tabele virtuale, sinonime, proceduri, funcții, pachete, secvențe etc.) este posibilă folosind tabela virtuală-dicționar *USER_OBJECTS*:

```
SELECT * FROM user_objects
```

⁴ Acesta este un moment publicitar

TABLE_NAME	COMMENTS
USER_RESOURCE_LIMITS	Display resource limit of the user
USER_PASSWORD_LIMITS	Display password limits of the user
USER_CATALOG	Tables, Views, Synonyms and Sequences owned by the user
ALL_CATALOG	All tables, views, synonyms, sequences accessible to the user
USER_CLUSTERS	Descriptions of user's own clusters
ALL_CLUSTERS	Description of clusters accessible to the user
USER_CLU_COLUMNS	Mapping of table columns to cluster columns
USER_COL_COMMENTS	Comments on columns of user's tables and views
ALL_COL_COMMENTS	Comments on columns of accessible tables and views
USER_COL_PRIVS	Grants on columns for which the user is the owner, grantor or grantee
ALL_COL_PRIVS	Grants on columns for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
USER_COL_PRIVS_MADE	All grants on columns of objects owned by the user
ALL_COL_PRIVS_MADE	Grants on columns for which the user is owner or grantor
USER_COL_PRIVS_RECV	Grants on columns for which the user is the grantee

Figura 15.27. O porțiune din tabela virtuală „centrală” din dicționarul bazei de date (Oracle)

Unele obiecte au informațiile presărate în mai multe tabele virtuale. De exemplu, în USER_TABLES ne-am aștepta să găsim o sumedenie de detalii despre tabele. Așa este (vezi figura 15.28), doar că interesați ar fi doar administratorii. De fapt, *view*-ul mai are destule atribute, dar să nu vă inchipuiți că sunt prea multe pe care să le înțelegeți (dacă aveți nivelul meu de know-how (how) în Oracle).

TABLE_NAME	TABLESPACE_NAME	CLUSTER_NAME	IOT_NAME	STATUS	PCT_FREE	PCT_USED	INITIALLY
1 RD1	USERS	(null)	(null)	VALID	10	(null)	
2 RD2	USERS	(null)	(null)	VALID	10	(null)	
3 CLIENTI	USERS	(null)	(null)	VALID	10	(null)	
4 PERSOANE	USERS	(null)	(null)	VALID	10	(null)	
5 JUDETE	USERS	(null)	(null)	VALID	10	(null)	
6 CODURI_POSTALE	USERS	(null)	(null)	VALID	10	(null)	
7 PERSCLIENTI	USERS	(null)	(null)	VALID	10	(null)	
8 PRODUSE	USERS	(null)	(null)	VALID	10	(null)	
9 FACTURI	USERS	(null)	(null)	VALID	10	(null)	
10 LINIIFACT	USERS	(null)	(null)	VALID	10	(null)	
11 INCASARI	USERS	(null)	(null)	VALID	10	(null)	
12 INCASFACT	USERS	(null)	(null)	VALID	10	(null)	

Figura 15.28. Câteva linii și coloane din tabela virtuală USER_TABLES

Mai mucalită este tabela virtuală USER_CATALOG din care putem afla numai numele tuturor tabelor, tabelor virtuale, sinonimelor și secvențelor din schema utilizatorului curent:

```
SELECT * FROM user_catalog
```

Despre atributele unei tabele aflăm câte ceva din tabela `USER_TAB_COLUMNS`. O parte dintre coloanele acesteia (și liniile dedicate atributelor tablei `FACTURI`) este prezentată în figura 15.29.

SELECT * FROM user_tab_columns WHERE table_name = 'FACTURI'

TABLE_NAME	COLUMN_NAME	DATA_TYPE	DATA_TYPE_MOD	DATA_TYPE_OWNER	DATA_LENGTH	DATA_PRECISION	DATA_SCALE	NUL
FACTURI	NRFAC	NUMBER	(null)	(null)	22	8	0	N
FACTURI	DATAFACT	DATE	(null)	(null)	7	(null)	(null)	N
FACTURI	CODCL	NUMBER	(null)	(null)	22	6	0	N
FACTURI	OBS	VARCHAR2	(null)	(null)	50	(null)	(null)	Y
FACTURI	VALTOTALA	NUMBER	(null)	(null)	22	10	2	Y
FACTURI	TVA	NUMBER	(null)	(null)	22	10	2	Y
FACTURI	VALINCASATA	NUMBER	(null)	(null)	22	10	2	Y
FACTURI	REDUCERI	NUMBER	(null)	(null)	22	9	2	Y
FACTURI	PENALIZARI	NUMBER	(null)	(null)	22	9	2	Y

Figura 15.29. Câteva coloane din view-ul `USER_TAB_COLUMNS` dedicate atributelor tablei `FACTURI`

Informațiile sintetice despre toate restricțiile definite în tabelele aflate în schema curentă sunt disponibile în tabela virtuală `USER_CONSTRAINTS` în care atributul `Constraint_Type` poate avea valorile P (pentru cheile primare), U (pentru cheile alternative), R (pentru restricțiile referențiale) și C (pentru restricțiile-utilizator la nivel de atribut și înregistrare). Astfel, pentru a afla toate restricțiile de tip CHECK definite în tabela `JUDETE` (figura 15.30), folosim interogarea:

SELECT constraint_name, search_condition
FROM user_constraints
WHERE table_name = 'JUDETE' AND constraint_type = 'C'

CONSTRAINT_NAME	SEARCH_CONDITION
NN_JUDET	"JUDET" IS NOT NULL
NN_REGIUNE	"REGIUNE" IS NOT NULL
CK_JUD	jud=LTRIM(UPPER(jud))
CK_JUDET	judet=LTRIM(INITCAP(judet))
CK_REGIUNE	regiune IN ('Banat', 'Transilvania', 'Dobrogea', 'Oltenia', 'Muntenia', 'Moldova')

