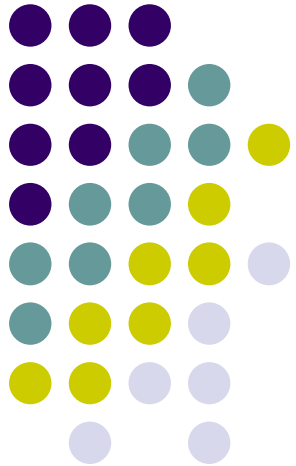
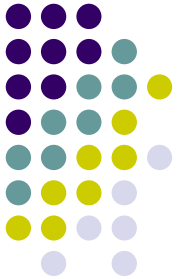
Bases de Données

Meltem Öztürk

- Introduction
- Modèle Entité/Association
- Modèle relationnel
- Algèbre relationnelle
- SQL



Bases de données



- 24h Cours, 24h TD, 12h TP
- note finale= TP/5+4/5*sup(EX, (CC+EX)/2)
- References:
  - Ch. Date *Introduction aux bases de données,* Vuibert, Paris 2004
  - G. Gardarin, *Bases de Données - objet/relationnel*, Eyrolles, 1999
  - Polycopiés de Ph. Rigaux, M. Manouvrier et St. Gançarski

Introduction

Base de données:

- collection d'informations ou de données qui existent sur une ***longue période de temps*** et qui décrivent les activités d'une ou plusieurs organisations

- ensemble de données ***modélisant les objets d'une partie du monde réel*** et servant de support à une application informatique

- un gros ensemble d'informations ***structurées mémorisées*** sur un support permanent

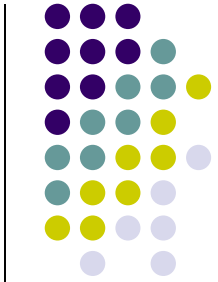
Introduction

SGBD

Systèmes de Gestion de Bases de Données (*DataBase Management Systems - DBMS*)

ensemble de logiciels systèmes permettant aux utilisateurs de faire des ***applications*** (insérer, modifier, et rechercher) efficacement des données spécifiques dans ***une grande masse d'informations*** (pouvant atteindre plusieurs milliards d'octets) ***partagée par de multiples utilisateurs***

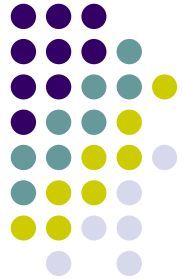
Introduction



Exemples de SGBD:

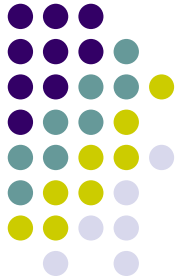
- BD d'université (données sur les étudiants, les enseignements, les salles, etc.)
- BD de compagnie aérienne (données sur les clients, les vols, les réservations, etc.)
- BD bancaire (données sur les clients, les comptes, les transactions, etc.)

Problèmes en absence de SGB



- Programmes d ’application écrits directement au-dessus du système de gestion de fichier
  - redondance (coût de stockage et d ’accès)
  - incohérence (ex: changement d ’adresse)
  - difficulté d ’accès (requêtes non prévues dans les programmes)
  - isolation des données (nouveau programme qui cherche des données dans des fichiers variés de différents formats)
  - manque de sécurité
  - gestion de l ’intégrité (obéir à des contraintes)

Introduction



Principaux composants d'un SGBD

- **Système de gestion de fichiers (physique)**

  Stockage et accès des fichiers…

- **Gestionnaire de requêtes**

  Traduction des requête des mis à jour et d'interrogation

- **Gestionnaire de transactions**

  Regroupement des actions (modifications, mises à jour, etc.) qui doivent être exécutées ensemble séquentiellement (ex : virement d'une somme de « A » à « B », lire la somme de A, effacer de A, ajouter à B, etc.)

Introduction

Principales fonctionnalités d'un SGBD:

- **Contrôle de la redondance d'information**
- **Partage des données**
- **Gestion des autorisations d'accès**
- **Vérifications des contraintes d'intégrité**

  Contraintes structurelles (un employé a un seul chef), contraintes dynamiques (un salaire ne peut diminuer), etc.
- **Sécurité et reprise sur panne**

Introduction

Abstraction de données : 3 niveaux ***(abstraction des données, indépendance entre utilisateurs et gestion):***

- **Niveau interne ou physique** :
  - plus bas niveau
  - indique **comment** (avec quelles structures de données) sont stockées physiquement les données
- **Niveau logique ou conceptuel**
- **Niveau externe ou vue**

Introduction

Abstraction de données : 3 niveaux

- **Niveau interne ou physique**
- **Niveau logique ou conceptuel** :
  - décrit par un ***schéma conceptuel***
  - indique quelles sont les données stockées et quelles sont leurs relations indépendamment de l'implantation physique
- **Niveau externe ou vue**

Introduction

Abstraction de données : 3 niveaux

- **Niveau interne ou physique**
- **Niveau logique ou conceptuel**
- **Niveau externe ou vue** :
  - **propre à chaque utilisateur**
  - décrit par un ou plusieurs ***schémas externes***

Introduction

Modèle de données ***(décrire les données, les relations entre elles, leur sémantique, les contraintes d 'intégrité, etc.):***

- Modèle conceptuel (entité/association)
  - Plus lisibles (graphiques)
  - Entité, association, spécialisation, attribut, identificateur, etc.

- Modèle logique (logique relationnel)
  - Plus facilement implantable
  - Relation, attribut, domaine, clé, n-uplet, etc.

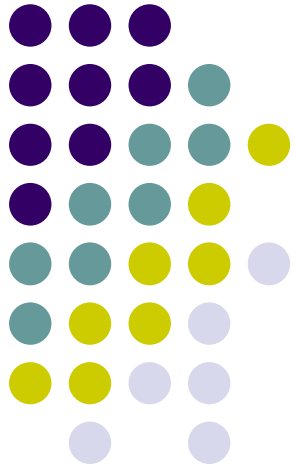
Introduction

Différents langages d ’un SGBD :

- ***LDD*** : Langage de Définition de Données,
  - construire un schéma pour décrire la structure, incluant les contraintes
- ***LMD*** : Langage de Manipulation de Données
  - appliquer les opérations aux données (retrouver et mettre à jour les données)

Bases de Données

- Introduction
- Modèle Entité/Association
- Modèle relationnel
- Algèbre relationnelle
- SQL



Exemple de base de données

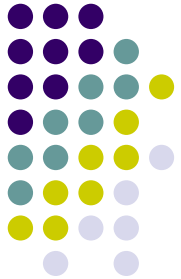
titre	année	nomMES	prénomMES	annéeNaiss
Alien	1979	Scott	Ridley	1943
Vertigo	1958	Hitchcock	Alfred	1899
Psychose	1960	Hitchcock	Alfred	1899
Kagemusha	1980	Kurosawa	Akira	1910
Volte-face	1997	Woo	John	1946
Pulp Fiction	1995	Tarantino	Quentin	1963
Titanic	1997	Cameron	James	1954
Sacrifice	1986	Tarkovski	Andrei	1932

Exemple de base de données

Critiques sur notre exemple:

- Anomalie lors d'une insertion :
  - Insertion du même film plusieurs fois, et même avec différentes données!
  - Comment distinguer un film d'un autre?
- Anomalie lors d'une modification
  - Si on modifie la date de naissance de Hitchcock?
- Anomalie lors d'une destruction
  - Si on supprime un film, on supprime toutes données associées, y compris celles du réalisateur

Bonne méthode:



- représenter individuellement les films et les réalisateurs (une action sur l ’un n ’entraine pas systématiquement une action sur l ’autre)
- méthode d ’identification d ’un film ou d ’un réalisateur (permet d ’assurer que la même information est représentée une seule fois)
- préserver le lien entre les films et les réalisateurs

Changeons notre modèle

titre	année
Alien	1979
Vertigo	1958
Psychose	1960
Kagemusha	1980
Volte-face	1997
Pulp Fiction	1995
Titanic	1997
Sacrifice	1986

La table des films

id	nomMES	prénomMES	annéeNaiss
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899
3	Kurosawa	Akira	1910
4	Woo	John	1946
5	Tarantino	Quentin	1963
6	Cameron	James	1954
7	Tarkovski	Andrei	1932

La table des réalisateurs

Changeons notre modèle

titre	année	*idMES*
Alien	1979	1
Vertigo	1958	2
Psychose	1960	2
Kagemusha	1980	3
Volte-face	1997	4
Pulp Fiction	1995	5
Titanic	1997	6
Sacrifice	1986	7

La table des films

id	nomMES	prénomMES	annéeNaiss
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899
3	Kurosawa	Akira	1910
4	Woo	John	1946
5	Tarantino	Quentin	1963
6	Cameron	James	1954
7	Tarkovski	Andrei	1932

