Le modèle Objet-Relationnel

© Olivier Caron

Objectifs

- √ Les **performances** du modèle relationnel et la **richesse** de l'approche objet.

Objectifs

- √ Les **performances** du modèle relationnel et la **richesse** de l'approche objet.
- √ Apport de l'objet : identité, héritage, . . .

Objectifs

- √ Les **performances** du modèle relationnel et la **richesse** de l'approche objet.
- √ Apport de l'objet : identité, héritage, . . .
- √ Améliorer les types de données (ex : base de données multimédia)

Objectifs

- √ Les **performances** du modèle relationnel et la **richesse** de l'approche objet.
- √ Apport de l'objet : identité, héritage, . . .
- √ Améliorer les types de données (ex : base de données multimédia)
- √ Vers un langage plus riche : SQL 3

Objectifs

- √ Les **performances** du modèle relationnel et la **richesse** de l'approche objet.
- √ Apport de l'objet : identité, héritage, . . .
- √ Améliorer les types de données (ex : base de données multimédia)
- √ Vers un langage plus riche : SQL 3
- √ Remet en cause plusieurs fondements de l'approche relationnelle (ex : forme normale)

Plan

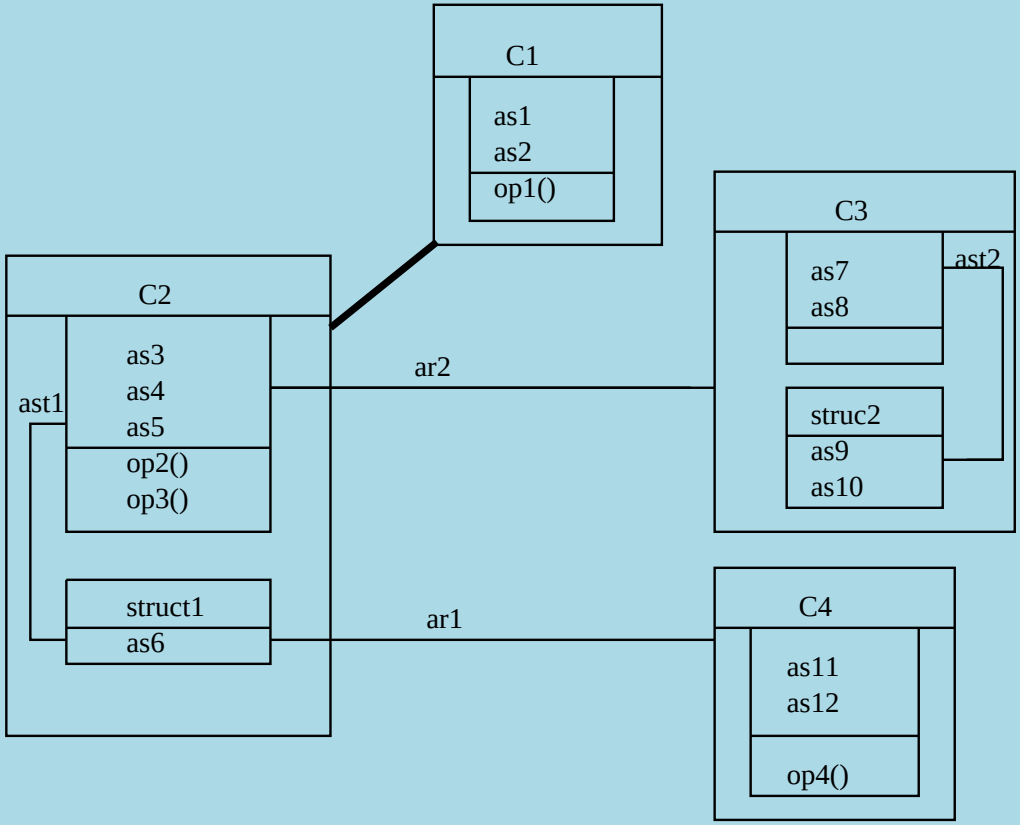
- √ Le modèle objet-relationnel
- √ Aspects objets Oracle 8
- √ Postgres en tant que SGBDOR