Figura 15.30. Restricțiile de tip CHECK din tabela `JUDETE`

Alcătuirea cheilor primare și alternative este disponibilă recurând (și) la tabela virtuală `USER_CONS_COLUMNS`. Folosim o interogare similară celei din paragraful anterior (vezi figura 15.31):

SELECT uc.constraint_name, constraint_type, 'C' || ucc1.column_name ||
CASE WHEN ucc2.column_name IS NULL THEN " ELSE 'C' || ucc2.column_name
END ||
CASE WHEN ucc3.column_name IS NULL THEN " ELSE 'C' || ucc3.column_name
END ||
CASE WHEN ucc4.column_name IS NULL THEN " ELSE 'C' || ucc4.column_name
END || ')' AS expresie_restricție
FROM user_constraints uc

```

LEFT OUTER JOIN user_cons_columns ucc1
    ON uc.constraint_name=ucc1.constraint_name AND ucc1.position=1
LEFT OUTER JOIN user_cons_columns ucc2
    ON uc.constraint_name=ucc2.constraint_name AND ucc2.position=2
LEFT OUTER JOIN user_cons_columns ucc3
    ON uc.constraint_name=ucc3.constraint_name AND ucc3.position=3
LEFT OUTER JOIN user_cons_columns ucc4
    ON uc.constraint_name=ucc4.constraint_name AND ucc4.position=4
WHERE uc.table_name = 'LINIIFACT' AND uc.constraint_type IN ('P', 'U')

```

CONSTRAINT_NAME	CONSTRAINT_TYPE	EXPRESIE_RESTRICTIE
PK_LINIIFACT	P	(NRFACT,LINIE)
UN_LINIIFACT	U	(NRFACT,CODPR)

Figura 15.31. Compoziția cheilor primare și alternative ale tabelului LINIIFACT

Încheiem paragraful cu același exemplu din PostgreSQL, dedicat recompunerii restricțiilor referențiale. Oracle nu permite clauza ON UPDATE CASCADE, iar ON DELETE nu ne interesează, așa că efortul redactării comenzii SELECT care să afișeze informații despre cheile străine declarate în tabela LINIIFACT (figura 15.32) este ceva mai redus:

```

SELECT uc.constraint_name AS restrictie_copil, ucc1.table_name AS tabela_copil,
    ucc1.column_name AS atribut_copil,
    r_constraint_name AS restrictie_parinte, ucc2.table_name AS tabela_parinte,
    ucc2.column_name AS atribut_parinte
FROM user_constraints uc
    INNER JOIN user_cons_columns ucc1
        ON uc.constraint_name = ucc1.constraint_name
    INNER JOIN user_cons_columns ucc2
        ON uc.r_constraint_name = ucc2.constraint_name
        AND ucc1.position = ucc2.position
WHERE uc.table_name = 'LINIIFACT' AND uc.constraint_type = 'R'
ORDER BY uc.constraint_name, ucc1.position

```

RESTRICTIE_COPII	TABELA_COPII	ATRIBUT_COPII	RESTRICTIE_PARINTE	TABELA_PARINTE	ATRIBUT_PARINTE
FK_LINIIFACT_FACTURI	LINIIFACT	NRFACT	PK_FACTURI	FACTURI	NRFACT
FK_LINIIFACT_PRODUSE	LINIIFACT	CODPR	PK_PRODUSE	PRODUSE	CODPR

Figura 15.32. Afișarea datelor despre cheile străine ale tabelului LINIIFACT

15.2.3. Obiecte din schema unei baze de date în DB2

DB2 administrează două seturi de tabele virtuale în cadrul dicționarului de date, în schemele SYSCAT și SYSSTAT. Ambele seturi sunt accesibile utilizatorilor.

Tabelele virtuale din SYSSTAT reprezintă un subset al celor din SYSCAT, singurul avantaj fiind că o parte din atribute sunt actualizabile, deși acest lucru este foarte riscant. Cei de la IBM au mai introdus câteva view-uri în schema SYSIBM și pentru a se apropia de prevederile standardului. Date despre tabelele virtuale (figura 15.33) dicționar pot fi obținute simplu:

```
SELECT * FROM syscat.views
```

VIEWSCHEMA	VIEWNAME	OWNER	OWNERTYPE	SEQ...	VIEWCHECK	READONLY	VALID	QUA
SYSIBM	CHECK_CONSTRAINTS	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	COLUMNS	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	COLUMNS_S	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	DUAL	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	REFERENTIAL_CONSTRAINTS	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	REF_CONSTRAINTS	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	TABLE_CONSTRAINTS	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	TABLES	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	TABLES_S	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	USER_DEFINED_TYPES	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	UDT_S	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	VIEWS	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	PARAMETERS	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	PARAMETERS_S	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	ROUTINES	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	ROUTINES_S	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	SYSFUNCTIONS	SYSIBM	S	1	N	N	Y	SYSI
SYSIBM	SYSPROCEDURES	SYSIBM	S	1	N	N	Y	SYSI
SYSIBM	SYSFUNCPARMS	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	SYSPROCPARMS	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	SYSREVTYP mappings	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	SYSDDMMY1	SYSIBM	S	1	N	Y	Y	SYSI
SYSIBM	SYSROUTINEPROPERTIESJAVA	SYSIBM	S	1	N	Y	Y	SYSI
SYSCAT	ATTRIBUTES	SYSIBM	S	1	N	Y	Y	SYSI
SYSCAT	AUDITPOLICIES	SYSIBM	S	1	N	Y	Y	SYSI
SYSCAT	AUDITTRIG	SYSIBM	S	1	N	Y	Y	SYSI