La table des réalisateurs

Schéma de notre exemple

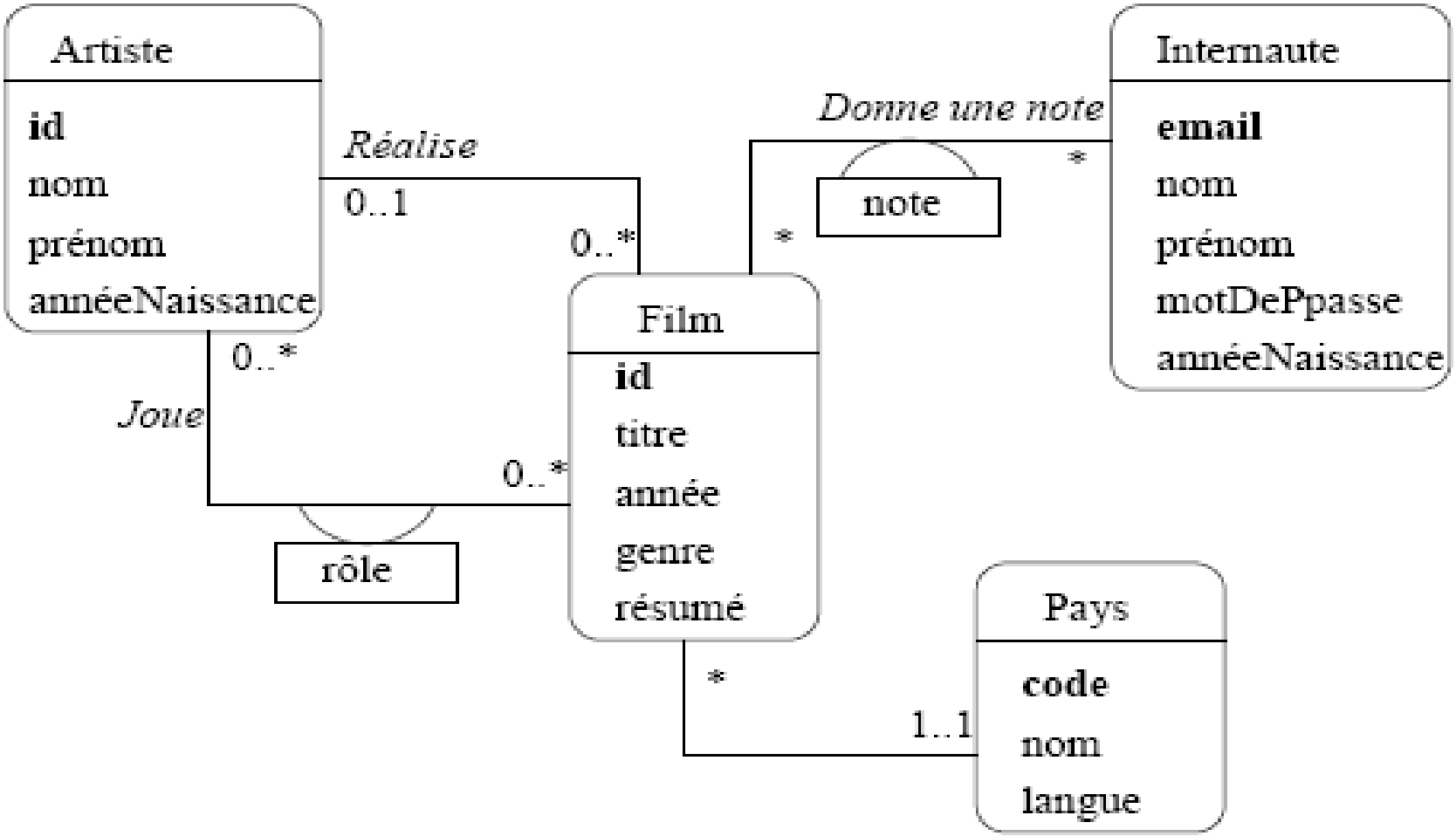


FIG. 3.1 – *Le schéma de la base de données Films*

Modèle Entité/Association

- ***Entité*** : objet concret ou abstrait, identifiable, décrit par l'information et pertinent pour l'application
  - Ex : Vertigo (film), Hitckcock (réalisateurs)

- Une entité est représentée par un ensemble d' ***attributs*** qui la décrive.
  - Ex : film décrit par le nom, l'année de création, …

- Chaque attribut a ***un domaine*** qui correspond à l'ensemble des valeurs qu'il peut prendre
  - Ex : les années compris entre 1920-2005

Type d'entité

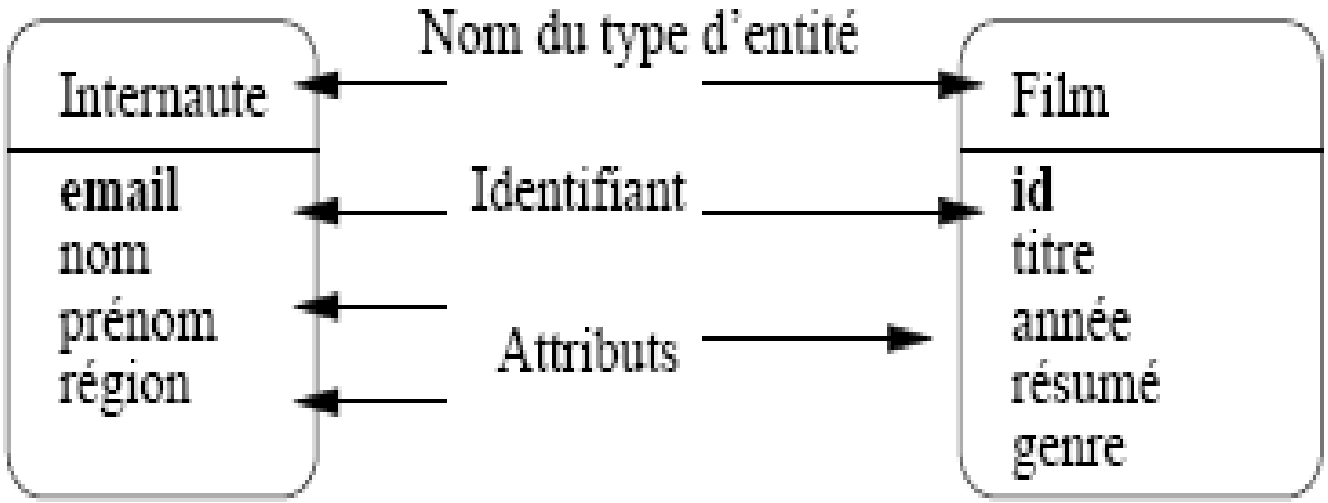


FIG. 3.2 – *Représentation des types d'entité*

Modèle Entité/Association

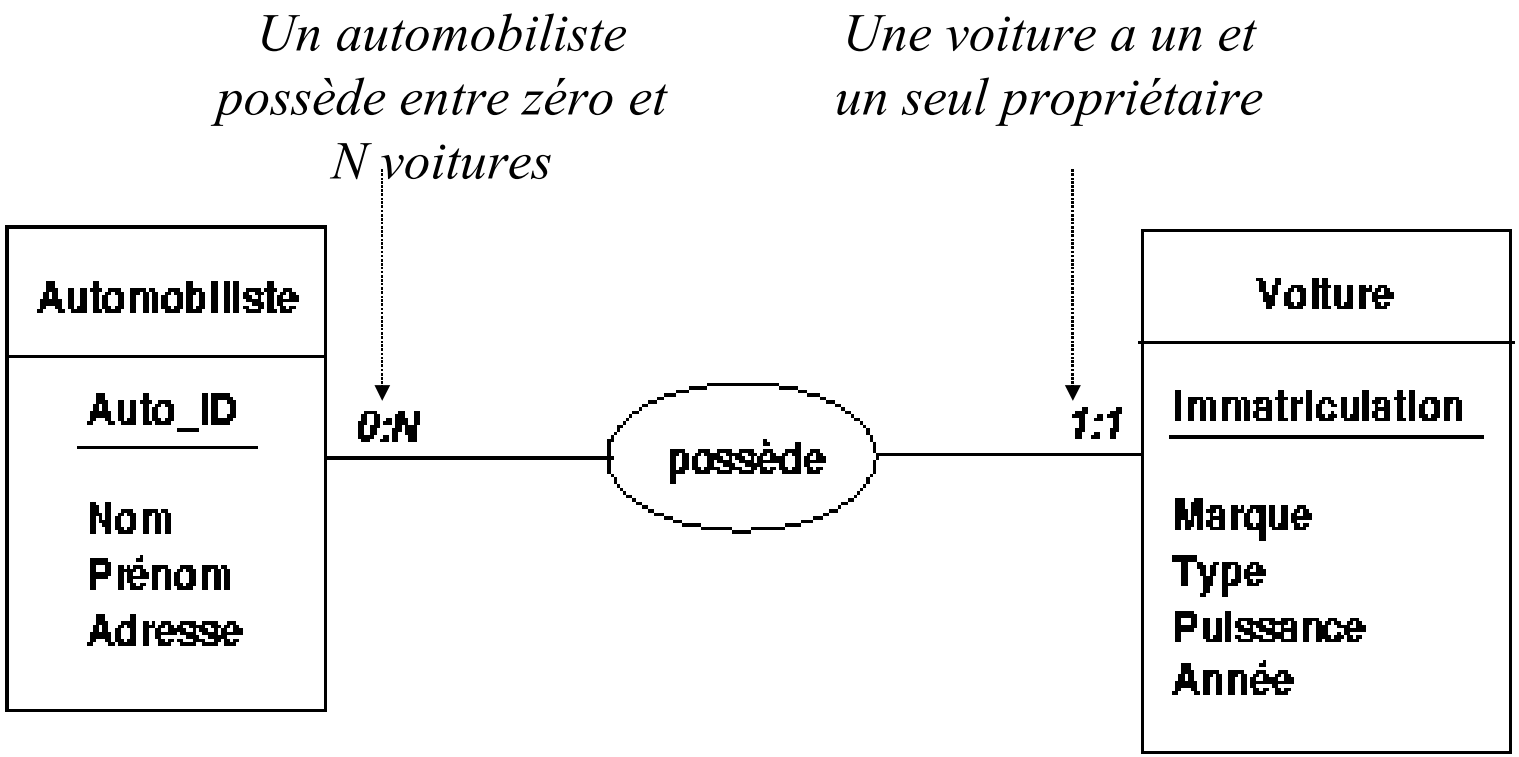
- ***Clé (Identificateur)*** : Un sous-ensemble d'attribut permettant d'identifier l'entité de manière unique.
  - Ex : le nom du film, numéro d'identifiant du réalisateur

- Une ***association*** correspond à une relation entre entité et peut avoir des attributs
  - Ex: Hitchcock *a réalisé* Vertigo

- ***Cardinalité*** d'une association : nombre d'entité que l'association relie.
  - Noté (Min, Max)
  - « * » : « 0. * », « 1 »: « 1.1 »