Conception d'un schema Objet(-relationnel)

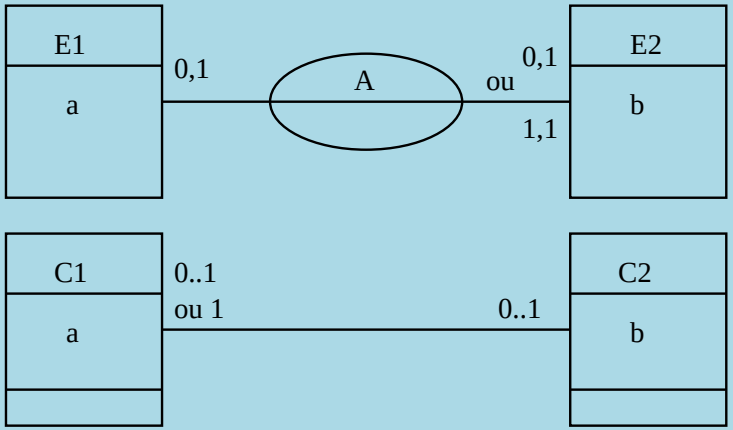
- √ Etape 1 : schéma conceptuel => formalisme de type EA (Merise), diagramme de classes (UML)
- √ Etape 2 : schéma "navigationnel"
- √ Etape 3 : traduction pour votre SGBD (ici, Oracle 8)

Vous savez faire l'étape 1 => on passe à l'étape 2.

