EL LLENGUTAGE DDL (DATA DEFINITION LANGUAGE)

El Llenguatge de Definició de Dades és un subllenguatge SQL que descriu la part de SQL que crea, altera i suprimeix objectes de la base de dades. Aquests objectes de base de dades inclouen esquemes, taules, vistes, seqüències, índex, variables, sinònims... Recordem que un esquema és una agrupació lògica d'objectes SQL.

Objectes de la base de dades

Objecte	Descripció
Taula	Unitat bàsica d'emmagatzematge; formada per files i columnes.
Vista	Representa lògicament subjocs de dades d'una o més taules.
Seqüència	Generador de valor numèric.
Índex	Millora el rendimient d'algunes consultes.
Sinònim	Dona noms alternatius als objectes.

GESTIÓ DE TAULES EN LA BASE DE DADES

NOMS DELS OBJECTES DE LA BASE DE DADES

Noms de taules i columnes:

- Han de començar per una lletra.
- Només han de contenir A Z, a z, 0 9, _, $, i #.
- No es pot duplicar el nom d'un altre objecte que sigui propietat del mateix usuari.
- No ha de ser una paraula reservada del Servidor Postgresql.

LA SENTÈNCIA CREATE TABLE

Per executar aquesta sentència l'usuari ha de tenir:

- El privilegi *CREATE* per a l'esquema on es vol crear la taula.
- Una àrea d'emmagatzematge.

La sentència especifica:

- Nom de taula
- Per a les columnes: nom, tipus de dada i mida. Addicionalment, pot especificar certes restriccions (aquest tema es tractarà més endavant).

```
CREATE TABLE [IF NOT EXISTS] table_name (
   column1 datatype(length) column_contraint,
   column2 datatype(length) column_contraint,
   column3 datatype(length) column_contraint,
   table_constraints
);
```

> Exemple:

```
CREATE TABLE accounts (
        user_id serial PRIMARY KEY,
        username VARCHAR (50) UNIQUE NOT NULL,
        password VARCHAR 50 NOT NULL,
        email VARCHAR 255 UNIQUE NOT NULL,
        created_on TIMESTAMP NOT NULL,
        last_login TIMESTAMP);
```

Podem realitzar una consulta sobre el diccionari de dades per a verificar que la taula s'ha creat correctament:

```
SELECT table_name, column_name, data_type
FROM information_schema.columns
WHERE table_name ='accounts';
```

ALGUNS TIPUS DE DADES

Tipo de Dato	Descripción
VARCHAR(size)	Dada caràcter de longitud variable
CHAR(size)	Dada caràcter de longitud fixe.
TEXT	Dada caràcter de longitud ilimitada.
NUMERIC(p,s)	Dada numèrica de longitud variable.
SMALLINT / INT	Enters de 2 i 4 bytes
SERIAL	Enter auto-incremental
DATE	Data
TIME	Temps
TIMESTAMP	Data amb temps

TAULES DE LA BASE DE DADES

Taules d'usuari:

- Recopilació de taules creades i mantingudes per l'usuari.
- Contenen informació d'usuari.

Diccionari de dades:

- Recopilació de taules creades i mantingudes pel servidor de PostgreSQL.
- Contenen informació de la base de dades.

> Exemple, consulta en el diccionari de dades de la informació de les taules propietat d'un usuari.

```
SELECT *
FROM pg_catalog.pg_tables
WHERE schemaname != 'pg_catalog' AND
      schemaname != 'information_schema';
```

COLUMNES AUTO-INCREMENTALS

- Utilitzarem el tipus de dada **SERIAL** per definir columnes auto-incrementals. El tipus de dada serà enter.
- Es generarà un objecte **SEQUENCE** associat a la columna auto-incremental.
- Hi ha 3 tipus de dada SERIAL:
  - SMALLSERIAL (2 bytes d'1 a 32767)
  - SERIAL (4 bytes d'1 a 2.147.483.647)
  - BIGSERIAL (8 bytes d'1 a 9.223.372.036.854.775.807)
- Les columnes tipus SERIAL generen un valor automàtic, però podem assignar-li un valor explícitament.