Figura 15.33. Câteva informații despre (câteva) view-uri dicționar

Cele două scheme din care se afișează view-urile sunt SYSIBM și SYSCAT (figura ilustrează doar o parte din rezultat). Numele tabelelor sunt destul de sugestive: TABLES, COLUMNS, CHECK_CONSTRAINTS etc. Spre exemplu, ne interesează să aflăm numele tabelelor din schema curentă. Folosind view-ul TABLES din schema SYSCAT, interogarea este:

```
SELECT tablename FROM SYSCAT.tables  
WHERE tabschema=CURRENT_USER AND type='T'  
ORDER BY 1
```

iar folosind view-ul cu același nume dar din (sub)schema SYSIBM sintaxa este:

```
SELECT table_name FROM SYSIBM.tables  
WHERE table_schema = CURRENT_USER AND table_type = 'BASE TABLE'  
ORDER BY 1
```

Câteva informații despre atributele tabelii FACTURI pot fi aflate fie prin interogarea ce folosește *syscat.columns* (vezi figura 15.34):

```
SELECT colname, colno, typename, length, scale, default, nulls  
FROM SYSCAT.columns  
WHERE tabschema=CURRENT_USER AND tablename='FACTURI'
```

ORDER BY colno

COLNAME	COLNO	TYPENAME	LENGTH	SCALE	DEFAULT	NULLS
NRFACT	0	INTEGER	4	0		N
DATAFACT	1	DATE	4	0	CURRENT DATE	N
CODCL	2	SMALLINT	2	0		N
OBS	3	VARCHAR	50	0		Y
VALTOTALA	4	DECIMAL	10	20		Y
TVA	5	DECIMAL	10	20		Y
VALINCASATA	6	DECIMAL	10	20		Y
REDUCERI	7	DECIMAL	9	20		Y
PENALIZARI	8	DECIMAL	9	20		Y

Figura 15.34. Informații despre atributele tabeli FACTURI furnizate de *syscat.columns*

Alte detalii, dintre care doar o parte sunt cuprinse în figura 15.35, sunt obținute prin consultarea *sysibm.columns*:

TABL...	TA...	TABL...	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULL	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	CHAR...
SQL2008	MARIN	FACTURI	CODCL	3		NO	SMALLINT		
SQL2008	MARIN	FACTURI	DATAFACT	2	CURRENT DATE	NO	DATE		
SQL2008	MARIN	FACTURI	NRFACT	1		NO	INTEGER		
SQL2008	MARIN	FACTURI	OBS	4		YES	CHARACTER ...	50	
SQL2008	MARIN	FACTURI	PENALIZARI	9		YES	DECIMAL		
SQL2008	MARIN	FACTURI	REDUCERI	8		YES	DECIMAL		
SQL2008	MARIN	FACTURI	TVA	6		YES	DECIMAL		
SQL2008	MARIN	FACTURI	VALINCASATA	7		YES	DECIMAL		
SQL2008	MARIN	FACTURI	VALTOTALA	5		YES	DECIMAL		

Figura 15.35. Informații despre atributele tabeli FACTURI furnizate de *sysibm.columns*

Restricțiile definite în tabelele unei scheme constituie obiectul tabeli virtuale TABCONST din schema *syscat*. Iată cum le obținem pe cele declarate în tabele JUDETE, FACTURI, LINIIFACT, DOCTORI și GĂRZI (vezi figura 15.36):

CONSTNAME	TABNAME	TYPE
PK_DOCTORI	DOCTORI	P
FK_FACTURI_CLIENTI	FACTURI	F
CK_DATAFACT	FACTURI	K
PK_FACTURI	FACTURI	P
SQL081009123234650	GARZI	F
PK_GARZI	GARZI	P
CK_JUD	JUDETE	K
CK_REGIUNE	JUDETE	K
PK_JUDETE	JUDETE	P
NN_REGIUNE	JUDETE	U
FK_LINIIFACT_FACTURI	LINIIFACT	F
FK_LINIIFACT_PRODUSE	LINIIFACT	F
CK_LINIE	LINIIFACT	K
PK_LINIIFACT	LINIIFACT	P
UN_LINIIFACT	LINIIFACT	U

Figura 15.36. Restricții în câteva tabele

```
SELECT constname, tabname, type FROM SYSCAT.tabconst
WHERE tabschema=CURRENT_USER AND
      tabname IN ('JUDETE', 'FACTURI', 'LINIIFACT', 'DOCTORI', 'GARZI')
ORDER BY tabname, type, constname
```

Pentru că nu am folosit clauza CONSTRAINT la declararea restricției referențiale între tabela GARZI (copil) și DOCTORI (părinte), numele atribuit „din burtă” acestei restricții este destul de lung - SQL08010091...

Am crede că (aproximativ) același rezultat l-am obține dacă am recurge și la view-ul TABLE_CONSTRAINTS din schema SYSIBM, știind că numele atributelor sunt un pic schimbate:

```
SELECT constraint_name, table_name, constraint_type
FROM sysibm.table_constraints
WHERE table_schema=CURRENT_USER AND
      table_name IN ('JUDETE', 'FACTURI', 'LINIIFACT',
                    'DOCTORI', 'GARZI')
ORDER BY table_name, constraint_type, constraint_name
```

Figura 15.37 ne înșală așteptările, numărul restricțiilor furnizate fiind sensibil mai mare. Apar în plus o serie de CHECK-uri de care nu ne amintim să le fi declarat în scripturile de creare sau ulterior. Fiecare dintre cele patru tabele este „împroprietărită” cu câteva CHECK-uri, deci sistemul pare destul de democratic. Vom vedea îndată explicația.