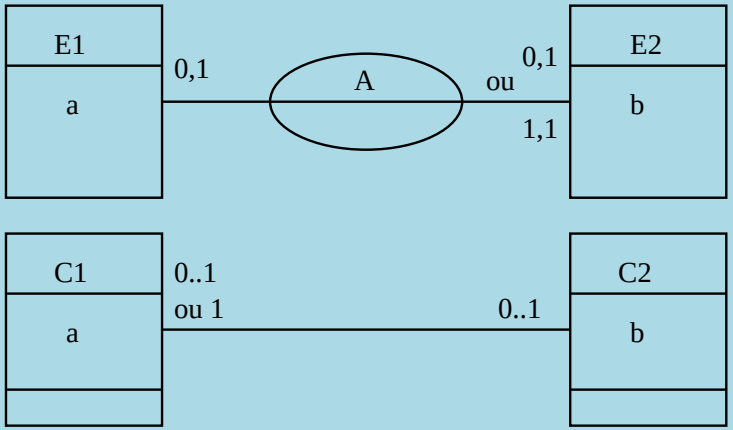
Schéma navigationnel

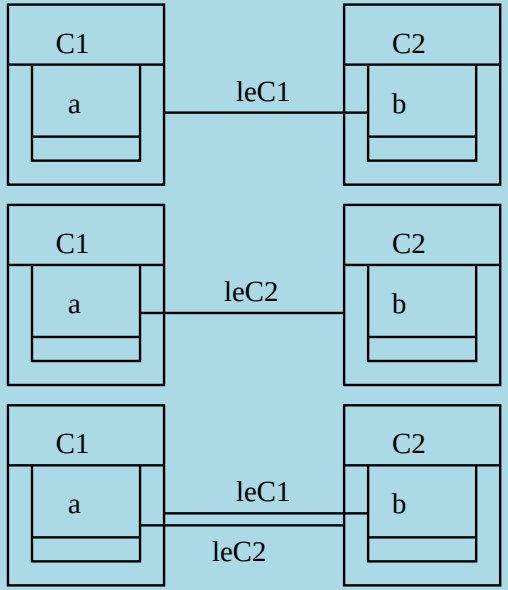


Associations 1-1

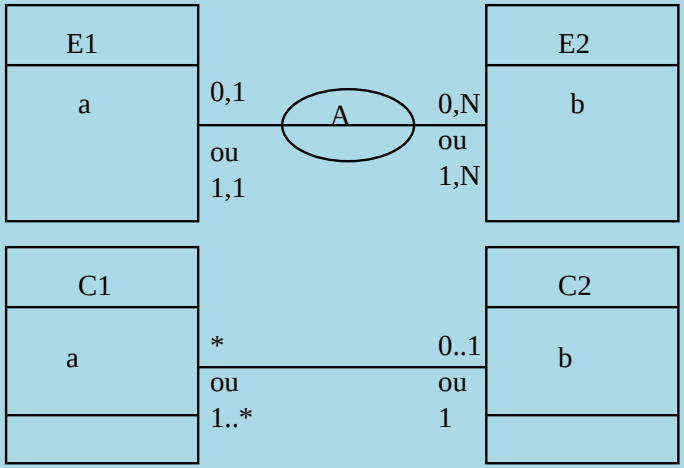


Associations 1-1

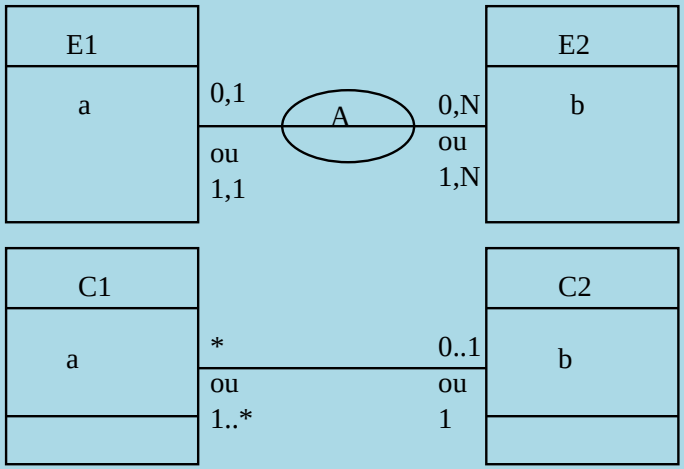


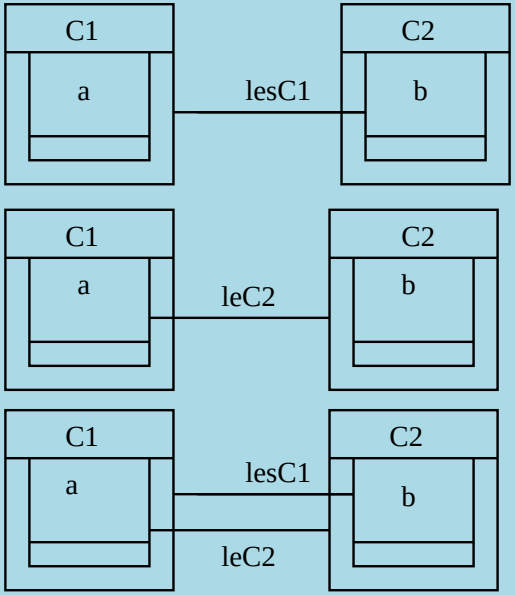


Associations 1-*

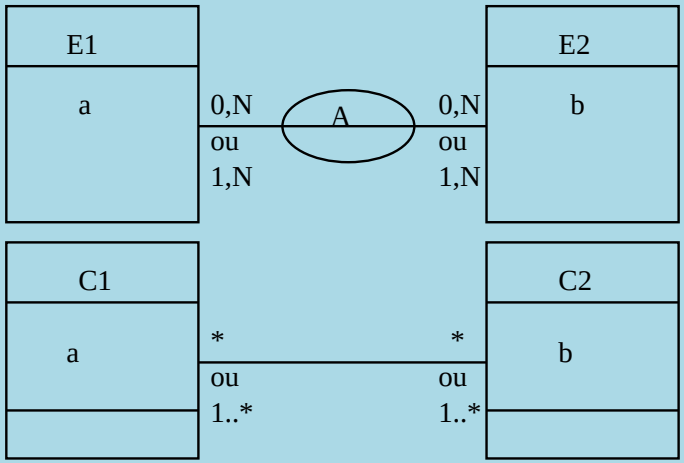


Associations 1-*