- ***Clé d'une association*** : couple formé des clés des deux entités

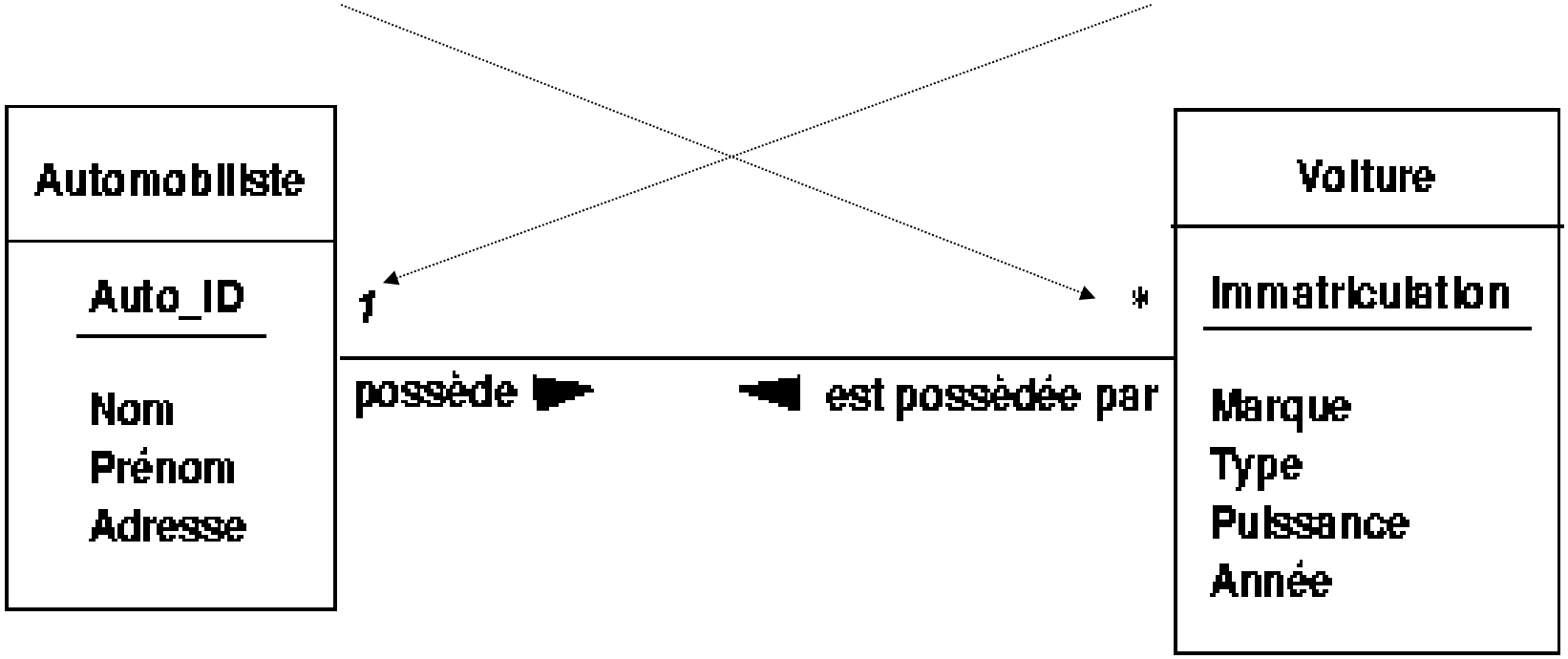
Format Merise



Modélisation UML

Un automobiliste possède entre zéro et N voitures

Une voiture a un et un seul propriétaire



Attribut d'une association

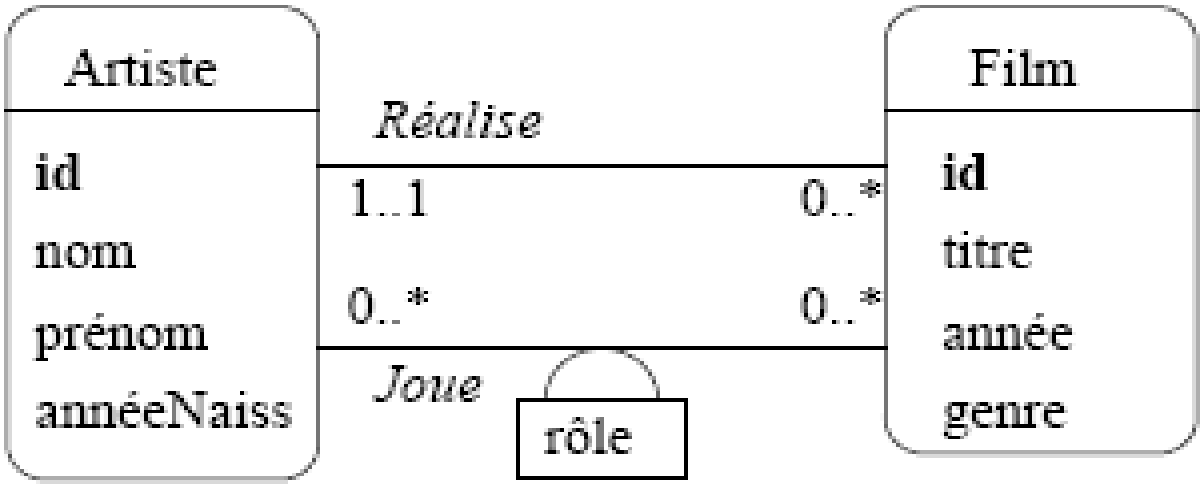


FIG. 3.6 – *Associations entre* Artiste *et* Film.

Association n-aires

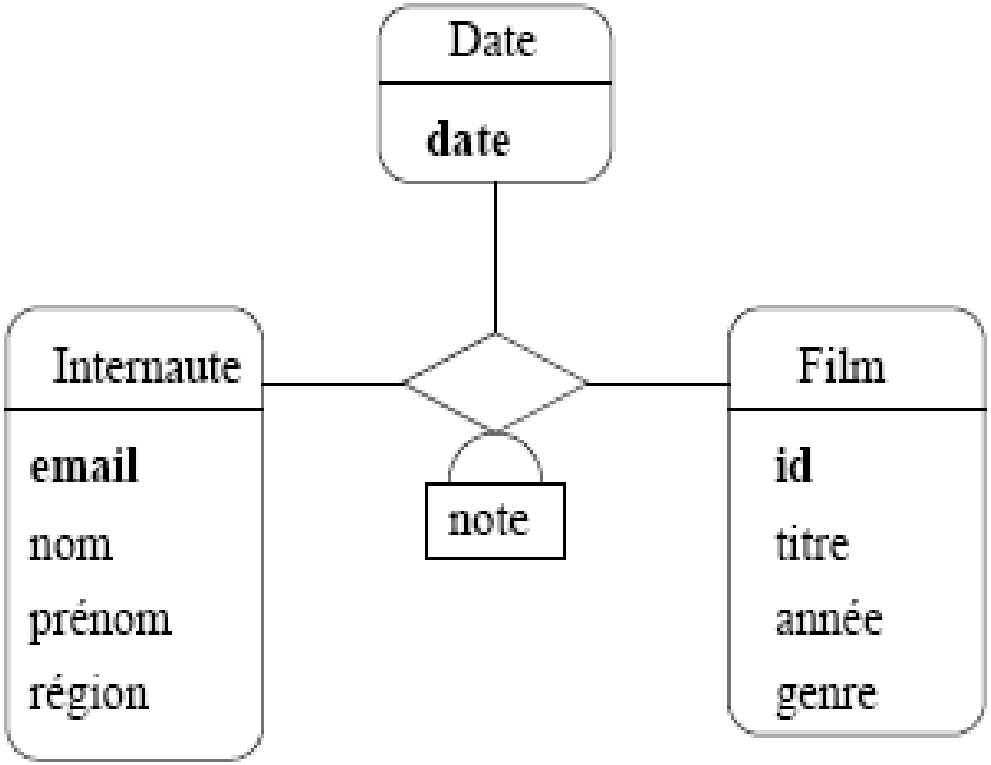


FIG. 3.7 – *Ajout d'une entité* Date *pour conserver l'historique des notations*

Association n-aire (cardinalité « 0.*», clé ?)