CONSTRAINT_NAME	TABLE_NAME	CONSTRAINT_TYPE
PK_DOCTORI	DOCTORI	PRIMARY KEY
1009123234465001000005000200	DOCTORI	CHECK
1009123234465001010005000200	DOCTORI	CHECK
FK_FACTURI_CLIENTI	FACTURI	FOREIGN KEY
PK_FACTURI	FACTURI	PRIMARY KEY
110308195870300500000F000200	FACTURI	CHECK
110308195870300501000F000200	FACTURI	CHECK
110308195870300502000F000200	FACTURI	CHECK
CK_DATAFACT	FACTURI	CHECK
SQL0801009123234650	GARZI	FOREIGN KEY
PK_GARZI	GARZI	PRIMARY KEY
1009123234637001000006000200	GARZI	CHECK
1009123234637001010006000200	GARZI	CHECK
PK_JUDETE	JUDETE	PRIMARY KEY
NN_REGIUNE	JUDETE	UNIQUE
1103081957500001000009000200	JUDETE	CHECK
1103081957500001010009000200	JUDETE	CHECK
1103081957500001020009000200	JUDETE	CHECK
CK_JUD	JUDETE	CHECK
CK_REGIUNE	JUDETE	CHECK
FK_LINIIFACT_FACTURI	LINIIFACT	FOREIGN KEY
FK_LINIIFACT_PRODUSE	LINIIFACT	FOREIGN KEY
PK_LINIIFACT	LINIIFACT	PRIMARY KEY
UN_LINIIFACT	LINIIFACT	UNIQUE
1103081958812001000010000200	LINIIFACT	CHECK
1103081958812001010010000200	LINIIFACT	CHECK
1103081958812001020010000200	LINIIFACT	CHECK
1103081958812001030010000200	LINIIFACT	CHECK
1103081958812001040010000200	LINIIFACT	CHECK
CK_LINIE	LINIIFACT	CHECK

Figura 15.37. *sysibm.table_constraints* furnizează mai multe restricții decât *syscat.tabconst*

Restricțiile-utilizator la nivel de atribut și înregistrare pot fi aflate din tabela virtuală CHECKS din schema *syscat* – vezi figura 15.38. Au fost extrase numele restricției, numele tabelului și clauza CHECK efectivă:

```
SELECT constname, tabname, text
FROM SYSCAT.checks
WHERE tabschema=CURRENT_USER
```

CONSTNAME	TABNAME	TEXT
CK_ADRESA_C...	CLIENTI	SUBSTR(ADRESA,1,1) = UPPER(SUBSTR(ADRESA,1,1))
CK_CODCL	CLIENTI	CODCL > 1000
CK_CODFISCAL	CLIENTI	SUBSTR(CODFISCAL,1,1) = UPPER(SUBSTR(CODFISCAL,1,1))
CK_DENCL	CLIENTI	SUBSTR(DENCL,1,1) = UPPER(SUBSTR(DENCL,1,1))
NN_LOC	CODURI_POS...	CODPOST=LTRIM(CODPOST)
CK_DATAFACT	FACTURI	DATAFACT >= '2007-01-01' AND DATAFACT <= '2015-12-31'
CK_CODDOC	INCASARI	CODDOC=UPPER(LTRIM(CODDOC))
CK_DATADOC	INCASARI	DATADOC >= '2007-01-01' AND DATADOC <= '2010-12-31'
CK_DATAINC	INCASARI	DATAINC >= '2007-01-01' AND DATAINC <= '2015-12-31'
CK_DATE_INC...	INCASARI	DATAINC >= DATADOC
CK_JUD	JUDETE	JUD=LTRIM(UPPER(JUD))
CK_REGIUNE	JUDETE	REGIUNE IN ('Banat', 'Transilvania', 'Dobrogea', 'Oltenia', 'Muntenia', 'Moldova')
CK_LINIE	LINIIFACT	LINIE > 0
CK_FUNCIE	PERSCLIENTI	SUBSTR(FUNCIE,1,1) = UPPER(SUBSTR(FUNCIE,1,1))
CK_ADRESA_P...	PERSOANE	SUBSTR(ADRESA,1,1) = UPPER(SUBSTR(ADRESA,1,1))
CK_CNP	PERSOANE	CNP=LTRIM(UPPER(CNP))
CK_SEX	PERSOANE	SEX IN ('F','B')
CK_CODPR	PRODUSE	CODPR > 0
CK_DENPR	PRODUSE	SUBSTR(DENPR,1,1) = UPPER(SUBSTR(DENPR,1,1))
CK_PRODUSE...	PRODUSE	SUBSTR(GRUPA,1,1) = UPPER(SUBSTR(GRUPA,1,1))

Figura 15.38. Restricții de tip CHECK furnizare de *syscat.checks*

Numele fiecărui atribut implicat într-un CHECK este disponibil dintr-o altă tabelă virtuală a dicționarului – *syscat.COLCHECKS*.

Dacă dorim să comparăm rezultatul cu cel furnizat de view-ul *SYSIBM.CHECK_CONSTRAINTS*, ne așteptăm ca, după exemplul precedent, să apară câteva linii în plus. Supriză: este identic !

```
SELECT cc.constraint_name, table_name, check_clause
FROM sysibm.check_constraints cc INNER JOIN sysibm.table_constraints tc
ON cc.constraint_name=tc.constraint_name
WHERE table_schema=CURRENT_USER
ORDER BY table_name, constraint_name
```

Să recapitulăm ceea ce am observat în ultimele două exemple folosind view-uri din schema *SYSIBM*. *TABLE_CONSTRAINTS* furnizează niște restricții (cu nume dubioase) de tip CHECK care însă nu se găsesc în *CHECK_CONSTRAINTS*. Să le extragem pe cele din tabele *JUDETE*, *LINIIFACT*, *DOCTORI* și *GARZI* (figura 15.39):

```
SELECT constraint_name, table_name, constraint_type
FROM sysibm.table_constraints
WHERE table_schema=CURRENT_USER AND table_name IN
```

```

('JUDETE', 'FACTURI', 'LINIIFACT', 'DOCTORI', 'GARZI')
AND constraint_type = 'CHECK' AND constraint_name NOT IN
(SELECT constraint_name
FROM sysibm.check_constraints )
ORDER BY table_name, constraint_type, constraint_name

```

În figură, pentru tabela JUDETE sunt trei linii. Folosindu-ne de intuiția feminină putem concluziona că aceste restricții sunt, de fapt, NOT NULL-uri pe care DB2 le consideră de tip CHECK, cu de la sine putere.