> Veure: Postgresql Sequences

- A partir de la versió 10 s'introdueix la clàusula **GENERATED AS IDENTITY** que permet generar valors auto-incrementals sobre qualsevol columna de tipus SMALLINT, INT o BIGINT. (Aquesta opció s'explicarà amb més detall en el tema 9 Llenguatge DML quan veiem les sentències d'inserció de files en una taula).

> Veure: Postgresql Identity Column

> Exemple:

```
CREATE TABLE fruits(
   id SERIAL PRIMARY KEY,
   name VARCHAR NOT NULL);
```

> Per inserir un element indicarem que li doni el valor per defecte perquè generi l'autoincremental.

```
INSERT INTO fruits(id,name)
VALUES(DEFAULT,'Apple');
```

LA SENTÈNCIA ALTER TABLE

Utilitzarem la sentència **ALTER TABLE** per:

- Afegir una nova columna.
- Modificar una columna existent.
- Definir un valor per defecte a una columna.
- Esborrar una columna.
- Canviar el nom d'una columna.

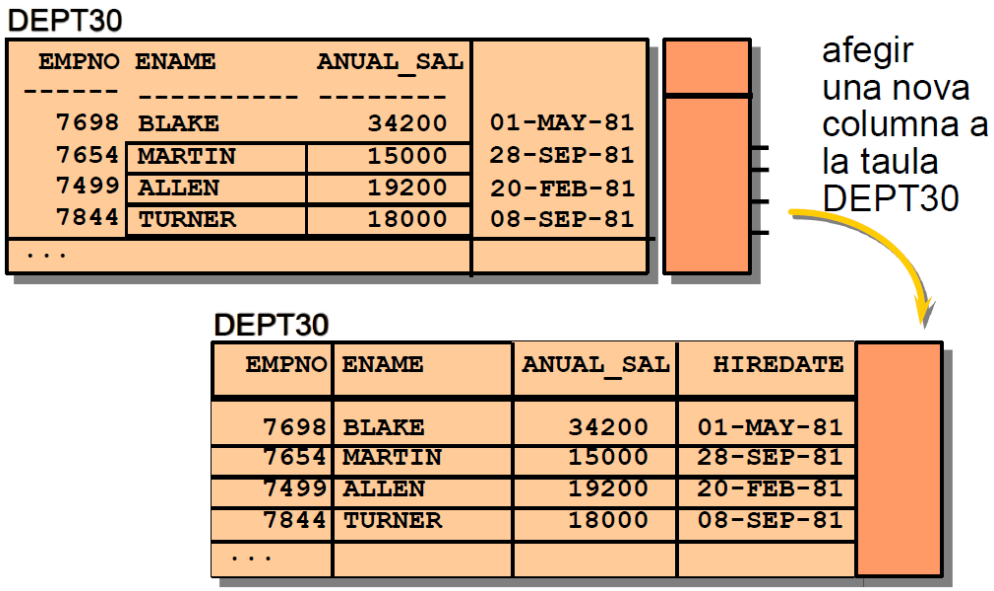
```
ALTER TABLE table
ADD COLUMN   (column datatype [DEFAULT expr]
             [, column datatype]...);
```

```
ALTER TABLE table
ALTER COLUMN column [TYPE datatype |SET DEFAULT expr
                  | SET|DROP NOT NULL];
```

```
ALTER TABLE table
RENAME COLUMN column
TO new_column;
```

```
ALTER TABLE table
DROP COLUMN [IF EXISTS] (column) [RESTRICT|CASCADE]
```

Addició d'una columna:



> Exemple:

```sql
ALTER TABLE dept30
ADD COLUMN job VARCHAR(9);
```

La nova columna passa a ser l'última.