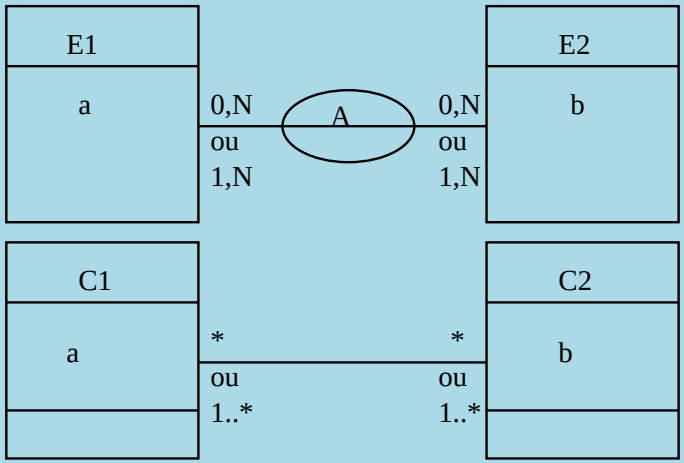


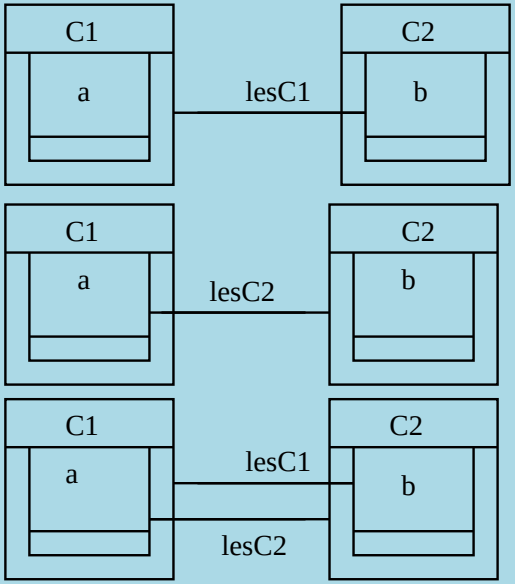


Associations *-* sans propriété

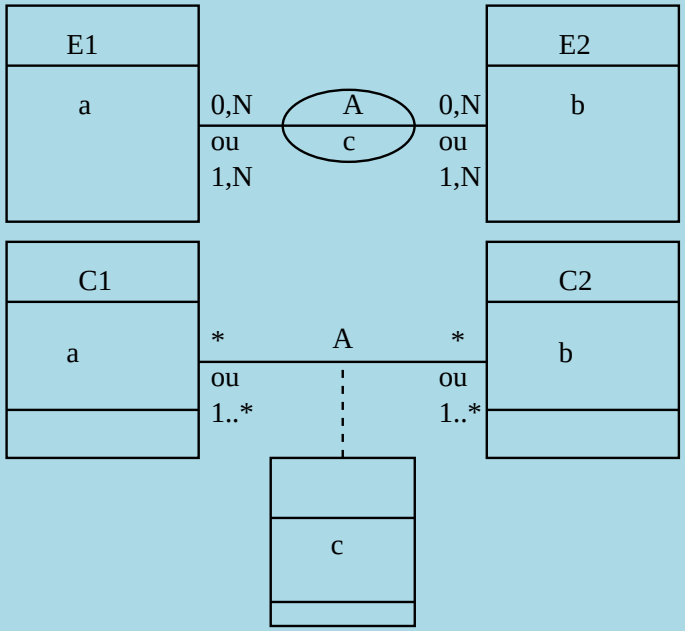


Associations *-* sans propriété

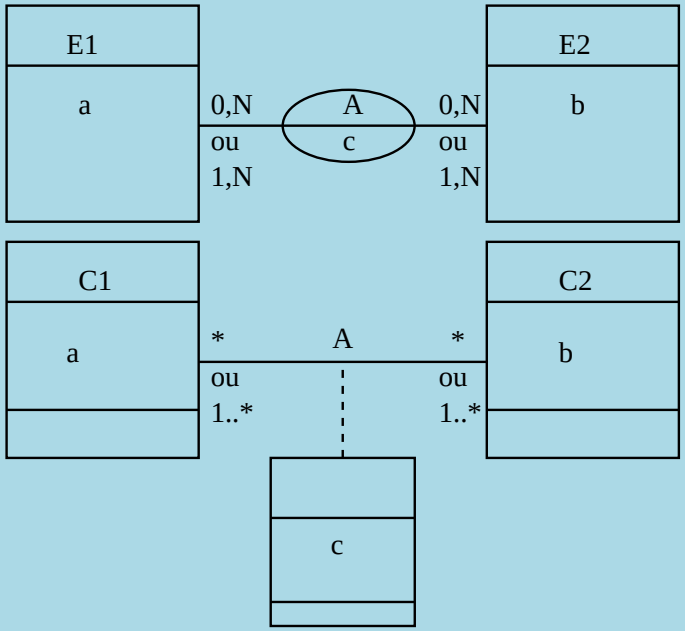


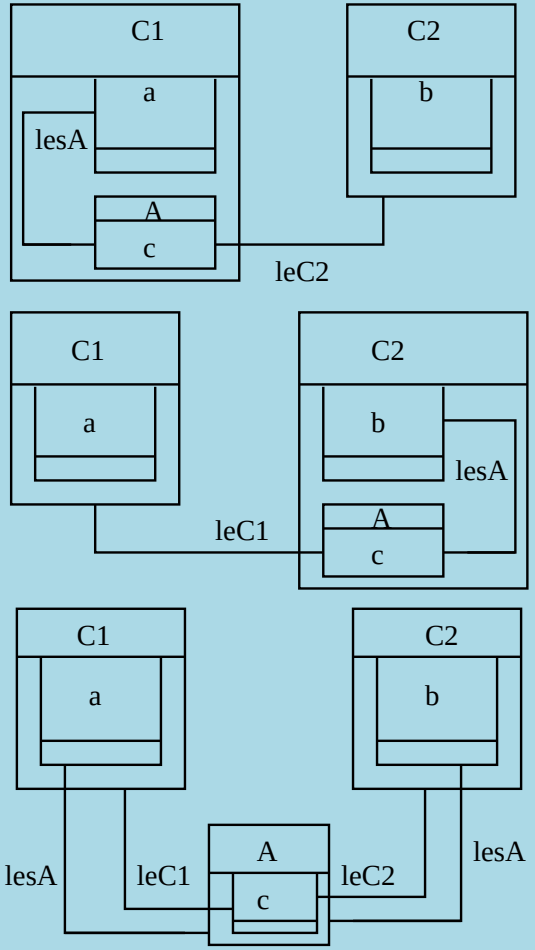


Associations *-* avec propriété



Associations *-* avec propriété



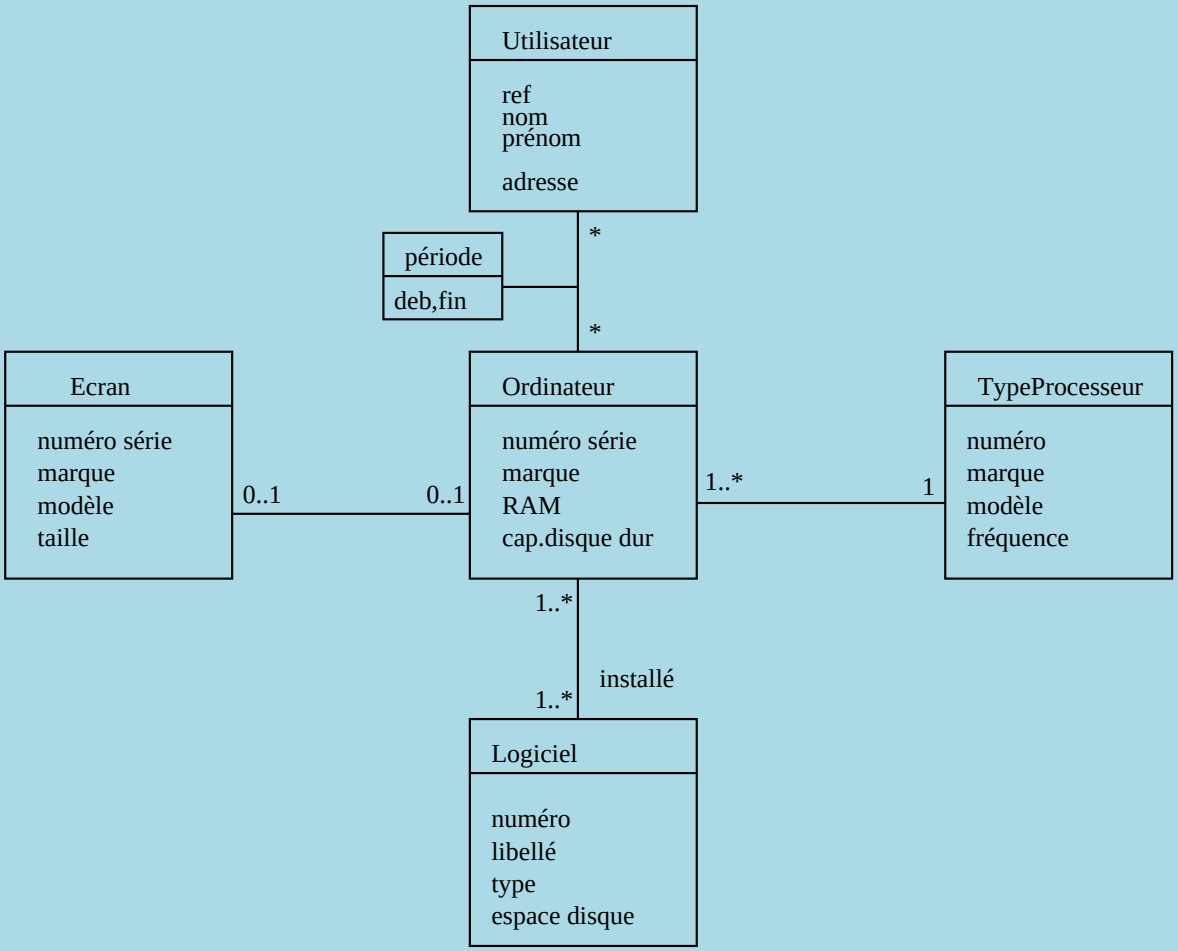


Contraintes

Il existe des contraintes sur les liens, qui ne sont pas visibles sur le schéma navigationnel.

- √ contraintes de liens inverses. Exemple : relation père-fils
  $\forall p \in$ Personne $p \in p$.pere.fils
  $\forall p \in$ Personne $p = p$.fils.pere
- √ contraintes ensemblistes. Exemple : affectation de stages.
  $\forall p \in$ Etudiant p.affectation $\in p$.voeux

Exemple



Présentation d'Oracle 8
Comparaison avec SQL3

- √ Définitions de types et de sous-types.
- √ Types abstraits de données (ADT) = types d'objets.
- √ Pointeurs (attributs REF).
- √ Collections :
  - Tables imbriquées (Nested Tables)
  - Tableaux prédimensionnés (VARRAY)
- √ Pas d'héritage (ni sur les ADT, ni sur les tables).