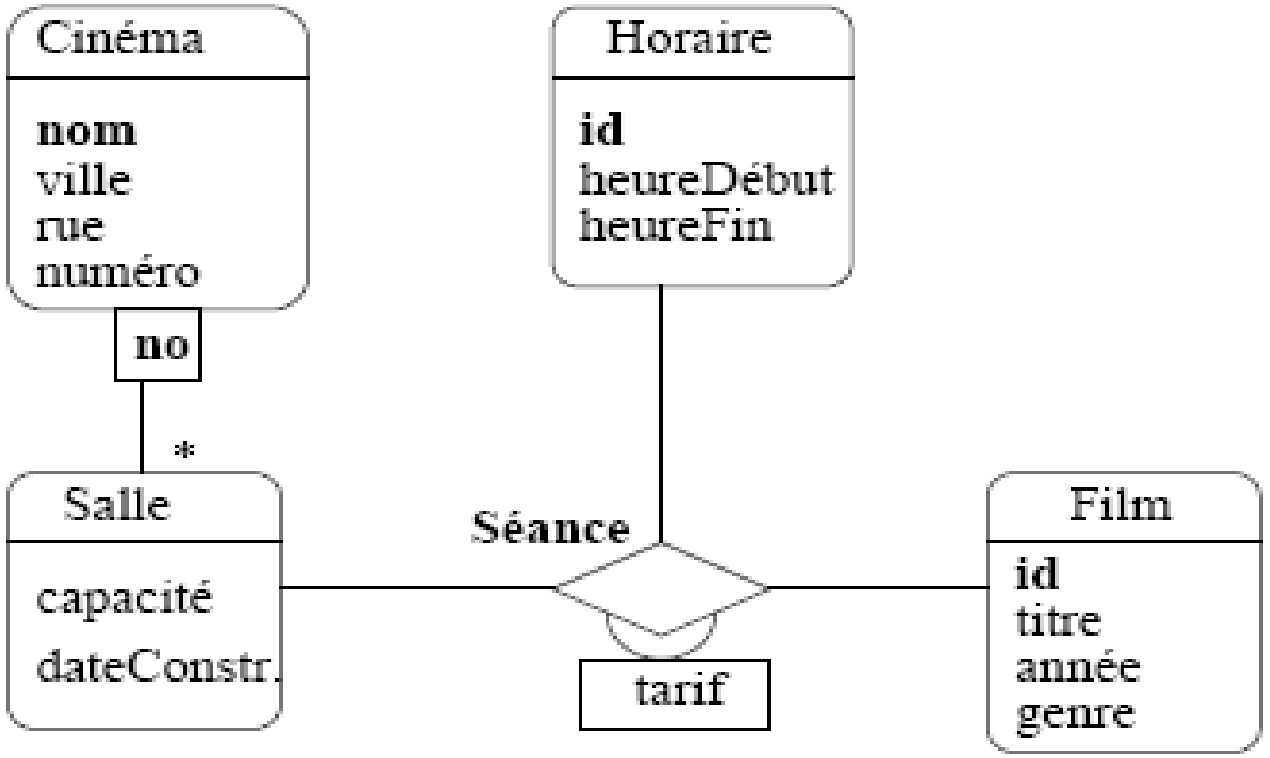


FIG. 3.11 – *Association ternaire représentant les séances*

Association ternaire - - > entité

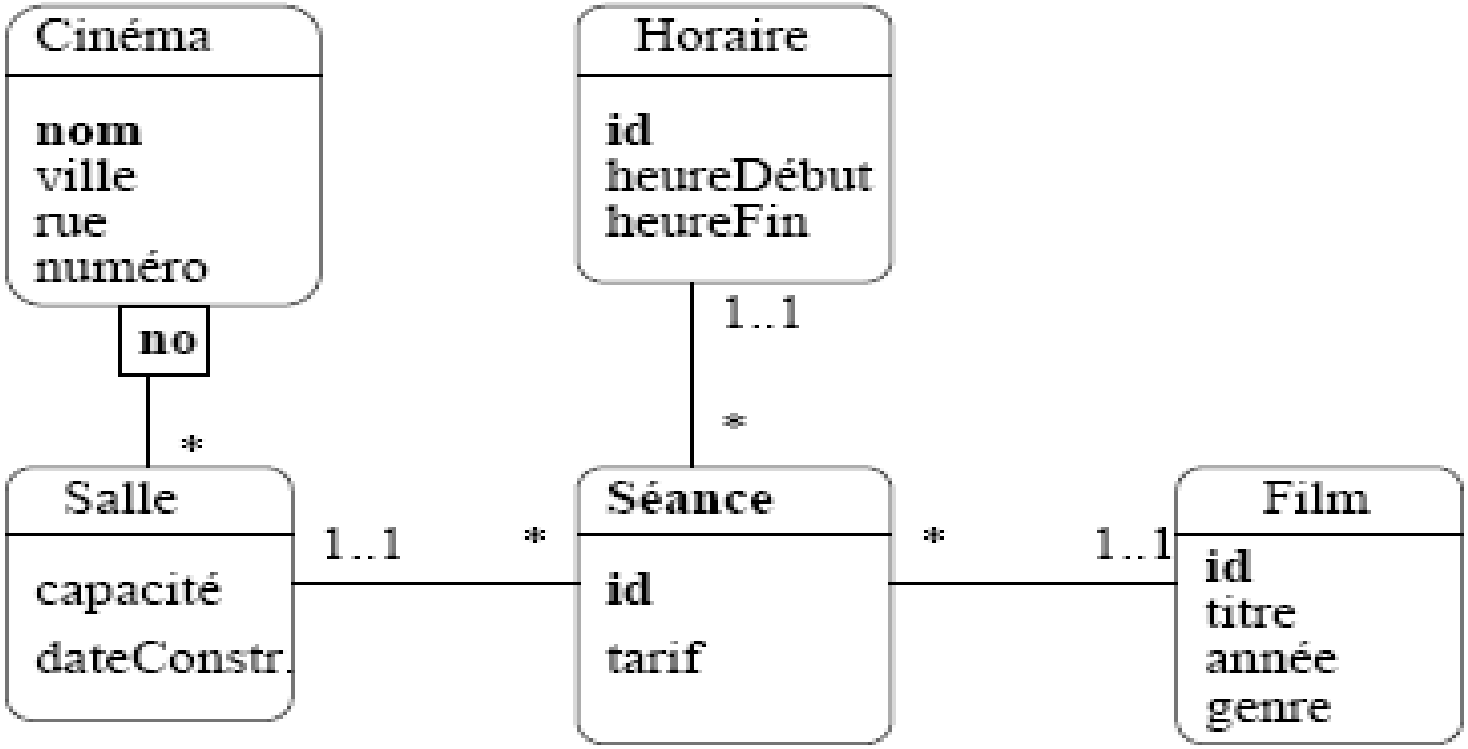
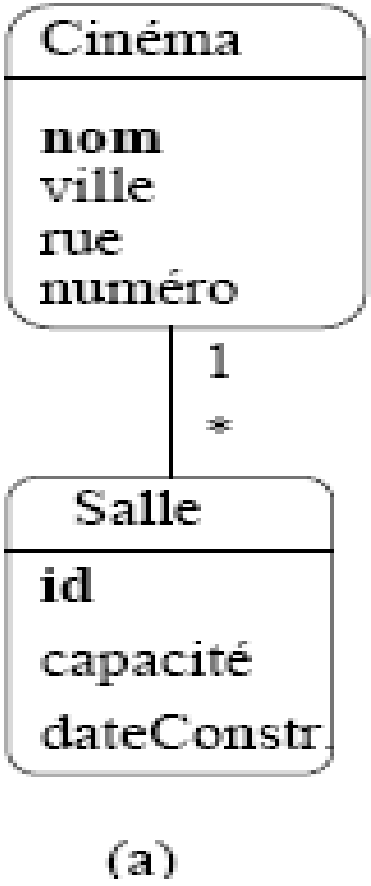


FIG. 3.13 – *L'association* Séance *transformée en entité*

Entité faible



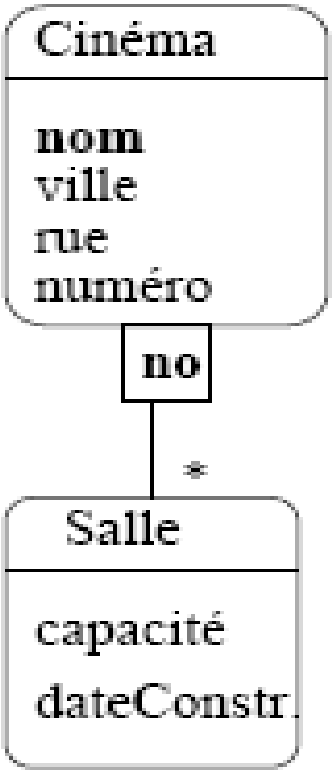
(a)

Chaque salle est notée par un numéro

Il y a tant de numéro que le nombre de salles

Numérotation indépendant du cinéma

Entité faible



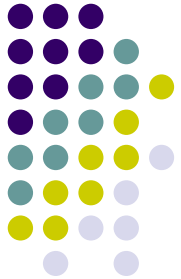
(b)

Chaque salle a un numéro unique dans un cinéma donné

Ex. Salle 1 du cinéma A et Salle 1 du cinéma C

Pour distinguer une salle d'une autre, il faut connaître le cinéma auquel elle est rattachée

(Des)Avantages du modèle E/A

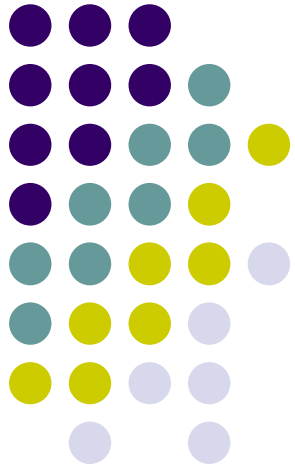


- Avantages
  - Que 3 concepts: entités, associations, attributs
  - Représentation graphique et rapide

- Désavantages
  - Pas de règle absolue pour déterminer qui est attribut, entité ou association…

Bases de Données

- Introduction
- Modèle Entité/Association
- Modèle relationnel
- Algèbre relationnelle
- SQL



Modèle relationnel

La relation du « nom » Film :

Film (titre: string, année: number, genre : string)

domaine

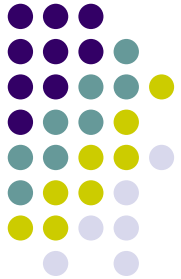
Attribut

N-uplet (tuple)
entité

titre	année	genre
Alien	1979	Science-Fiction
Vertigo	1958	Suspense
Volte-face	1997	Thriller
Pulp Fiction	1995	Policier

FIG. 4.1 – *Une relation*

Modèle relationnel

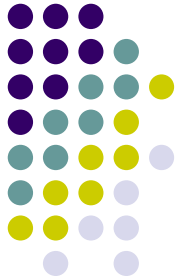


- ***Domaine*** : ensemble d'instance d'un type élémentaire (en extension ou en intension)
  - Ex : réels, boolean, chaine de caracteres, etc.

- ***Attribut*** : colonne d'une relation, associé a un domaine

- ***Schéma de relation*** : nom suivi de la liste des attributs, chq attribut étant associé à son domain
  - R(A1 : D1, A2 : D2, …, An : Dn)
  - Ex : *Film* (titre: string, année: number, genre : string)

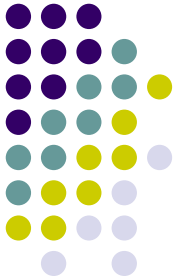
Modèle relationnel



- ***Relation (R)*** : sous-ens. fini du produit cartésien des domaines des attributs de R

  - représentée par une table à 2 dimensions
  - colonne = domaine du produit cartésien
  - même domaine peut apparaître +ieurs fois
  - ensemble de tuplets sans doublons (pas 2 fois même ligne)
  - ordre des lignes n'a pas d'importance
  - pas de case vide dans la table

Modèle relationnel



- ***Clé d'une relation*** : le + petit sous-ens. Des attributs qui permet d'identifier chq ligne d'une manière unique
  - ex : film (**titre**, année, genre)

- ***Tuple (n-uplet)*** : une liste de n valeurs (v1, …vn), où chq vi est la valeur d'un attribut Ai du domain Di
  - ex : ('Cyrano', 1992, 'Rappeneau')

- ***Base de données*** : ensemble fini de relations.