Podem canviar el tipus de dada, mida i valor per defecte d'una columna. Si canvia el valor per defecte, afecta només posteriors insercions a la taula.

> Exemple:

```
ALTER TABLE dept30
ALTER COLUMN ename TYPE VARCHAR(15);
```

Podem eliminar columnes.

> Exemple:

```
ALTER TABLE dept80
DROP COLUMN job_id;
```

ELIMINACIÓ D'UNA TAULA

S'esborren totes les dades i l'estructura de la taula. Tots els índexs de la taula s'esborren.

```
DROP TABLE [IF EXISTS] table_name
[CASCADE | RESTRICT];
```

Si existeixen altres objectes relacionats amb la taula (vistes, triggers, funcions...) mitjançant les clàusules **CASCADE** i **RESTRICT** inidicarem com s'ha de comportar la sentència, amb CASCADE forcem l'eliminació de la taula i de totes les seves dependències, en canvi, amb l'opció per defecte RESTRICT no s'eliminarà la taula si existeixen dependències.

> Exemple:

```
DROP TABLE IF EXISTS author;
```

La clàusula IF EXISTS força al sistema a verificar si la taula existeix, en cas afirmatiu executa la sentència, en cas negatiu no fa res. Sense aquesta clàusula si la taula no existeix el sistema ens tornaria un error.

CANVIAR EL NOM D'UNA TAULA

Per canviar el nom d'una taula, vista, seqüència o sinònim, executarem la instrucció ALTER [objecte] ... RENAME. Per poder canviar el nom d'un objecte s'ha de ser el propietari de l'objecte.

```
ALTER TABLE table_name
RENAME TO new_table_name;
```

ADDICIÓ DE COMENTARIS EN UNA TAULA

Podem afegir comentaris a una taula o columna amb la comanda COMMENT ON.

> Exemples:

```
COMMENT ON TABLE emp
IS 'Employee Information';

COMMENT ON COLUMN emp.id
IS 'Column id Information';
```

RESTRICCIONS (CONSTRAINTS)

Les **restriccions** o *constraints* forcen regles en l'àmbit de taula. Les *constraints* prevenen l'esborrament d'una taula si hi ha dependències. Són vàlids els tipus de restriccions següents:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

Podem assignar un nom a la restricció, en cas contrari el servidor en generarà un usant el format *table-name_pkey*. Es pot crear una restricció en el moment de **crear la taula**, o bé, **després que la taula ja ha estat creada**. Es pot definir una restricció, en **l'àmbit de columna** o en **l'àmbit de taula**. Podrem consultar les *constraints* mitjançant el diccionari de dades.

```
CREATE TABLE [schema.]table
             (column datatype [DEFAULT expr]
          [column_constraint],
          …
          [table_constraint]);
```

> Exemple:

```
CREATE TABLE emp(
       empno NUMERIC(4),
       empname VARCHAR(10),
       deptno NUMERIC(7,2) NOT NULL,
CONSTRAINT emp_empno_pk PRIMARY KEY (empno));
```

Les *constraint* es poden definir en l'àmbit de columna:

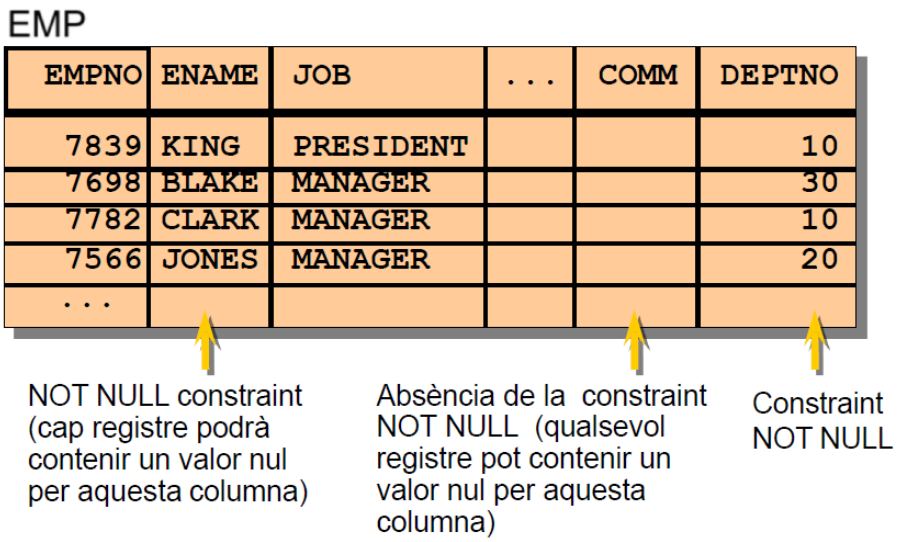
```
column [CONSTRAINT constraint_name] constraint_type,
```

O bé, es poden definir en l'àmbit de taula:

```
column,...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

CONSTRAINT NOT NULL

Assegura que els valors NULL no seran permesos per a la columna. La constraint NOT NULL es defineix sempre en l'àmbit de columna.

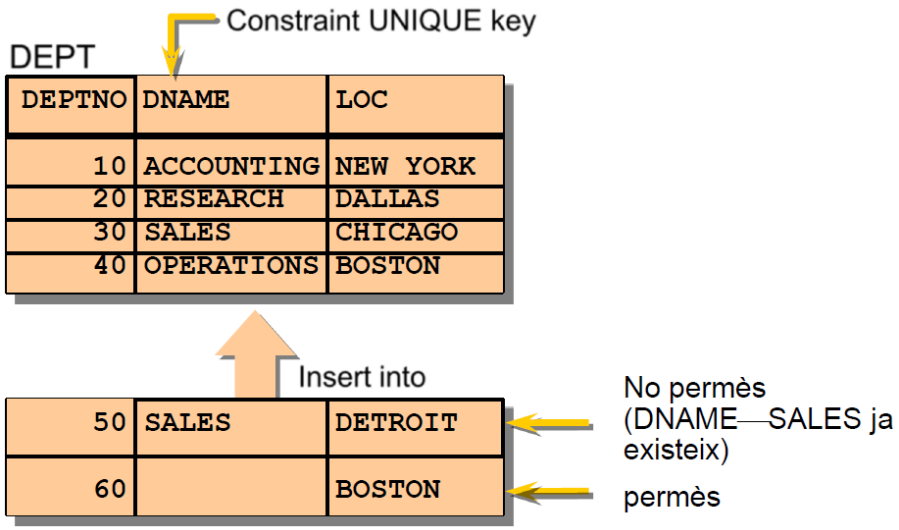


> Exemple

```sql
CREATE TABLE emp(
    empno NUMERIC(4),
    ename VARCHAR(10) NOT NULL,
    job VARCHAR(9),
    mgr NUMERIC(4),
    hiredate DATE,
    sal NUMERIC(7,2),
    comm NUMERIC(7,2),
    deptno NUMERIC(7,2) NOT NULL);
```

CONSTRAINT UNIQUE KEY

Assegura que els valors d'una columna, o la combinació de valors de diverses columnes, no es puguin repetir, és a dir, que siguin únics.



Aquesta *constraint* es pot definir, bé en l'àmbit de taula o bé, en l'àmbit de columna:

> Exemples:

```
CREATE TABLE dept(
  deptno NUMERIC(2),
  dname VARCHAR (14),
  loc VARCHAR (13),
  CONSTRAINT dept_dname_uk UNIQUE(dname));
```

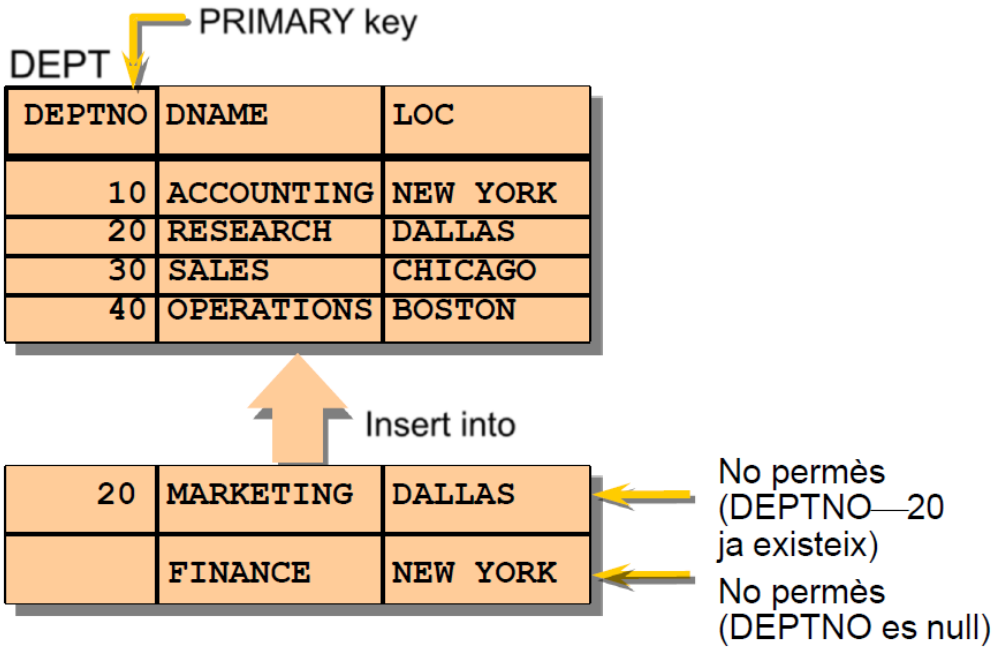
```
CREATE TABLE dept(
  deptno NUMERIC(2),
  dname VARCHAR(14) UNIQUE,
  loc VARCHAR(13));
```

Podem definir múltiples columnes, en aquest cas el que no es pot repetir és qualsevol combinació de valors de les múltiples columnes.

```
CREATE TABLE dept(
  deptno NUMERIC(2),
  dname  VARCHAR(14),
  loc    VARCHAR(13),
  CONSTRAINT dept_dname_uk UNIQUE(dname,loc));
```

CONSTRAINT PRIMARY KEY

Estableix quina columna o conjunt de columnes seran la clau primària de la taula. Recordem que els valors de la clau primària han de ser únics i no pot tenir valors NULL.



Aquesta *constraint* es pot definir, bé en l'àmbit de taula o bé, en l'àmbit de columna:

> Exemples:

```
CREATE TABLE dept(
 deptno NUMERIC(2),
 dname  VARCHAR(14),
 loc    VARCHAR(13),
 CONSTRAINT dept_deptno_pk PRIMARY KEY(deptno));
```

