TEMA 6:

BASES DE DADES

OBJECTE-RELACIONALS

1. Introducció

2. Definició d'una classe
    - 2.1. Creació d'una classe
    - 2.2. Modificació d'una classe
    - 2.3. Esborrat d'una classe

3. Definició d'altres elements a partir d'una classe
    - 3.1. Definir el tipus d'un atribut d'una altra classe
    - 3.2. Definir el tipus d'un camp d'una taula
    - 3.3. Definir una taula

4. Declaració d'objectes
    - 4.1. Declaració d'un objecte a partir d'una classe
    - 4.2. Declaració d'un objecte a partir de la definició d'una taula

5. Ús d'objectes
    - 5.1. Accés als camps de l'objecte
    - 5.2. Assignació de valor a l'objecte
    - 5.3. Inserció en una taula amb algun camp compost (objecte)

6. Herència de taules
    - 6.1. Creació d'una taula filla
    - 6.2. Canvis en els registres de la taula filla
    - 6.3. Canvis en els registres de la taula mare

1. Introducció

Les bases de dades objecte-relacionals (BDOR) han evolucionat des del model relacional per a incorporar la tecnologia orientada a objectes (OO): fan ús de classes, objectes i herència.

Avantatges de les BDODR

- L'usuari pot crear els seus tipus de dades (classes) per a agrupar camps relacionats. Així aconseguim que les dades s'acosten més a la vida real.
- Reutilització d'objectes: podem definir objectes a partir d'altres.
- Més fàcil la integració dels programes OO (com Java) a la BD.

MySQL és el SGBD vist fins ara però no suporta la tecnologia OO. Per tant, en este tema vorem PostgreSQL: un SGBD lliure amb suport per a objectes capaç de competir amb qualsevol SGBD comercial.

Vorem com crear tipus de dades (classes) en PostgreSQL i utilitzar-les en la definició de taules. També sabrem com es fa el tractament de les dades en estes taules i, per últim, vorem la tecnologia de l'herència en les BD. Tot, mitjançant exemples per a que s'entenga més fàcilment.

2. Definició d'una classe

2.1. Creació d'una classe

```
CREATE TYPE Domicili AS (
        carrer varchar(30),
        numero int,
        cp int
);
```

La sintaxi és com la de la creació de taula però sense restriccions (de claus, de nul·litat, etc).

2.2. Modificació d'una classe

```
ALTER TYPE Domicili DROP ATTRIBUTE numero;
ALTER TYPE Domicili ADD ATTRIBUTE num INT;
ALTER TYPE Domicili ALTER ATTRIBUTE num TYPE CHAR(5);
```

Vorem que es poden definir altres elements a partir d'una classe base ja creada. En eixe cas, si intentem modificar eixa classe base ens donarà error. Si volem que els canvis de la classe base afecten també als altres elements definits a partir d'ella, afegirem la paraula CASCADE al final de les ordres anteriors.

```
ALTER TYPE Domicili  ...  CASCADE;
```

2.3. Esborrat d'una classe

```
DROP TYPE Domicili;
```

Si hi ha altres elements definits a partir d'eixe tipus, caldrà fer:

```
DROP TYPE Domicili CASCADE;
```

Vorem els efectes del CASCADE en cada cas.

3. Definició d'altres elements a partir d'una classe

Una volta creada la classe, la podem usar per a definir nous elements:

3.1. Definir el tipus d'un atribut d'una altra classe

```
CREATE TYPE Persona AS (
        dni varchar (9),
        nom varchar (15),
        domicili Domicili
);
```

Nota: PostgreSQL NO és *case sensitive* (tant en paraules reservades com en identificadors) però posarem les classes en majúscula per similitud amb els programes OO.

Nota: DROP TYPE Domicili CASCADE eliminaria el camp domicili de la classe Persona.

3.2. Definir el tipus d'un camp d'una taula

```
CREATE TABLE professors (
        ...
        domicili Domicili);
```

Nota: DROP TYPE Domicili CASCADE eliminaria el camp domicili de la taula professors.

3.3. Definir una taula

```
CREATE TABLE persones OF Persona (
        PRIMARY KEY (dni),
);
```

Estem dient que la taula té els mateixos camps que té la classe. I, a més, li diem les restriccions de la taula (restricció de clau, etc).

Nota: una taula no pot dir-se igual que un tipus

Nota: DROP TYPE Persona CASCADE eliminaria la taula persones.

4. Declaració d'objectes

Es fa en la secció DECLARE d'una funció. De dos formes possibles: a partir d'una classe o a partir de la definició d'una taula.