Passage de E/A au relationnel

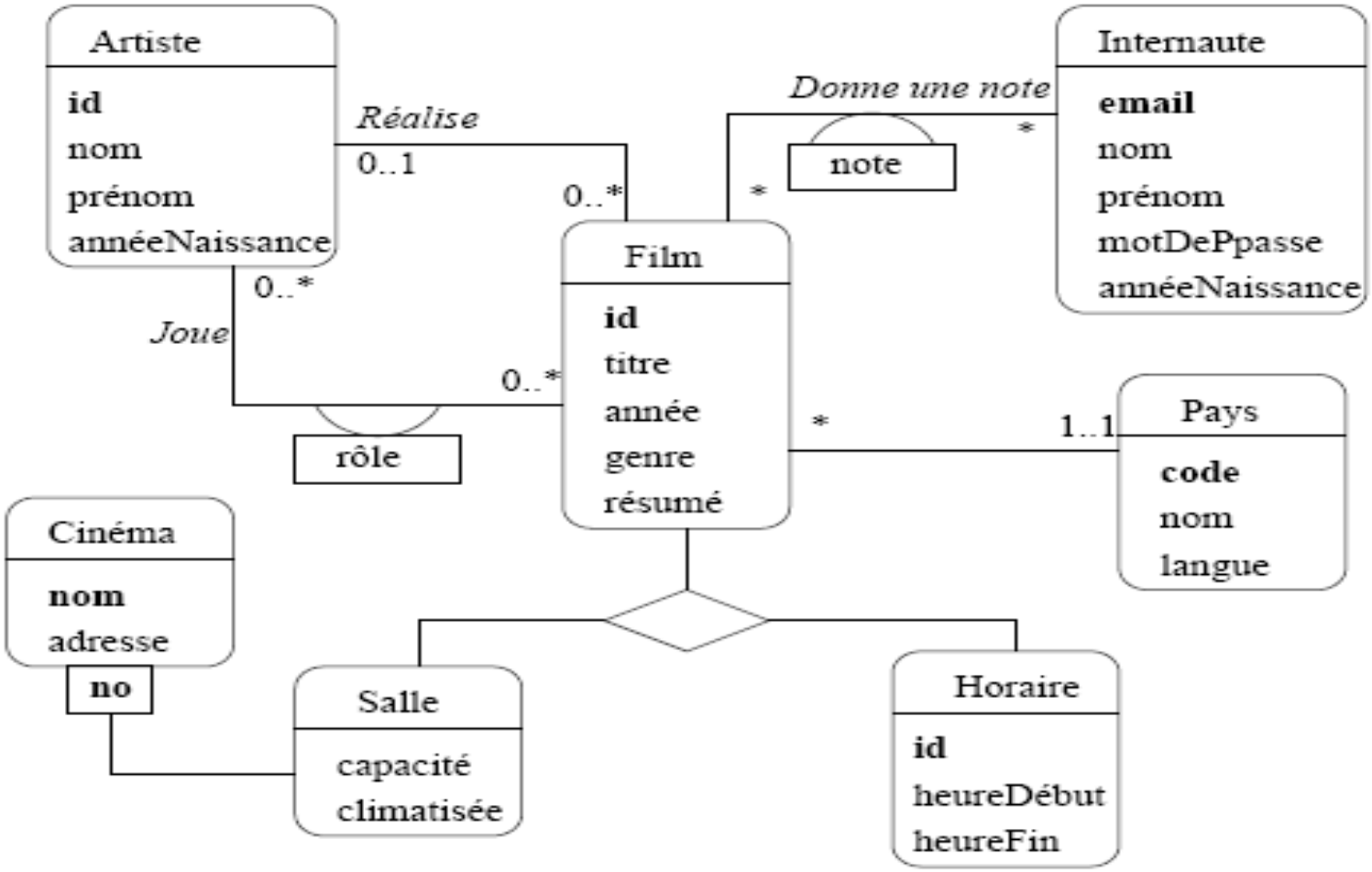
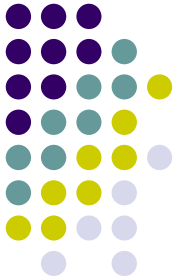


FIG. 4.2 – *Le schéma de la base de données Films*

Règles de passages (1)

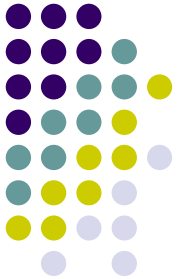


- Règles générales – Entité :
  1. On crée une relation de même nom que l'entité.
  2. Chaque propriété de l'entité, y compris l'identifiant, devient un attribut de la relation.
  3. Les attributs de l'identifiant constituent la clé de la relation.

  Ex:

  - *Film (**idFilm**, titre, année, genre, résumé)*
  - *Artiste (**idArtiste**, nom, prénom, annéeNaissance)*
  - *Internaute (**email**, nom, prénom, région)*
  - *Pays (**code**, nom, langue)*

Règles de passages (2)



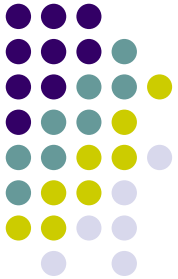
Association de un à plusieurs :

1. On crée les relations Ra et Rb correspondant respectivement aux entités A et B.
2. L'identifiant de B devient un attribut de Ra

ex:

- *Film (**idFilm**, titre, année, genre, résumé, idArtiste, codePays)*
- *Film (**idFilm**, titre, année, genre, résumé, idMes, codePays)*
- *Artiste (**idArtiste**, nom, prénom, annéeNaissance)*
- *Pays (**code**, nom, langue)*

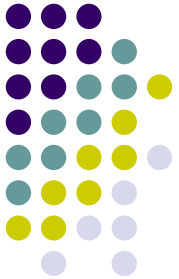
Exemple de passage



idFilm	titre	année	genre	*idMES*	*codePays*
100	Alien	1979	Science Fiction	1	US
101	Vertigo	1958	Suspense	2	US
102	Psychose	1960	Suspense	2	US
103	Kagemusha	1980	Drame	3	JP
104	Volte-face	1997	Action	4	US
105	Van Gogh	1991	Drame	8	FR
106	Titanic	1997	Drame	6	US
107	Sacrifice	1986	Drame	7	FR

La table *Film*

Règles de passage (3)

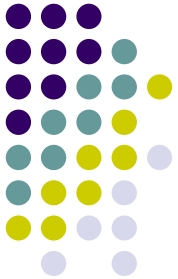


Association avec entité faible : même que le passage des associations du type de un à plusieurs

Ex:

- *Cinéma (**nomCinéma**, numéro, rue, ville)*
- *Salle (**nomCinéma, no**, capacité)*

Règles de passage (4)



Association binaire de plusieurs à plusieurs :

1. On crée les relations Ra et Rb des entités A et B.
2. On crée une relation Ra+b pour l'association
3. La clé de Ra et la clé de Rb deviennent des attributs de Ra+b
4. La clé de cette relation est la concaténation des clés des relations Ra et Rb
5. Les propriétés de l'association deviennent des attributs de Ra+b

Ex :

- *Film (**idFilm**, titre, année, genre, résumé, idMES, codePays)*
- *Artiste (**idArtiste**, nom, prénom, annéeNaissance)*
- *Role (**idFilm, idActeur**, nomRôle)*

Exemple de passage

Ex :

- *Film (**idFilm**, titre, année, genre, résumé, idMES, codePays)*
- *Artiste (**idArtiste**, nom, prénom, annéeNaissance)*
- *Role (**idFilm, idActeur**, nomRôle)*

idFilm	*idActeur*	rôle
20	100	William Munny
20	101	Little Bill
21	101	Bril
21	103	Robert Dean

La table *Rôle*

Règles de passage (5)

Association ternaire :

2. Même principe d'une association binaire:
   - Salle (***nomCinéma*, no**, capacité)
   - Film (**idFilm**, titre, année, genre, résumé, *idMES*, *codePays*)
   - Horaire (**idHoraire**, heureDébut, heureFin)
   - Séance (***idFilm***, ***nomCinéma***, ***noSalle***, ***idHoraire***, tarif)

idFilm	*nomCinéma*	*noSalle*	*idHoraire*	tarif
103	Le Rex	2	1	23
103	Le Rex	2	2	56
100	Kino	1	2	45
102	Le Rex	2	1	50

La table *Séance*

Exemple de passage

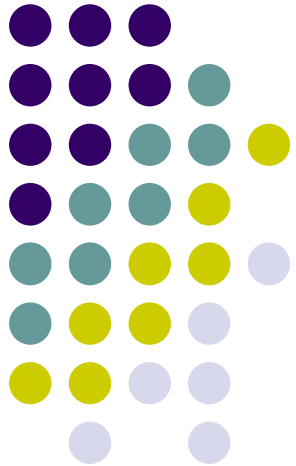
- Problème avec notre exemple : même salle présente deux films différents au même horraire

  - Salle (***nomCinéma*, no**, capacité)
  - Film (**idFilm**, titre, année, genre, résumé, *idMES*, *codePays*)
  - Horaire (**idHoraire**, heureDébut, heureFin)
  - Séance (***nomCinéma***, ***noSalle***, ***idHoraire***, tarif)

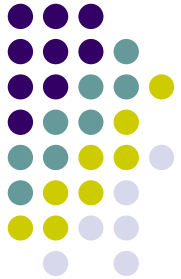
Clé de la relation est un sous-ensemble de la concatination des clés

Bases de Données

- Introduction
- Modèle Entité/Association
- Modèle relationnel
- Algèbre relationnelle
- SQL



Algèbre relationnelle



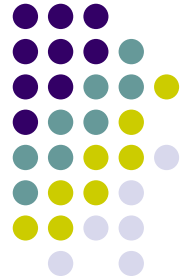
Algèbre : ensemble d'opérateurs manipulant des expressions dont le résultat est une expression qui peut être manipulée …

Algèbre relationnelle : ensemble d'opérateurs prenant en entrée une ou deux expressions relationnelles et produise une expression relationnelle à la sortie.

Algèbre relationnelle

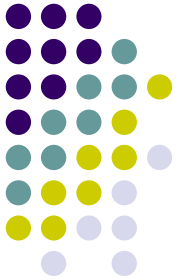
- Opérateurs fondamentaux
  - Opérateurs unaires
    - Sélection
    - Projection
  - Opérateurs binaires
    - Produit cartésien
    - Union
    - Différence
- Opérateurs dérivés
    - Jointure
    - …

Exemple:organisme de voyage



- Station (**nomStation**, capacité, lieu, région, tarif)
- Activite (***nomStation***, **libellé**, prix)
- Client (**id**, nom, prénom, ville, région, solde)
- Séjour (***idClient***, **station**, **début**, nbPlaces)

Algèbre relationnelle



La sélection $(\sigma_F (R))$: extrait de la relation *R* les tuples qui satisfont la critère de sélection *F*. *F* est une formule de logique:

- Constantes
- Attributs figurant dans la relation
- Comparateurs (=, <, >, etc.)
- Connecteurs logiques (ou, et, non, etc.)

Ex : toutes les stations aux Antilles: $\sigma_{région='Antilles'}(Station)$

nomStation	capacité	lieu	région	tarif
Venusa	350	Guadeloupe	Antilles	1200
Santalba	150	Martinique	Antilles	2000

Algèbre relationnelle

La projection $(\Pi_{A\,1\,,\,A2,\,\ldots,\,Ak}\,(R))$: ne garde que les attributs $A_1, A_2, \ldots, A_k$

Ex : les noms des stations et leur région / seulement les régions (pas de doublure de ligne)

$\Pi_{nomStation,\ region}(Station)$

nomStation	région
Venusa	Antilles
Farniente	Océan Indien
Santalba	Antilles
Passac	Europe

$\Pi_{nomSta\ldots}$ (Station)

région
Antilles
Océan Indien
Europe

Algèbre relationnelle

Produit cartésien *(R x S)* : crée une nouvelle relation où chaque tuple de *R* est associé à chaque tuple de *S*.