```
CREATE TABLE dept(
 deptno NUMERIC(2) CONSTRAINT dept_deptno_pk PRIMARY KEY,
 dname  VARCHAR(14),
 loc    VARCHAR(13));
```

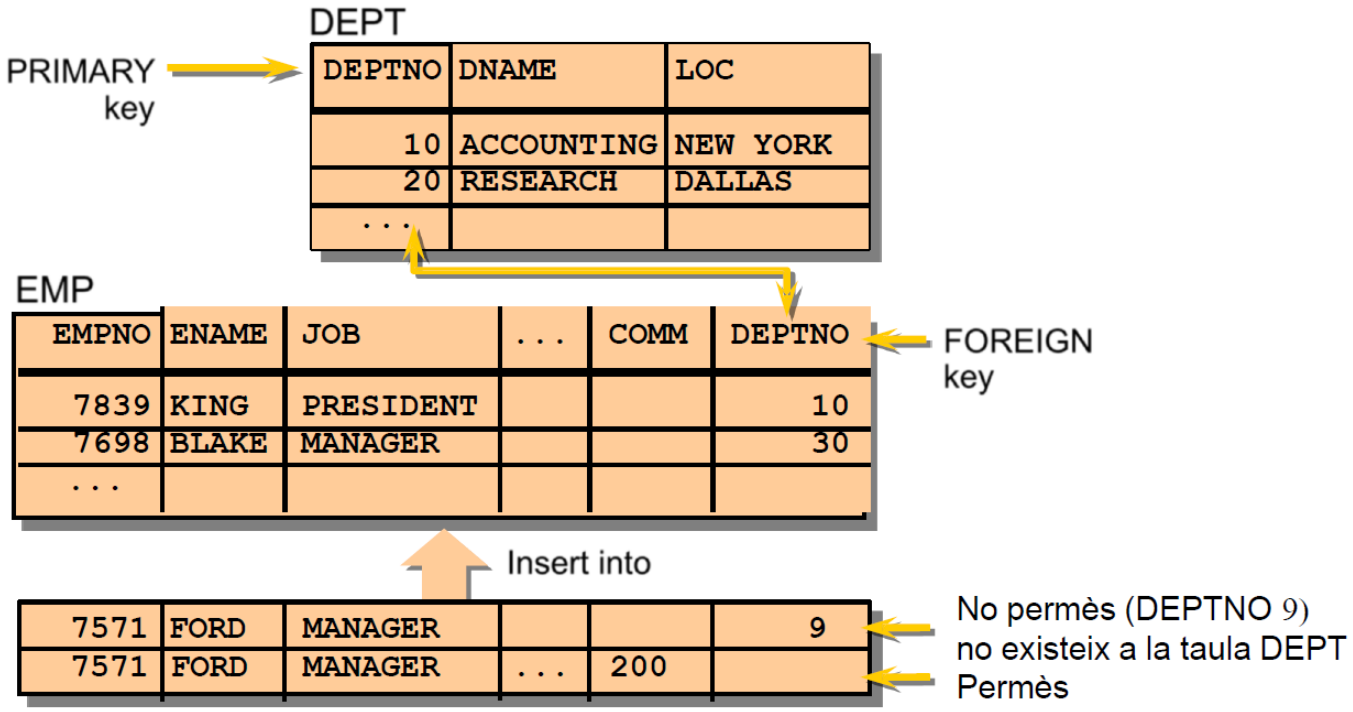
Quan tenim una clau primària múltiple la única opció és definir-la en l'àmbit de taula.

> Exemple:

```
CREATE TABLE pelicula_actor(
 IdPeli   NUMERIC(4),
 IdActor  NUMERIC(5),
 fecha    DATE,
 PRIMARY KEY(IdPeli, IdActor));
```

CONSTRAINT FOREIGN KEY

Estableix quina columna o conjunt de columnes seran una clau forana. Recordem que els valors de la clau forana han d'existir en la columna que referencien o bé ser NULL.



Aquesta *constraint* es pot definir, bé en l'àmbit de taula o bé, en l'àmbit de columna:

> Exemples:

```
CREATE TABLE emp(
  empno   NUMERIC(4) PRIMARY KEY,
  ename   VARCHAR(10) NOT NULL,
  deptno  NUMERIC(7,2) NOT NULL,
  sal NUMERIC(7,2),
  CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno)
      REFERENCES dept (deptno));
```