4.1. Declaració d'un objecte a partir d'una classe

```
DECLARE
        nom  nom_tipus_objecte;
```

Exemple:

```
DECLARE
        elMeuDomicili  Domicili;
```

4.2. Declaració d'un objecte a partir de la definició d'una taula

```
DECLARE
         nom  nom_taula%ROWTYPE;
```

Exemples:

```
DECLARE
        persona  persones%ROWTYPE;
```

Nota: igual que podem declarar un objecte del tipus d'una taula, també podem declarar una variable del tipus d'una columna:

```
DECLARE
        nomVar  nomTaula.nomCamp%TYPE;
```

Exemple:

```
DECLARE
        codiArt  articles.codi%TYPE;
```

5. Ús d'objectes

5.1. Accés als camps de l'objecte

a) En les variables:

objecte.camp -- Igual que en Java

Exemple:

```
DECLARE pers1 Persona;
pers1.domicili.carrer := "Sequial";
```

b) En les columnes de les taules. És a dir, en sentències Select i en condicions de Update i Delete:

(objecte**)**.camp -- Parèntesis obligatoris

Exemple:

```
DECLARE pers1 Persona;
SELECT (domicili).carrer
        INTO pers1.domicili.carrer
        FROM persones
        WHERE (domicili).cp = 46410...;
```

Si no posem els (), MySQL pensa que *domicili* és una taula i donarà error.

Si també cal posar el nom de la taula, el posem dins dels parèntesis:

```
SELECT (persones.domicili).carrer
        INTO pers1.domicili.carrer
        FROM persones, ...
        WHERE (persones.domicili).cp = 46410...;
```

c) En els valors de retorn d'una funció.

Si en la SELECT es fa una crida a una funció que retorna un objecte i volem accedir a un dels seus camps, també usarem ():

```
SELECT (nomFuncio(...)).camp FROM...
```

5.2. Assignació de valor a l'objecte

Suposant que tenim l'objecte alumne següent:

```
DECLARE
        alumne   alumnes%ROWTYPE;
```

Podem assignar el resultat d'una fila de la taula a eixe objecte:

```
SELECT *
        INTO alumne
        FROM alumnes
        WHERE...;
```

5.3. Inserció en una taula amb algun camp compost (objecte):

```
INSERT INTO persones VALUES (   '555555555',
                                'Pep',
                                ROW('Sequial', 13, 46410)
                            );
```

O bé, sense el ROW (només amb els parèntesis):

```
INSERT INTO persones VALUES (   '555555555',
                                'Pep',
                                ('Sequial', 13, 46410)
                            );
```

Nota: les constants van entre cometes simples (no dobles, com també admet MySQL).

6. Herència de taules

6.1. Creació d'una taula filla

a) Taula mare

Creem una taula "normal":

```
CREATE TABLE persones (
        id SERIAL PRIMARY KEY,
        nom VARCHAR (30),
        adreça VARCHAR (30)
);
```

Nota: el tipus de dades "SERIAL" és un enter autonumèric. Això vol dir que si s'insereix un registre en la taula sense indicar el valor d'eixe camp, li s'assigna un número automàticament que encara no estiga en la taula. Però per a usar l'herència no cal que hi haja un camp SERIAL; ni tan sols que hi haja una clau primària (encara que és aconsellable).

b) Taula filla

Creem una filla de l'anterior amb la paraula INHERITS:

```
CREATE TABLE estudiants (
        carrera varchar (50),
        grup char,
        grau int
) INHERITS (personal);
```

Hem creat la taula *estudiants* amb els mateixos camps que la taula *persones* més els camps: *carrera*, *grup* i *grau*.

Una taula pot tindre moltes classes filles (i nétes, besnétes...)

6.2. Canvis en els registres de la taula filla

Si inserim/esborrem/modifiquem un registre en la taula filla, TAMBÉ es reflecteixen els canvis en el registre corresponent de la taula mare:

```
INSERT INTO estudiants (nom, adreça, carrera, grup, grau)
        VALUES ('Anna Guillen' , 'Tarragona 19', 'Psicologia', 'C', 2);
```

Hem inserit un registre en la taula estudiants i un altre en personal, amb els mateixos valors en id, nom i adreça. De forma anàloga passa en els esborrats i modificacions.

6.3. Canvis en els registres de la taula mare

- Si inserim un registre en la taula mare, NO afecta per a res a la taula filla:

```
INSERT INTO personal (nom, adreça)
        VALUES ('Lluís Arnau', 'Barcelona 3');
```

Hem inserit un registre en la taula *personal* però no en la taula *estudiant*.

- Si modifiquem/esborrem registres en la taula mare, TAMBÉ es reflecteixen els canvis en els registres corresponents de la taula filla:

```
UPDATE personal SET ... WHERE ...
DELETE FROM personal WHERE ...
```

Hem modificat registres en la taula *personal* i els corresponents en la taula *estudiants* (els que complien la condició).

Si només volem modificar/esborrar registres de la taula mare que NO estiguen en cap taula filla:

```
UPDATE ONLY personal SET nom='Sr. ' || nom;
```

Només es modificaria Lluís (que no és estudiant), però no Anna (que sí que és estudiant).

```
DELETE FROM ONLY personal WHERE ...
```