A	B
a	b
x	y

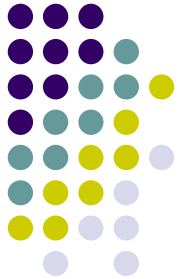
C	D
c	d
u	v
x	y

A	B	C	D
a	b	c	d
a	b	u	v
a	b	x	y
x	y	c	d
x	y	u	v
x	y	x	y

Exemple:

S.nomStation	cap.	lieu	région	tarif	A.nomStation	libelle	prix
Venusa	350	Guadeloupe	Antilles	1200	Venusa	Voile	150
Venusa	350	Guadeloupe	Antilles	1200	Venusa	Plongée	120
Venusa	350	Guadeloupe	Antilles	1200	Farniente	Plongée	130
Venusa	350	Guadeloupe	Antilles	1200	Passac	Ski	200
Venusa	350	Guadeloupe	Antilles	1200	Passac	Piscine	20
Venusa	350	Guadeloupe	Antilles	1200	Santalba	Kayac	50
Farniente	200	Seychelles	Océan Indien	1500	Venusa	Voile	150
Farniente	200	Seychelles	Océan Indien	1500	Venusa	Plongée	120
Farniente	200	Seychelles	Océan Indien	1500	Farniente	Plongée	130
Farniente	200	Seychelles	Océan Indien	1500	Passac	Ski	200
Farniente	200	Seychelles	Océan Indien	1500	Passac	Piscine	20
Farniente	200	Seychelles	Océan Indien	1500	Santalba	Kayac	50

Algèbre relationnelle



Le renommage $(\rho_{A \to A', B \to B'})$: renomme l'attribut *A* en *A'* et *B* en *B'*

L'union $(R \cup S)$: crée une relation comprenant tous les tuples existants dans l'une ou l'autre des relation R et *S* (R et *S* doivent avoir le même schéma!).

La différence $(R - S)$: crée une relation comprenant tous les tuples de R qui ne sont pas dans *S* (R et *S* doivent avoir le même schéma!).

Algèbre relationnelle

La jointure (∞) : sélection + produit cartésien

Ex: informations sur les stations et leur activité

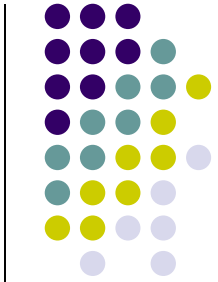
$\sigma_{S.nomStation='A.nomStation'}$ *(Station x Activité)*

Station (∞) *Activité*

S.nomStation	cap.	lieu	région	tarif	A.nomStation	libelle	prix
Venusa	350	Guadeloupe	Antilles	1200	Venusa	Voile	150
Venusa	350	Guadeloupe	Antilles	1200	Venusa	Plongée	120
Farniente	200	Seychelles	Océan Indien	1500	Farniente	Plongée	130
Santalba	150	Martinique	Antilles	2000	Santalba	Kayac	50
Passac	400	Alpes	Europe	1000	Passac	Ski	200
Passac	400	Alpes	Europe	1000	Passac	Piscine	20

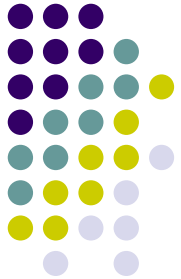
FIG. 5.4 – *Une jointure* $\sigma_{S.nomStation=A.nomStation}(Station \times Activite)$, *composition de* $\times$ *et* σ

Excercices:



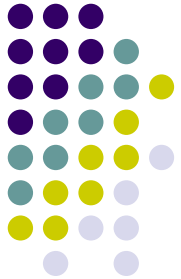
- les stations qui sont aux Antilles *et* dont la capacité est supérieure à 200 :
- les stations qui sont aux Antilles, *ou* dont la capacité est supérieure à 200 :
- les stations dont la capacité est supérieure à 200 mais qui ne sont *pas* aux Antilles :
- Le nom des stations aux Antilles :
- Le nom et prénom des clients européens :

Excercices:



- le nom et la région des stations où l'on pratique la voile:
- le nom des clients qui sont allés à Passac :
- quelles régions a visité le client 30 :
- le nom des clients qui ont eu l'occasion de faire de la voile :
- les noms des clients qui sont partis en vacances dans leur région, ainsi que le nom de cette région :

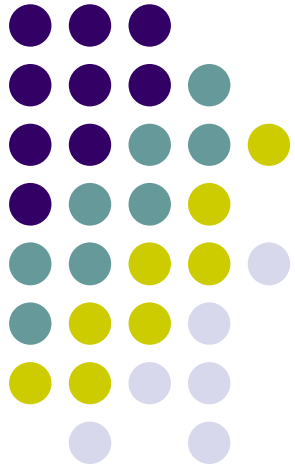
Excercices:



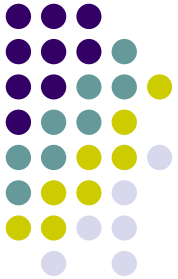
- quelles sont les stations qui *ne* proposent *pas* de voile ?
- le nom des régions où il y a des clients, mais pas de station :
- le nom des stations qui n'ont pas reçu de client américain :
- l'Id des clients qui ne sont pas allés aux Antilles :
- les ids des clients *et* les stations où ils ne sont pas allés.
- quelles sont les stations dont *toutes* les activités ont un prix supérieur à 100 ?
- les ids des clients qui sont allés dans *toutes* les stations :

Bases de Données

- Introduction
- Modèle Entité/Association
- Modèle relationnel
- Algèbre relationnelle
- SQL



SQL



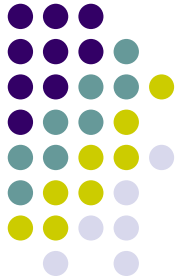
SQL : *Structured Query Language*

- **Langage de Manipulation de Données (DML) :** interroger et modifier les données de la base
- **Langage de Définition de Données (DDL) :** définir le schéma de la base de données
- **Langage de contrôle d'accès aux données**

SQL / types

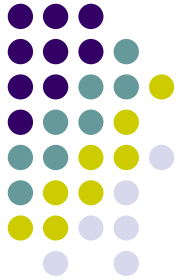
Type	Description	Taille
`INTEGER`	Type des entiers relatifs	4 octets
`SMALLINT`	Idem.	2 octets
`BIGINT`	Idem.	8 octets
`FLOAT`	Flottants simple précision	4 octets
`DOUBLE PRECISION`	Flottants double précision	8 octets
`REAL`	Synonyme de `FLOAT`	4 octets
`NUMERIC` (*M*, *D*)	Numérique avec précision fixe.	*M* octets
`DECIMAL` (*M*, *D*)	Idem.	*M* octets
`CHAR`(*M*)	Chaînes de longueur fixe	*M* octets
`VARCHAR`(*M*)	Chaînes de longueur variable	*L*+1 avec $L \leqslant M$
`BIT VARYING`	Chaînes d'octets	Longueur de la chaîne.
`DATE`	Date (jour, mois, an)	env. 4 octets
`TIME`	Horaire (heure, minutes, secondes)	env. 4 octets
`DATETIME`	Date et heure	8 octets
`YEAR`	Année	2 octets

SQL / création des tables



```
CREATE TABLE Internaute
(email VARCHAR (50),
nom VARCHAR (20),
prenom VARCHAR (20),
motDePasse VARCHAR (60),
anneeNaiss DECIMAL (4))

CREATE TABLE Cinéma
(nom VARCHAR (50),
adresse VARCHAR (50))
```

SQL / création des tables



Contrainte d'intégrité:

- Un attribut peut toujours avoir une valeur.
- Un attribut (groupe d'attributs) constitue la clé
- Un attribut dans une table est lié à la clé primaire d'une autre table
- La valeur d'un attribut doit être unique au sein de la relation
- Enfin tout règle s'appliquant à la valeur d'un attribut.

Synopsis

CREATE TABLE *nom_table*
([
{ *nom_colonne type_donnees*
[DEFAULT *default_expr*] [*contrainte_colonne* [...]]
| *contrainte_table* }])

où *contrainte_colonne* peut être :

[CONSTRAINT *nom_contrainte*]
{ NOT NULL | NULL | UNIQUE | PRIMARY KEY | CHECK (*expression*) |
REFERENCES *table_ref* [(*colonne_reference*)][MATCH FULL |MATCH SIMPLE]
[ON DELETE *action*] [ON UPDATE *action*] }

et *contrainte_table* :

[CONSTRAINT *nom_contrainte*]
{ UNIQUE (*nom_colonne* [, ...]) | PRIMARY KEY (*nom_colonne* [, ...])
CHECK (*expression*) |
FOREIGN KEY (*nom_colonne* [, ...]) REFERENCES *table_reference* [(
colonne_reference [, ...])]
[MATCH FULL | MATCH SIMPLE] [ON DELETE *action*] [ON UPDATE *action*] }

Création des tables: NULL/DEFAULT

```
CREATE TABLE Internaute
(email VARCHAR (50) NOT NULL,
nom VARCHAR (20) NOT NULL,
prenom VARCHAR (20),
motDePasse VARCHAR (60) NOT NULL,
anneeNaiss DECIMAL (4))

CREATE TABLE Cinéma
(nom VARCHAR (50) NOT NULL,
adresse VARCHAR (50) DEFAULT 'Inconnue')
```

Création des tables: PRIMARY KEY

Clé primaire ***(primary key),***

CREATE TABLE Notation

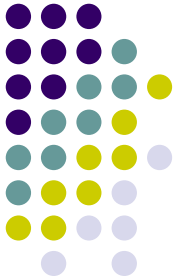
(idFilm INTEGER **NOT NULL**,

email VARCHAR (50) **NOT NULL**,

note INTEGER **DEFAULT** 0,

PRIMARY KEY (idFilm, email))

Création des tables : UNIQUE



***Clé secondaire*(unique)**

```
CREATE TABLE Artiste
(id INTEGER NOT NULL,
nom VARCHAR (30) NOT NULL,
prenom VARCHAR (30) NOT NULL,
anneeNaiss INTEGER,
PRIMARY KEY (id),
UNIQUE  (nom, prenom));
```

Création des tables: UNIQUE

Clé secondaire(unique)

```
CREATE TABLE Artiste

(id INTEGER NOT NULL,

nom VARCHAR (30) NOT NULL,

prenom VARCHAR (30) NOT NULL,

anneeNaiss INTEGER,

PRIMARY KEY (id),

CONSTRAINT unic_nom_prenom UNIQUE (nom, prenom));
```

Création des tables: FOREIGN KEY

Clé étrangère (Foreign key) : les attributs faisant Reference une ligne d'une autre table

```
CREATE TABLE Film
        (idFilm INTEGER NOT NULL,
        titre VARCHAR (50) NOT NULL,
        annee INTEGER NOT NULL,
        idMES INTEGER,
        codePays INTEGER,
        PRIMARY KEY (idFilm),
        FOREIGN KEY (idMES) REFERENCES Artiste (idArtiste),
        FOREIGN KEY (codePays) REFERENCES Pays);
```

Création des tables: FOREIGN KEY

ON DELETE NO ACTION ON UPDATE NO ACTION	Toute action de modification de clef ou de suppression de ligne dans la table mère échoue pour les lignes en relation d'intégrité référentielle entre table mère et fille.	Dans le cas du NO ACTION, il y a blocage de l'ordre SQL UPDATE ou DELETE dans le but de maintenir le lien d'intégrité tel quel. Ce blocage intervient en fin de transaction.

