OLAP LHAJ Base de données OO (V)

- modèle de données ⟷ structure de données / langage de manipulation
- modèle relationnel ⟷ table, type, langage SQL
  langage de manipulation (SQL)

=> pour présenter la sémantique: la contrainte du domaine que je peux pas faire avec SQL

=> le modèle relationnel : Divise l'entité en petit morceau descriptif par morceau

→ 1) la contrainte d'intégrité référentielle est la contrainte la plus dure, surveille simultanément deux tables

→ 2) opérateur de jointure:

① + ② produit des morceaux donc j'ai besoin d'un langage pour reconstituer mon entité de départ.

=> Donc on ne peut le faire qu'avec SQL

- SQL + langage de programation révèle des incompatibilité qu'on doit gérer avec un langage intégré par SGBD.

=> le modèle objet est un langage de programation complet

classe CPARCELLES {
private int IDPARC;
⋮

- Cela peut révéler un lien réflexive [Autojointure sur la même table]

- requete par navigation: PERS.PERE.getEnfants() ∩ PERS.MERE.getEnfants()

2) • Le modèle relationnel ne manipule que des valeurs mais pas les entités. D'où l'apport du modèle objet
• on stocke les objets mais pas les tuples

De même :

Aspect Bdd	Aspect OO
• persistence	• identité de l'objet
• fiabilité	• Encapsulation
• sécurité	• classes

• Les règles facultatives → Héritage multiple
• Les règles ouvertes → Nommage uniforme

• Aspect OO :

B hérite A càd typeB ⊂ typeA

Surcharge et résolution tardive :

Polymorphisme nom : nom désigne plusieurs choses
Polymorphisme type : type qui désigne plusieurs types différents
→ on fait le typage dynamique au moment de l'exécution

• L'Objet occupe un espace physique (new — ())
• classe peut être un objet (stocker les objets instanciés dans cette classe)
• classe est un schema qui nous permet de stocker nos objets.
• identité d'objet découle de la persistence
• objet temporaire : les gens qui ne me doit rien (pas de relations permanentes)
• Objet persistant : la famille et les amis (de relations permanentes)

SGBD
→ ne pas le reconnaître par la valeur de l'objet (valeur peut changer)
→ ne pas le reconnaître par adresse physique (adresse peut changer)
→ Valeurs de primary key peut changer

- pour le connaître on utilise l'identité de l'objet (OID) (3 c'est une grande valeur associé à chaque objet persistant)
  - Objet persistant <OID, VALUES>
    - SGBDOO → OID
    - VALUES ← donnée par le développeur
  - $q.b = O_2 \rightarrow b = OID(O_2)$ ⚠ (pas de contrainte d'intégrité référentielle)
- un Objet n'est jamais dupliqué (modification dans un espace unique)

Test d'égalité: == ✓ (profond) = ✓ (de surface)

→ Persistance → Nommage

- OID ≠ Primary key (car primary key peut changer)

SGBDOO	SGBDR
• développeur sollicité il attribue la value à un objet persistant	• développeur non sollicité

~~Le modèle relationnel étendu (Sect 3)~~

→ un objet temporaire peut devenir persistant

- persistant par classe : Create persistant class Personne Type Tuple (nom : String, Voiture : véhicule)
- persistant par objet : personne pers = new Personne() Vehicule BMW = new Vehicule()

  pers → nom = 'Farid'

  pers → voiture = *BMW

. le nommage ⟹ persistance par transitivité

- On gère le modèle relationnel au niveau logique : Table
- de SGBD descend dans le modèle objet avec les racines de persistance

ex: on crée chien et on l'ajoute dans la persistance & Chien et du coup il devient persistant (par transitivité)

- racine de persistence ≡ table
- objet ≡ tuple

BDOO = programmation objet + persistance

- pas de contrainte d'intégrité dans le modèle objet

⇒ Intro SQL3 (Relationnel etendu) → Oracle → informix ⋮

- Noyau SQL2 + Extension objet

⇒ SQL3 plus de puissance que SQL2 mais il n'est plus normalisé + contrainte primary key n'est plus exigée

- pour étendre le relationnel on passe du monovalué au multivalué
- Concept du modèle objet implementé dans SQL3 → collections

objet complexes — identité objet (Reference) — heritage et encapsulation

⚠ pas comme OO

⚠ généricité : concerne les structures de données qui ~~contiennent~~ peut contenir des types différents

+ on définit une classe abstraite dans laquelle
- on peut tirer des classes ⇐⇒ chien, chat, humain
- on peut créer un tableau où on stocke (chien, chat, ...)

NB → les types abstraits ≡ classes

```
CREATE TYPE t_adresse As OBJECT (
        Numero  Number(4)
        ⋮
);
```

CREATE TABLE ADRS OF t_adresse(

=

- Il va ajouter REF ≡ l'identifiant objet

CREATE TABLE adresses OF t_adresse
Va remplacer ADRESSE par OID t_adresse

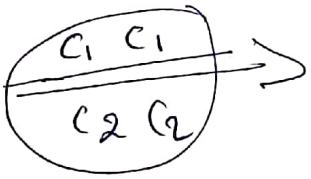
- Objet imbriqués n'auront pas d'identité
- Ref est un charge de développeur dans SQL3 cherche la ref et l'insérer dans adresse (usage des pointeurs)

- Encapsulation : cacher les données et les faire apparaître dans les méthodes (inutile dans le relationnel) on crée des méthodes :

SGBDOO

C1 méth11

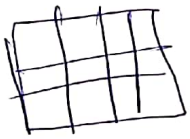
C2 méth21



OK

SGBDR

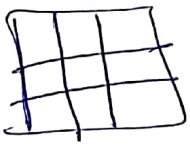
C1 méth11



??

NON

C2 méth21



Nutype
pas de méthodes

NB : l'importance des collections dans SQL3 : implémenter les associations dans la classe mère sans passer par des tables intermédiaires sans besoin de l'op de jointures

- Héritage ⟶ très compliqué au niveau OO.