CONSTRAINT_NAME	TABLE_NAME	CONSTRAINT_TYPE
1009123234465001000005000200	DOCTORI	CHECK
1009123234465001010005000200	DOCTORI	CHECK
110308195870300500000F000200	FACTURI	CHECK
110308195870300501000F000200	FACTURI	CHECK
110308195870300502000F000200	FACTURI	CHECK
1009123234637001000006000200	GARZI	CHECK
1009123234637001010006000200	GARZI	CHECK
1103081957500001000009000200	JUDETE	CHECK
1103081957500001010009000200	JUDETE	CHECK
1103081957500001020009000200	JUDETE	CHECK
1103081958812001000010000200	LINIIFACT	CHECK
1103081958812001010010000200	LINIIFACT	CHECK
1103081958812001020010000200	LINIIFACT	CHECK
1103081958812001030010000200	LINIIFACT	CHECK
1103081958812001040010000200	LINIIFACT	CHECK

Figura 15.39. Restricții de tip CHECK „mincinoase”

Analog paragrafelor anterioare, ne propunem să aflăm compoziția cheilor primare, alternative și străine din tabela LINIIFACT. Iată una dintre soluțiile DB2:

```

SELECT tc.constname, '(' || kcu1.colname ||
CASE WHEN kcu2.colname IS NULL THEN " ELSE ',' || kcu2.colname END
|| CASE WHEN kcu3.colname IS NULL THEN " ELSE ',' || kcu3.colname END
|| CASE WHEN kcu4.colname IS NULL THEN " ELSE ',' || kcu4.colname END
|| ')' AS expresie_restricție
FROM SYSCAT.tabconst tc
INNER JOIN SYSCAT.keycoluse kcu1 ON tc.constname=kcu1.constname
AND kcu1.colseq = 1
LEFT OUTER JOIN SYSCAT.keycoluse kcu2 ON tc.constname=kcu2.constname
AND kcu2.colseq = 2
LEFT OUTER JOIN SYSCAT.keycoluse kcu3 ON tc.constname=kcu3.constname
AND kcu3.colseq = 3
LEFT OUTER JOIN SYSCAT.keycoluse kcu4 ON tc.constname=kcu4.constname
AND kcu4.colseq = 4
WHERE tc.tabschema=CURRENT_USER AND tc.tabname='LINIIFACT'

```

Cât despre restricțiile referențiale, varianta pe care o prezentăm pentru afișarea clauzelor FOREIGN KEY din tabela LINIIFACT (figura 15.40) este neverosimil de

simplă, bazându-se pe o singură tabelă virtuală – SYSCAT.REFERENCES (și pe faptul că cheile străine sunt simple):

```
SELECT ' CONSTRAINT ' || constname || ' FOREIGN KEY ( ' || RTRIM(fk_colnames)
      || ') REFERENCES ' || reftabname || ' ( ' || RTRIM(fk_colnames) || ' )'
FROM SYSCAT.references
WHERE tabschema=CURRENT_USER AND tabname='LINIIFACT'
```

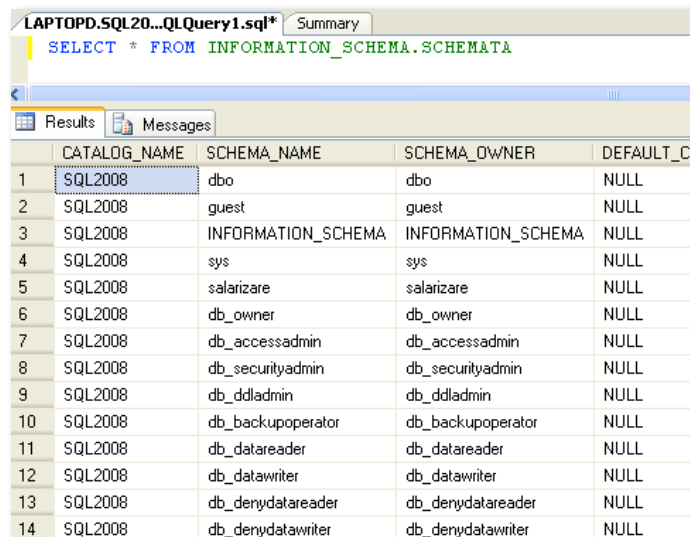


```
EXPRESIE FOREIGN KEY
CONSTRAINT FK_LINIIFACT_FACTURI FOREIGN KEY ( NRFACT) REFERENCES FACTURI ( NRFACT)
CONSTRAINT FK_LINIIFACT_PRODUSE FOREIGN KEY ( CODPR) REFERENCES PRODUSE ( CODPR)
```

Figura 15.40. Clauzele FOREIGN KEY pentru tabela LINIIFACT

15.2.4. Obiecte din schema unei baze de date în MS SQL Server

SQL Server are un triplu mecanism de furnizare a informațiilor despre schema BD. Primul, moștenit de pe „vremea Sybase”⁵, este bazat pe proceduri stocate speciale și nu ne interează deloc (cel puțin pe mine). Al doilea este un mecanism de tabele și tabele virtuale propriu creat în schema SYS, iar al treilea se dorește o aliniere la prevederile standardului, fiind implementat prin tabele virtuale plasate în schema INFORMATION_SCHEMA.