Association 1-1

```
create type ordi_type ;


create type ecran_type as object(
  numserie NUMBER(2),
  marque VARCHAR2(20),
  modele VARCHAR2(20),
  taille NUMBER(2),
  ordi REF ordi_type
)
```

```
create type ordi_type as object(
  numserie NUMBER(2),
  marque VARCHAR2(20),
  RAM NUMBER(4),
  capDisqueDur NUMBER(3),
  ecran REF ecran_type,
  ...)
```

Association 1-N

```
create type refOrdi_type as OBJECT (refOrdi REF ordi_type)

create type ensOrdi_type as TABLE of refOrdi_type
```

```
create type typeProc_type
as object(
  marque VARCHAR2(20),
  modele VARCHAR2(20),
  frequence NUMBER(5),
  lesOrdi ensOrdi_type
)
```

```
create type ordi_type as object(
  ...
  typeProc REF typeProc_type,
  ...)
```

Association N-N sans propriété

```
create type logiciel_type

create type reflogi_type as OBJECT (refLogi REF logiciel_type)

create type enslogi_type as TABLE of reflogi_type
```

```
create type logiciel_type
as OBJECT(
  numero NUMBER(2),
  libelle VARCHAR2(50),
  espaceDisque NUMBER,
  installeSur ensOrdi_type
)
```

```
create type ordi_type as object(
  ...
  logiciels enslogi_type,
  ...)
```

Association N-N avec propriété (1/2)

Création d'une classe pour l'association "utilise".

```
create type utilisateur_type ;

create type periodeUtil_type as OBJECT (
  debut date,
  fin date,
  utilisateur REF utilisateur_type,
  ordi REF ordi_type
)

create type refperiodeUtil_type
as OBJECT (periode REF periodeUtil_type)

create type ensPeriodeUtils_type as TABLE of refperiodeUtil_type
```

Association N-N avec propriété (2/2)

```
create type adresse_type
as object(
batiment VARCHAR2(20),
bureau VARCHAR2(10)
)

create type utilisateur_type
as OBJECT (
  numero NUMBER(2),
  nom VARCHAR2(20),
  prenom VARCHAR2(20),
  adresse adresse_type,
  utilise ensPeriodeUtils_type
)
```

```
create type ordi_type as object(
  ...
  utilisePar ensPeriodeUtils_type)
/
```

Création des tables et des contraintes (1/2)

```
create table ecran of ecran_type(
  constraint ecran_pkey primary key(numserie),
  constraint marque_ecran_non_null marque not null,
  constraint modele_ecran_non_null modele not null
);

create table typeProc of typeProc_type (
  constraint marque_proc_non_null marque not null,
  constraint modele_proc_non_null modele not null
) NESTED TABLE lesOrdi STORE AS tab_lesOrdi ;

create table logiciel of logiciel_type (
  constraint libelle_log_non_null libelle not null
) NESTED TABLE installeSur STORE AS tab_installeSur ;
```

Création des tables et des contraintes (2/2)

```
create table utilise of periodeUtil_type ;
create table utilisateur of utilisateur_type(
  constraint utilisateur_pkey primary key(numero)
) NESTED TABLE utilise STORE AS tab_utilise ;

create table ordi of ordi_type(
  constraint ordi_pkey primary key(numserie),
  constraint marque_ordi_non_null marque not null
) NESTED TABLE logiciels STORE AS tab_logiciels,
  NESTED TABLE utilisePar STORE AS tab_utilisePar ;
```

Le SGBD Objet-Relationnel Postgres

- √ Identité des T-uples

Le SGBD Objet-Relationnel Postgres

- √ Identité des T-uples
- √ Des attributs tableaux

Le SGBD Objet-Relationnel Postgres

- √ Identité des T-uples
- √ Des attributs tableaux
- √ Héritage de table

Le SGBD Objet-Relationnel Postgres

- √ Identité des T-uples
- √ Des attributs tableaux
- √ Héritage de table
- √ *Système de règles*

Identité de T-uple

- √ Chaque t-uple a une identité gérée par postgres (oid)
- √ Exemple :

```
create table personne (num integer primary key,
                       nom varchar(20) not null) ;
insert into personne values (1,'dupont') ;
INSERT 18829 1

select oid, * from personne ;
  oid  | num |  nom
-------+-----+--------
 18829 |   1 | dupont
(1 row)
```