```
CREATE TABLE emp(
  empno NUMERIC(4) PRIMARY KEY,
  ename VARCHAR(10) NOT NULL,
  deptno NUMERIC(7,2) NOT NULL CONSTRAINT emp_deptno_fk REFERENCES dept(deptno),
  sal NUMERIC(7,2),
  comm NUMERIC(7,2));
```

Paraules reservades:

- FOREIGN KEY: Defineix la columna a la taula filla a nivell de constraint de taula.
- REFERENCES : Identifica la taula i columna a la taula.
- ON DELETE CASCADE : Permet esborrar a la taula pare i esborrar registres dependents de la taula filla.
- ON DELETE SET NULL|SET DEFAULT : Converteix els valors de clau aliena dependents en valors nuls, en el seu valor per defecte.

- ON DELETE RESTRICT|NO ACTION : No permet esborrar a la taula pare si existeixen valors dependents de la taula filla.

> Una altre exemple:

```
CREATE TABLE emp(
  empno   NUMERIC(4) PRIMARY KEY,
  ename   VARCHAR(10) NOT NULL,
  job VARCHAR(9),
  mgr NUMERIC(4),
  hiredate DATE,
  sal NUMERIC(7,2),
  comm    NUMERIC(7,2),
  deptno  NUMERIC(7,2) NOT NULL,
  CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno)
  REFERENCES dept (deptno)
      ON DELETE CASCADE);
```

CONSTRAINT CHECK

Defineix una condició que ha de satisfer cada fila. Aquesta *constraint* es pot definir, bé en l'àmbit de taula o bé, en l'àmbit de columna:

> Exemple:

```
CREATE TABLE dept(
   deptno NUMERIC(2) PRIMARY KEY ,
   dname VARCHAR(14),
   loc VARCHAR(13),
   CONSTRAINT check_deptno_ok
   CHECK (deptno BETWEEN 10 AND 99),
   CONSTRAINT check_dname_ok
   CHECK (dname = UPPER(dname)));
```

```
CREATE TABLE emp (
  empno    NUMERIC(4) PRIMARY KEY,
  empname  VARCHAR(20),
  es_jefe  CHAR(1) CHECK (es_jefe IN('S','N')));
```

La sintaxi és igual que la que utilitzem en una clàusula WHERE d'una consulta:

> Veure: Clàusula Where

CONSTRAINT DEFAULT

Aquesta clàusula defineix un valor per defecte quan no s'informa el valor de la columna de forma explícita. Aquesta clàusula sempre es defineix a nivell de columna.

> Exemple:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric DEFAULT 9.99);
```

AFEGIR/ELIMNAR I ACTIVAR/DESACTIVAR CONSTRAINTS

Mitjançant la comanda ALTER TABLE podem afegir *constraints* a la nostra taula, eliminar-les o bé, activar-les i/o desactivar-les.

```
ALTER TABLE table
ADD [CONSTRAINT constraint] type (column);
```

```
ALTER TABLE table
DROP|ENABLE|DISABLE [CONSTRAINT constraint];
```

> Exemple:

```
ALTER TABLE emp
ADD CONSTRAINT emp_mgr_fk
FOREIGN KEY (mgr) REFERENCES emp(empno);
```

> Exemple, eliminar una *constraint*:

```
ALTER TABLE emp
DROP CONSTRAINT emp_mgr_fk;
```

Normalment haurem de saber el nom associat a la *constraint*, ja sigui un valor conegut o el que li ha assignat el sistema per defecte, ho podem saber consultant el diccionari de dades.

```
SELECT *
    FROM pg_catalog.pg_constraint;
```

> Algunes constraint, com la clau primària es poden eliminar sense necessitat de referenciar-les pel seu nom.

```
ALTER TABLE dept
DROP PRIMARY KEY;
```

Finalment, ens pot ser útil en algun moment determinat desactivar alguna constraint (per exemple una check constraint), i en qualsevol moment tornar-la a habilitar o activar.

> Exemple:

```
ALTER TABLE emp
DISABLE CONSTRAINT emp_empno_p k;
```

```
ALTER TABLE emp
ENABLE CONSTRAINT emp_empno_p k;
```