	CATALOG_NAME	SCHEMA_NAME	SCHEMA_OWNER	DEFAULT_CHARACTER_SET_NAME
1	SQL2008	dbo	dbo	NULL
2	SQL2008	guest	guest	NULL
3	SQL2008	INFORMATION_SCHEMA	INFORMATION_SCHEMA	NULL
4	SQL2008	sys	sys	NULL
5	SQL2008	salarizare	salarizare	NULL
6	SQL2008	db_owner	db_owner	NULL
7	SQL2008	db_accessadmin	db_accessadmin	NULL
8	SQL2008	db_securityadmin	db_securityadmin	NULL
9	SQL2008	db_ddladmin	db_ddladmin	NULL
10	SQL2008	db_backupoperator	db_backupoperator	NULL
11	SQL2008	db_datareader	db_datareader	NULL
12	SQL2008	db_datawriter	db_datawriter	NULL
13	SQL2008	db_denydatareader	db_denydatareader	NULL
14	SQL2008	db_denydatawriter	db_denydatawriter	NULL

Figura 15.41. Subscheme din BD curentă (în SQL Server)

Astfel, pentru a vizualiza toate (subs)schemele existente în baza de date, interogarea se scrie astfel:

⁵ [Kriegel & Trukhnov 2008], p.511

```
SELECT * FROM INFORMATION_SCHEMA.SCHEMATA
```

Din figura 15.41, recunoaștem doar trei scheme, *dbo* – cea în care am creat tabelele de până acum, *salarizare* – cea creată în acest capitol și nou discutata *INFORMATION_SCHEMA*.

Un exemplu de view-dicționar non-standard este *objects* care conține informații sintetice despre toate obiectele din subschemele la care utilizatorul curent are drepturi de acces (figura 15.42):

```
SELECT * FROM sys.objects
```

	name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date
41	sysbinsubobjs	97	NULL	4	0	S	SYSTEM_TABLE	2005-10-14 01:36:22.110	2005-10-14
42	personal2	289436105	NULL	1	0	U	USER_TABLE	2008-10-20 11:59:30.153	2008-10-20
43	pk_personal2	305436162	NULL	1	289436105	PK	PRIMARY_KEY_CONSTRAINT	2008-10-20 11:59:30.153	2008-10-20
44	fk_personal2	321436219	NULL	1	289436105	F	FOREIGN_KEY_CONSTRAINT	2008-10-20 11:59:30.153	2008-10-20
45	sporuri	337436276	NULL	1	0	U	USER_TABLE	2008-10-20 11:59:30.153	2008-10-20
46	pk_sporuri	353436333	NULL	1	337436276	PK	PRIMARY_KEY_CONSTRAINT	2008-10-20 11:59:30.153	2008-10-20
47	fk_sporuri_pers2	369436390	NULL	1	337436276	F	FOREIGN_KEY_CONSTRAINT	2008-10-20 11:59:30.153	2008-10-20
48	rd1	417436561	NULL	1	0	U	USER_TABLE	2008-10-22 08:08:49.293	2008-10-22
49	rd2	433436618	NULL	1	0	U	USER_TABLE	2008-10-22 08:08:49.310	2008-10-22
50	judete	449436675	NULL	1	0	U	USER_TABLE	2008-10-23 15:22:36.013	2008-10-23
51	pk_judete	465436732	NULL	1	449436675	PK	PRIMARY_KEY_CONSTRAINT	2008-10-23 15:22:36.013	2008-10-23
52	un_judet	481436789	NULL	1	449436675	UQ	UNIQUE_CONSTRAINT	2008-10-23 15:22:36.013	2008-10-23
53	ck_jud	497436846	NULL	1	449436675	C	CHECK_CONSTRAINT	2008-10-23 15:22:36.013	2008-10-23
54	DF__judete__R...	513436903	NULL	1	449436675	D	DEFAULT_CONSTRAINT	2008-10-23 15:22:36.013	2008-10-23
55	ck_regiune	529436960	NULL	1	449436675	C	CHECK_CONSTRAINT	2008-10-23 15:22:36.013	2008-10-23
56	coduri_postale	545437017	NULL	1	0	U	USER_TABLE	2008-10-23 15:22:36.013	2008-10-23

Figura 15.42. O mică parte din conținutul tabelii virtuale SYS.OBJECTS

Ca și în DB2, putem formula interogări atât asupra view-urilor din schema SYS, cât și a celor din schema INFORMATION_SCHEMA. De exemplu, putem afla numele tabelilor create de noi în schema *dbo* fie cu interogarea:

```
SELECT name
FROM sys.tables
WHERE type_desc = 'USER_TABLE' AND schema_id IN
      (SELECT schema_id FROM sys.schemas WHERE name = 'dbo')
ORDER BY 1
```

fie prin:

```
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'dbo' AND TABLE_TYPE = 'BASE TABLE'
ORDER BY 1
```

Informațiile despre atribute sunt plasate în tabelele virtuale *sys.columns* și *INFORMATION_SCHEMA.COLUMNS*. Astfel, aflăm date despre atributele tabelii FACTURI (vezi figura 15.43) fie prin varianta:

```
SELECT *
```

```

FROM sys.columns
WHERE object_id IN (SELECT object_id FROM sys.tables
                    WHERE name='facturi')

ORDER BY column_id

```

fie prin:

```

SELECT *
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'facturi'
ORDER BY ORDINAL_POSITION

```

The screenshot shows two queries executed in SQL Server Enterprise Manager. The first query uses `sys.columns` and the second uses `INFORMATION_SCHEMA.COLUMNS`. Both result in the same table structure for the 'facturi' table.