Des attributs tableaux

- √ Possibilité de définir des tableaux à plusieurs dimensions

Des attributs tableaux

- √ Possibilité de définir des tableaux à plusieurs dimensions
- √ On peut fixer ou pas la taille des tableaux

Des attributs tableaux

- √ Possibilité de définir des tableaux à plusieurs dimensions
- √ On peut fixer ou pas la taille des tableaux
- √ Exemple :

```
create table groupe (numg integer primary key,
                     nom  varchar(20) not null unique,
                     membres integer []) ;
insert into groupe values (1,'clients', '{2, 4, 6}') ;

select membres[1] as premier_membre from groupe ;
select membres[2:3] as deux_derniers_membres from groupe ;
```

L'héritage (1/2)

√ Le premier SGBD !

L'héritage (1/2)

- √ Le premier SGBD !
- √ Possibilité d'héritage simple de table

L'héritage (1/2)

- √ Le premier SGBD !
- √ Possibilité d'héritage simple de table
- √ Possibilité d'héritage multiple !

L'héritage (1/2)

- √ Le premier SGBD !
- √ Possibilité d'héritage simple de table
- √ Possibilité d'héritage multiple !
- √ Prise en compte de l'extension des tables héritées

L'héritage (1/2)

- √ Le premier SGBD !
- √ Possibilité d'héritage simple de table
- √ Possibilité d'héritage multiple !
- √ Prise en compte de l'extension des tables héritées
- √ Distinction des types

L'héritage (2/2)

```
create table personnel(
  num int primary key,
  nom text not null,
  salaire float) ;

create table chef
 (prime float)
 inherits (personnel) ;

insert into personnel values (
 1,'dupont', 7000.0) ;
insert into chef values (
2, 'durant', 10000.0, 2000.0) ;
```

L'héritage (2/2)

```
create table personnel(
  num int primary key,
  nom text not null,
  salaire float) ;

create table chef
 (prime float)
 inherits (personnel) ;

insert into personnel values (
 1,'dupont', 7000.0) ;
insert into chef values (
2, 'durant', 10000.0, 2000.0) ;
```

```
select * from personnel ;
 num |  nom   | salaire
-----+--------+---------
   1 | dupont |    7000
(1 row)

select * from personnel* ;
 num |  nom   | salaire
-----+--------+---------
   1 | dupont |    7000
   2 | durant |   10000
(2 rows)
```

le système de règles

- √ Innovation Postgres

le système de règles

- √ Innovation Postgres
- √ Précurseur des triggers

le système de règles

- √ Innovation Postgres
- √ Précurseur des triggers
- √ Définition de comportement en fonction d'évènements

le système de règles

- √ Innovation Postgres
- √ Précurseur des triggers
- √ Définition de comportement en fonction d'évènements
- √ Gestion délicate !

Définition de règles

√ Syntaxe :

```
CREATE RULE name AS ON event
    TO object [ WHERE condition ]
    DO [ INSTEAD ] [ action | NOTHING ]
```

√ `name` : le nom de la règle

`event` : select, update, delete ou insert

`object` : une table ou une colonne d'une table

`condition` : n'importe quelle clause SQL

`action` : n'importe quelle action SQL

Utilisation de règles

- √ Deux nouvelles variables : `old` et `new`
- √ Un exemple :

```
CREATE RULE ajuster_salaire AS
    ON UPDATE personnel.salaire WHERE old.nom = "dupont"
    DO
        UPDATE personnel
        SET salaire = new.salaire
        WHERE personnel.nom = "dupond";
```

- √ Attention aux boucles !

Règle $\Longrightarrow$ sécurité

```
CREATE RULE preserver_salaire AS
    ON
        SELECT TO personnel.salaire
        WHERE current_user = "dupont"
    DO INSTEAD NOTHING;

select nom from personnel           -> ok
select * from personnel             -> rien pour 'dupont'
```