Création des tables: FOREIGN KEY

ON DELETE SET NULL ON UPDATE SET NULL	Toute action de modification ou de suppression dans la table mère est répercutée dans la table fille par la suppression des valeurs des clefs étrangères, si les colonnes en jeu sont *nullables*	Dans le cas du SET NULL, il y suppression des valeurs des clefs étrangères pour les lignes concernées, ce qui conduit à un déréférencement du lien d'intégrité.

Création des tables: FOREIGN KEY

la destruction d'un metteur en scène déclenche la mise à NULL (SET NULL) de la clé étrangère idMES pour tous les films qu'il a réalisé:

```
CREATE TABLE Film
        (titre VARCHAR (50) NOT NULL,
        annee INTEGER NOT NULL,
        idMES INTEGER,
        codePays INTEGER,
        PRIMARY KEY (titre),
        FOREIGN KEY (idMES) REFERENCES Artiste
                ON DELETE SET NULL,
        FOREIGN KEY (codePays) REFERENCES Pays);
```

Création des tables: FOREIGN KEY

ON DELETE SET DEFAULT ON UPDATE SET DEFAULT	Toute action de modification ou de suppression dans la table mère est répercutée dans la table fille par la mise en place des valeurs spécifiées par défaut pour les colonnes des clefs étrangères, si des valeurs par défaut ont bien été spécifiées pour toutes les colonnes en jeu.	Dans le cas du SET DEFAULT, il y mise en place des valeurs par défaut des clefs étrangères pour les lignes concernées, ce qui conduit à un glissement des valeurs de références du lien d'intégrité

Création des tables: FOREIGN KEY

ON DELETE CASCADE ON UPDATE CASCADE	Toute modification de clef ou suppression de ligne dans la table mère est répercutée dans la table fille.	Dans le cas du CASCADE, il y répercussion de l'ordre SQL avec modification de valeur ou suppression de lignes afin de maintenir ou d'évacuer le lien d'intégrité.

Création des tables: FOREIGN KEY

Quand on détruit un cinéma, on veut également détruire les salles ; quand on modifie la clé d'un cinéma, on veut répercuter la modification sur ses salles :

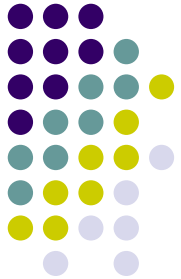
CASCADE appliquer la même opération

```
CREATE TABLE Salle
        (nomCinema VARCHAR (30) NOT NULL,
        no INTEGER NOT NULL,
        capacite INTEGER,
        PRIMAR KEY (nomCinema, no),
        FOREIGN KEY (nomCinema) REFERENCES Cinema
        ON DELETE CASCADE
        ON UPDATE CASCADE)
```

Match

Mode	Règle	Explication
MATCH SIMPLE	si toutes les colonnes contraintes sont renseignées, la contrainte s'applique si une colonne au moins possède un marqueur NULL, la contrainte ne s'applique pas	Il y a violation de contrainte si les valeurs divergent et qu'elles sont toutes renseignées
MATCH PARTIAL	La contrainte s'applique pour toutes les colonnes renseignées	Il y a violation de la contrainte si au moins une valeur renseignée diverge.
MATCH FULL	la contrainte s'applique toujours sauf si toutes les colonnes sont pourvues d'un marqueur NULL	Il y a violation de la contrainte si les valeurs divergent ou si une colonne est renseignée et l'autre pas

Match exemple



- **CREATE TABLE UTILISATUER**
- (
- NOM CHAR(32) **NOT NULL,**
- PRENOM VARCHAR(16) **NOT NULL,**
- **CONSTRAINT PK_USR**
- **PRIMARY KEY(NOM, PRENOM)**
-)

- Supposons que la table T_UTILISATEUR_USR soit remplie avec les lignes suivantes :
- NOM PRENOM
- DUBOIS Alain
- DURAND Paula

Match exemple

MATCH SIMPLE:

```
CREATE TABLE T_NEWS_NEW
(
NEW_ID INTEGER NOT NULL PRIMARY KEY,
NOM CHAR(32),
PRENOM VARCHAR(16),
CONSTRAINT CNT_MATCHSIMPLE
    FOREIGN KEY (NOM,PRENOM)
      REFERENCES UTILISATEUR (NOM, PRENOM)
                MATCH SIMPLE
)
```

Match exemple

MATCH FULL:

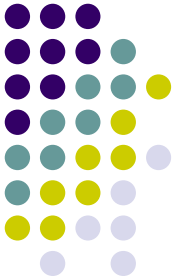
```
CREATE TABLE T_NEWS_NEW
(
NEW_ID INTEGER NOT NULL PRIMARY KEY,
NOM CHAR(32),
PRENOM VARCHAR(16),
CONSTRAINT CNT_MATCHSIMPLE
    FOREIGN KEY (NOM,PRENOM)
      REFERENCES UTILISATEUR (NOM, PRENOM)
                MATCH FULL
)
```

Match exemple

MATCH PARTIAL:

```
CREATE TABLE T_NEWS_NEW
(
NEW_ID INTEGER NOT NULL PRIMARY KEY,
NOM CHAR(32),
PRENOM VARCHAR(16),
CONSTRAINT CNT_MATCHSIMPLE
    FOREIGN KEY (NOM,PRENOM)
      REFERENCES UTILISATEUR (NOM, PRENOM)
                MATCH PARTIAL
)
```

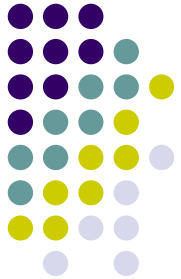
Création des tables: CHECK



Enumération des valeurs possibles avec CHECK

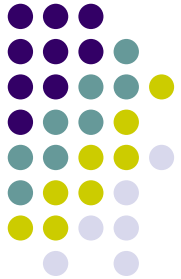
```
CREATE TABLE Film
    (titre VARCHAR (50) NOT NULL,
    annee INTEGER
          CHECK (annee BETWEEN 1890 AND 2000) NOT NULL,
    genre VARCHAR (10)
          CHECK (genre IN ('Histoire','Western','Drame')),
    idMES INTEGER,
    codePays INTEGER,
    PRIMARY KEY (titre),
    FOREIGN KEY (idMES) REFERENCES Artiste (idArtiste),
    FOREIGN KEY (codePays) REFERENCES Pays);
```

Exemple



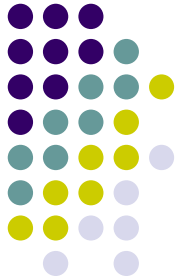
```
CREATE TABLE Patient
(PTN_ID INT NOT NULL,
PTN_NUM_SECU CHAR(13),
PTN_CLEF_SECU CHAR(2),
PTN_NOM CHAR(32) NOT NULL,
PTN_PRENOM VARCHAR(25),
PTN_DATE_NAIS DATE,
PTN_CIVILITE INT,
CONSTRAINT cle_primaire PRIMARY KEY (PTN_ID ),
CONSTRAINT nosecu
   UNIQUE(PTN_NUM_SECU,PTN_CLEF_SECU),
CONSTRAINT construction_clef CHECK
   (PTN_CLEF_SECU IS NULL)
   OR
   ( (substring(PTN_CLEF_SECU from 1 to 1) BETWEEN 0 AND 9)
      AND
      (substring(PTN_CLEF_SECU from 2 to 2) BETWEEN 0 AND 9)
   )
CONSTRAINT cle_Etrangere FOREIGN KEY (PTN_CIVILITE )
      REFERENCES Civilite (CVT_ID ));
```

Exemple



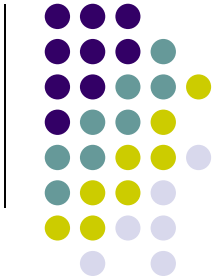
```
CREATE TABLE Facture
(
FCT_ID INT NOT NULL PRIMARY KEY,
FCT_DATE_EMMISSION DATE NOT NULL DEFAULT
   CURRENT_DATE,
FCT_DATE_PAIEMENT DATE CHECK
      (
   FCT_DATE_PAIEMENT IS NULL
   OR
   FCT_DATE_PAIEMENT >=FCT_DATE_EMMISSION
      ),
);
```

Types de données



- Types de données numériques
- Types caractères
- Types date/heure
- Types boolean

Types de données numériques

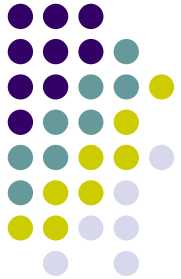


Nom	Taille de stockage	Description	Étendue
smallint	2 octets	entier de faible étendue	-32768 à +32767
integer	4 octets	entiers les plus courants	-2147483648 à +2147483647
bigint	8 octets	grands entiers	-9223372036854775808 à 9223372036854775807
decimal	variable	Précision indiquée par l'utilisateur. Valeurs exactes	pas de limite
numeric	variable	Précision indiquée par l'utilisateur. Valeurs exactes	pas de limite
real	4 octets	Précision variable. Valeurs inexactes	précision de 6 décimales
double precision	8 octets	Précision variable. Valeurs inexactes	précision de 15 décimales
serial	4 octets	entier à incrémentation automatique	1 à 2147483647
bigserial	8 octets	entier de grande taille à incrémentation automatique	1 à 9223372036854775807

Types de données numériques

- **`Integer, smallint, bigint`**: entiers
- **`Numeric`** = **`decimal`**
  - **`Numeric(precision, echelle)`**
    - Ex: 435,5678 precision:7 et echelle: 4
  - **`Numeric(precision)`** : echelle est zéro
  - **`Numeric`** : stockage des valeurs de n'importe quelle précision ou echelle (limite de précision!)
  - Valeur spéciale **`NaN`** (not-a-number)
  - Ex: **`UPDATE`** table `SET` x='**`NaN`**'

Types de données numériques

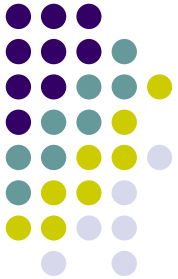


- **`real, double precision`**: précision à variable inexacte
- Attention : stockage+réaffichage, arrondis pour les calcules, très grandes ou petites valeurs, arrondis à zéro, égalité de deux reals, etc.!
- **`real`**: étendue d'au moins 1E-37 à 1E37
- **`double precision`**: étendue de 1E-307 à 1E+308
- Valeurs spéciales: '**`infinity`**', '**`-infinity`**', '**`NaN`**'

Types de données numériques

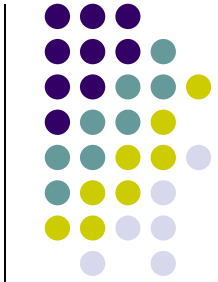
- **`serial, bigserial`** : pas de vrai types, ce sont des entiers dont les valeurs sont assignés par un générateur de séquence.
- Attention: si on veut pas de doublons: préciser **`UNIQUE`**
- Insertion: soit négliger cette colonne dans **`INSERT`** soit en utilisant le mot clé **`DEFAULT`**!