	object_id	name	column_id	system_type_id	user_type_id	max_length	precision	scale	collation_name	is_nullable	is_ansi_padded	is_rowid
1	1041438784	NrFact	1	56	56	4	10	0	NULL	0	0	0
2	1041438784	DataFact	2	58	58	4	16	0	NULL	0	0	0
3	1041438784	CodCl	3	52	52	2	5	0	NULL	0	0	0
4	1041438784	Obs	4	167	167	50	0	0	SQL_Romanian_CP1250_CS_AS	1	1	0
5	1041438784	ValTotala	5	108	108	9	10	2	NULL	1	0	0
6	1041438784	TVA	6	108	108	9	10	2	NULL	1	0	0
7	1041438784	ValIncasata	7	108	108	9	10	2	NULL	1	0	0
8	1041438784	Reduceri	8	108	108	5	9	2	NULL	1	0	0
9	1041438784	Penalizari	9	108	108	5	9	2	NULL	1	0	0

	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULLABLE	DATA_TYPE	CHARACTER_MAXIMUM
1	SQL2008	dbo	facturi	NrFact	1	NULL	NO	int	NULL
2	SQL2008	dbo	facturi	DataFact	2	(getdate())	NO	smalldatetime	NULL
3	SQL2008	dbo	facturi	CodCl	3	[(dbo].[f_v_codcl])]	NO	smallint	NULL
4	SQL2008	dbo	facturi	Obs	4	NULL	YES	varchar	50
5	SQL2008	dbo	facturi	ValTotala	5	[(0)]	YES	numeric	NULL
6	SQL2008	dbo	facturi	TVA	6	[(0)]	YES	numeric	NULL
7	SQL2008	dbo	facturi	ValIncasata	7	[(0)]	YES	numeric	NULL
8	SQL2008	dbo	facturi	Reduceri	8	[(0)]	YES	numeric	NULL
9	SQL2008	dbo	facturi	Penalizari	9	[(0)]	YES	numeric	NULL

Figura 15.43. Două variante de vizualizare a informațiilor despre atributele tablei FACTURI

Valorile implicite ale atributelor pot fi vizualizate dintr-un view special – `sys.default_constraints` – sau tot din `INFORMATION_SCHEMA.COLUMNS`.

Aflarea restricțiilor definite în tablele JUDETE, FACTURI, LINIIFACT și GARZI (vezi figura 15.44) presupune folosirea unei interogări aproape identice celei din DB2⁶:

```

SELECT constraint_name, table_name, constraint_type
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE table_schema=CURRENT_USER AND
      table_name IN ('judete', 'facturi', 'liniifact', 'doctori', 'garzi')
ORDER BY table_name, constraint_type, constraint_name

```

⁶ Numele celor cinci table se scriu cu minuscule (așa le-am creat în cap.3)

constraint_name	table_name	constraint_type
pk_doctori	doctori	PRIMARY KEY
ck_datafact	facturi	CHECK
ck_facturi_codcl	facturi	CHECK
fk_facturi_clienti	facturi	FOREIGN KEY
pk_facturi	facturi	PRIMARY KEY
FK_garzi_iddoctor_581D0306	garzi	FOREIGN KEY
pk_garzi	garzi	PRIMARY KEY
ck_jud	judete	CHECK
ck_regiune	judete	CHECK
pk_judete	judete	PRIMARY KEY
un_judet	judete	UNIQUE
ck_linie	liniifact	CHECK
fk_linifact_facturi	liniifact	FOREIGN KEY
fk_linifact_produce	liniifact	FOREIGN KEY
pk_linifact	liniifact	PRIMARY KEY

Figura 15.44. Restricții definite în cinci tabele

Expresiile clauzelor CHECK ale celor patru tabele se obțin prin varianta:

```
SELECT table_name, tc.constraint_name, check_clause
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS tc
     INNER JOIN INFORMATION_SCHEMA.CHECK_CONSTRAINTS cc
           ON tc.constraint_name = cc.constraint_name
WHERE table_schema=CURRENT_USER AND
      table_name IN ('judete', 'facturi', 'liniifact',
                    'doctori', 'garzi') AND constraint_type = 'CHECK'
ORDER BY table_name, tc.constraint_name
```

sau:

```
SELECT o2.name AS table_name, cc.name AS constraint_name,
      cc.definition AS check_clause
FROM sys.check_constraints cc
     INNER JOIN sys.objects o1 ON cc.name = o1.name
     INNER JOIN sys.objects o2 ON cc.parent_object_id = o2.object_id
     AND o2.name IN ('judete', 'facturi', 'liniifact', 'doctori', 'garzi')
```

Pentru un pic de variațiune, vom extrage expresiile cheilor primare și alternative ale tuturor tabelor din schema curentă. Varianta de mai jos, al cărei rezultat este cuprins în figura 15.45 folosește tabelele virtuale dicționar: SYS.TABLES, SYS.KEY_CONSTRAINTS, SYS.INDEX_COLUMNS și SYS.COLUMNS:

```
SELECT t.name AS table_name, k.name AS constraint_name,
      k.type_desc AS constraint_type,
      CASE WHEN x1.name IS NULL THEN " ELSE '(' + x1.name END +
      CASE WHEN x2.name IS NULL THEN " ELSE ',' + x2.name END +
```

```

CASE WHEN x3.name IS NULL THEN " ELSE ',' + x3.name END +
CASE WHEN x4.name IS NULL THEN " ELSE ',' + x4.name END +
CASE WHEN x1.name IS NOT NULL OR x2.name IS NOT NULL OR
      x3.name IS NOT NULL OR x4.name IS NOT NULL
      THEN ')' ELSE ',' END AS expresie_restrictie
FROM sys.tables t
INNER JOIN sys.key_constraints k ON t.object_id = k.parent_object_id
LEFT OUTER JOIN
  (SELECT i.object_id, index_id, c.name, key_ordinal
   FROM sys.index_columns i
        INNER JOIN sys.columns c ON i.object_id = c.object_id
        AND i.column_id = c.column_id
   ) x1 ON x1.object_id = t.object_id AND x1.index_id = k.unique_index_id
      AND x1.key_ordinal = 1
LEFT OUTER JOIN
  (SELECT i.object_id, index_id, c.name, key_ordinal
   FROM sys.index_columns i
        INNER JOIN sys.columns c ON i.object_id = c.object_id AND
        i.column_id = c.column_id
   ) x2 ON x2.object_id = t.object_id AND x2.index_id = k.unique_index_id
      AND x2.key_ordinal = 2
LEFT OUTER JOIN
  (SELECT i.object_id, index_id, c.name, key_ordinal
   FROM sys.index_columns i
        INNER JOIN sys.columns c ON i.object_id = c.object_id AND
        i.column_id = c.column_id
   ) x3 ON x3.object_id = t.object_id AND x3.index_id = k.unique_index_id
      AND x3.key_ordinal = 3
LEFT OUTER JOIN
  (SELECT i.object_id, index_id, c.name, key_ordinal
   FROM sys.index_columns i
        INNER JOIN sys.columns c ON i.object_id = c.object_id AND
        i.column_id = c.column_id
   ) x4 ON x4.object_id = t.object_id AND x4.index_id = k.unique_index_id
      AND x4.key_ordinal = 4
ORDER BY 1

```

table_name	constraint_name	constraint_type	expresie_restricție
candidati	PK_candidati__7251D655	PRIMARY_KEY_CONSTRAINT	(IdCandidat)
clienti	pk_clienti	PRIMARY_KEY_CONSTRAINT	(CodCl)
clienti	un_codfiscal	UNIQUE_CONSTRAINT	(CodFiscal)
coduri_postale	pk_coduri_post	PRIMARY_KEY_CONSTRAINT	(CodPost)
distante	PK_distante__55EA41D1	PRIMARY_KEY_CONSTRAINT	(Loc1,Loc2)
doctori	pk_doctori	PRIMARY_KEY_CONSTRAINT	(Iddoctor)
facturi	pk_facturi	PRIMARY_KEY_CONSTRAINT	(NrFact)
garzi	pk_garzi	PRIMARY_KEY_CONSTRAINT	(iddoctor,inceput_garda)
incasari	pk_incasari	PRIMARY_KEY_CONSTRAINT	(CodInc)
incasfact	pk_incasfact	PRIMARY_KEY_CONSTRAINT	(CodInc,NrFact)
judete	pk_judete	PRIMARY_KEY_CONSTRAINT	(Jud)
judete	un_judet	UNIQUE_CONSTRAINT	(Judet)
liniifact	pk_linifact	PRIMARY_KEY_CONSTRAINT	(NrFact,Linie)
mastere	PK_mastere__70698DE3	PRIMARY_KEY_CONSTRAINT	(Spec)
pacienti	pk_pacienti	PRIMARY_KEY_CONSTRAINT	(Idpacient)
perscienti	pk_perscienti	PRIMARY_KEY_CONSTRAINT	(CNP,CodCl,Funcție)
persoane	pk_persoane	PRIMARY_KEY_CONSTRAINT	(CNP)
personal2	pk_personal2	PRIMARY_KEY_CONSTRAINT	(Marca)
produse	pk_produse	PRIMARY_KEY_CONSTRAINT	(CodPr)
sporuri	pk_sporuri	PRIMARY_KEY_CONSTRAINT	(An,Luna,Marca)
triaj	pk_internari	PRIMARY_KEY_CONSTRAINT	(idexaminare)

Figura 15.45. Chei primare și alternative

Același raport se obține mai simplu prin utilizarea doar a tabele virtuale din INFORMATION_SCHEMA, TABLE_CONSTRAINTS și KEY_COLUMN_USAGE:

```
SELECT t.table_name, t.constraint_name, t.constraint_type,
CASE WHEN k1.column_name IS NULL THEN " ELSE '(' + k1.column_name END +
CASE WHEN k2.column_name IS NULL THEN " ELSE ',' + k2.column_name END +
CASE WHEN k3.column_name IS NULL THEN " ELSE ',' + k3.column_name END +
CASE WHEN k4.column_name IS NULL THEN " ELSE ',' + k4.column_name END +
CASE WHEN k1.column_name IS NOT NULL OR k2.column_name IS NOT NULL OR
      k3.column_name IS NOT NULL OR k4.column_name IS NOT NULL
THEN ')' ELSE ',' END AS expresie_restricție
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS t
LEFT OUTER JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE k1
ON t.table_name = k1.table_name AND
   t.constraint_name = k1.constraint_name
   AND k1.ordinal_position = 1
LEFT OUTER JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE k2
ON t.table_name = k2.table_name AND
   t.constraint_name = k2.constraint_name
   AND k2.ordinal_position = 2
LEFT OUTER JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE k3
```

```
ON t.table_name = k3.table_name AND
    t.constraint_name = k3.constraint_name
    AND k3.ordinal_position = 3
LEFT OUTER JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE k4
    ON t.table_name = k3.table_name AND
        t.constraint_name = k4.constraint_name
        AND k4.ordinal_position = 4
WHERE t.constraint_type IN ('PRIMARY KEY', 'UNIQUE')
ORDER BY 1
```

Rămâne de încercat pe cont propriu redactarea interogării prin care se reconstruiesc expresiile cheilor străine, folosindu-se, pe lângă tabelele virtuale dicționar de până acum, „clasicele” *sys.foreign_keys* și *sys.foreign_key_columns*, sau „standardizata” *REFERENTIAL_CONSTRAINTS* din *INFORMATION_SCHEMA*.