Types caractères



Nom	Description
`character varying(n), varchar(n)`	Longueur variable avec limite
`character(n), char(n)`	longueur fixe, comblé avec des espaces
`text`	longueur variable illimitée

Types caractères



- **`character(n)`** = chaîne de taille < n est complétée par des espaces
- **`varchar(n)`** = chaîne de taille <n stockée telle qu'elle est.
- Insérer une chaîne avec + de n caractères: erreure sauf espaces
- pour transtyper (cast) si la taille de la chaîne >n, la chaîne est tronquée a n caractères.
- **`character`** = **`character(1)`**
- **`varchar`** = chaîne de toutes tailles
- **`text`** = chaîne de toutes tailles(pour tres longues chaîne)

Types date/heure

Nom	Taille de stockage	Description	Valeur minimale	Valeur maximale	Résolution
`timestamp[(p)] [without time zone]`	8 octets	date et heure	4713 avant JC	5874897 après JC	1 microseconde / 14 chiffres
`timestamp[(p)] with time zone`	8 octets	date et heure, avec fuseau horaire	4713 avant JC	5874897 après JC	1 microseconde / 14 chiffres
`interval[(p)]`	12 octets	intervals de temps	-178000000 années	178000000 années	1 microseconde / 14 chiffres
`date`	4 octets	dates seulement	4713 avant JC	5874897 après JC	1 jour
`time[(p)] [without time zone]`	8 octets	heures seulement	00:00:00.00	23:59:59.99	1 microseconde / 14 chiffres
`time[(p)] with time zone`	12 octets	heures seulement, avec fuseau horaire	00:00:00.00+12	23:59:59.99-12	1 microseconde / 14 chiffres

Types date

Exemple	Description
January 8, 1999	sans ambiguïté quel que soit le style de date (`datestyle`)
1999-01-08	ISO-8601; 8 janvier, quel que soit le mode (format recommandé)
1/8/1999	8 janvier en mode `MDY`; 1er Août en mode `DMY`
1/18/1999	18 janvier en mode `MDY`; rejeté dans les autres modes
01/02/03	2 janvier 2003 en mode `MDY`; 1er février 2003 en mode `DMY`; 3 février 2003 en mode `YMD`
1999-Jan-08	8 janvier dans tous les modes
Jan-08-1999	8 janvier dans tous les modes
08-Jan-1999	8 janvier dans tous les modes
99-Jan-08	8 janvier en mode `YMD`, erreur sinon
08-Jan-99	8 janvier, sauf en mode `YMD`: erreur
Jan-08-99	8 janvier, sauf en mode `YMD`: erreur

Types date

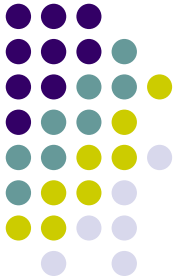
Exemple	Description
04:05:06.789	ISO 8601
04:05:06	ISO 8601
04:05	ISO 8601
040506	ISO 8601
04:05 AM	Identique à 04:05; AM n'affecte pas la valeur
04:05 PM	Identique à 16:05; l'heure doit être <= 12
04:05:06 PST	fuseau horaire précisé en lettres

Types intervalle

quantité unité [*quantité unité*...] [*direction*]

- *quantité* est un nombre ; *unité* est second, minute, hour, day, week, month, year, decade, century, millennium ;
- *direction* peut être ago ou vide
- Exemple: '1 day 12:59:10' est lu comme '1 day 12 hours 59 min 10 sec‘.

Type boolean

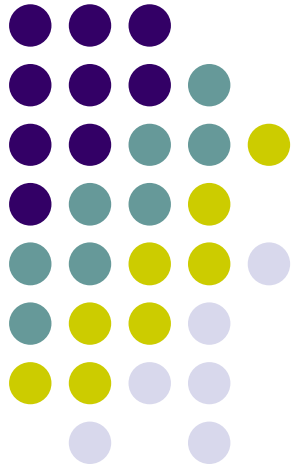


- << vrai >>: TRUE, 't', 'true', 'y', 'yes', '1'.
- << faux >>: FALSE, 'f', 'false', 'n', 'no', '0'.
- Exemple:

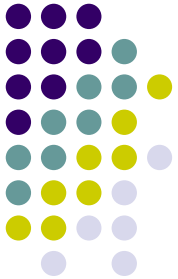
```
CREATE TABLE test1 (a boolean, b text);

INSERT INTO test1 VALUES (TRUE, 'sic est');
INSERT INTO test1 VALUES (FALSE, 'non est');

SELECT * FROM test1;

SELECT * FROM test1 WHERE a;
```

Bases de Données

- Introduction
- Modèle Entité/Association
- Modèle relationnel
- Algèbre relationnelle
- SQL
  - Modification de tables



Création de table

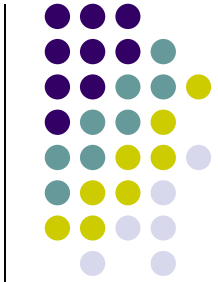


- **CREATE TABLE AS** requête:

  Exemple: Créer une table **films_recent** contenant les entrées récentes de la table **films** :

```
CREATE TABLE films_recent AS
    SELECT *
    FROM films
    WHERE date_prod >= '2006-01-01';
```

Modification des tables



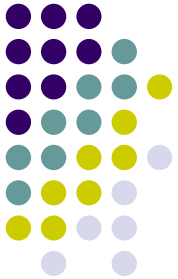
- **DROP TABLE** nom_table : efface les données et la structure de la table
- **TRUNCATE TABLE** nom_table: n'efface que les donnees
- **RENAME TABLE** ancien_nom TO nouveau-nom
- **ALTER TABLE** ancien_nom **TO** nouveau_nom

Modification des tables: Alter

ALTER TABLE table_name

```
{
ALTER COLUMN column_name {DROP DEFAULT | SET DEFAULT
constant_expression }
|
ADD { < column_definition > | < table_constraint > } [ ,...n ]
|
DROP { [ CONSTRAINT ] constraint_name | COLUMN column }] }
```

Modification de table



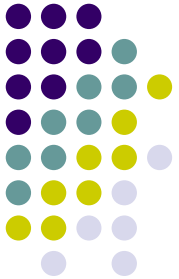
ALTER TABLE nom_table **ADD** [**CONSTRAINT**] nom_contrainte contrainte

- Exemple:

ALTER TABLE Client **ADD**
CONSTRAINT c1 **CHECK** datenais>1980

ALTER TABLE Client **ADD**
CONSTRAINT c2
FOREING KEY (adresse) **REFERENCES** Adresses(adresse)

Modification de table



ALTER TABLE nom_table **DROP CONSTRAINT** nom_contrainte

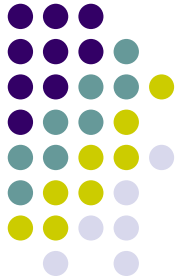
- Exemple:

ALTER TABLE Client **DROP CONSTRAINT** c1

Attention!

Avec postgreSQL seulement les contraintes **CHECK** peuvent être supprimées!

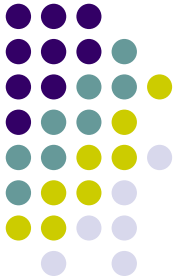
Modification de table



- Ex: si on veut effacer toutes les contraintes de la table Client:

```
CREATE TABLE temp AS SELECT * FROM Client;

DROP TABLE Client;

CREATE TABLE Client AS SELECT * FROM temp;

DROP TABLE temp;
```

Modification des tables

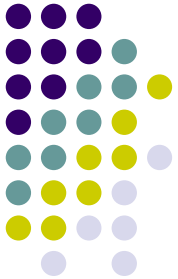


`ALTER TABLE` nom_table **`DROP`** [**`COLUMN`**] nom_colonne.

Si ce n'est pas supporté: la table doit être recréée et rechargée!

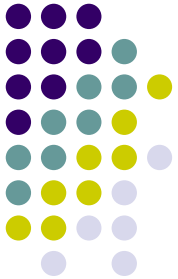
Comment supprimer l'avis_sur_prix de la table Vente (id_bar, biere, prix, avis_sur_prix)?

Modification de tables



```
CREATE TABLE temp AS  SELECT id_bar, biere, prix FROM Vente

DROP table Vente

CREATE TABLE Vente
(id_bar Integer NOT NULL, biere Varchar(15) NOT NULL,
prix decimal(5,2) ,
PRIMARY KEY (id_bar, biere)
FOREIGN KEY(id_bar) REFERENCES Bar
FOREIGN KEY (biere) REFRENCES Biere);

INSERT INTO Vente SELECT * FROM  temp;

DROP TABLE temp;
```

Modification des tables

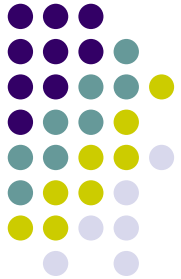


ALTER TABLE nom_table **ADD** [**COLUMN**]
nom_colonne type_colonne colonne_contrainte

- Exemple:

```
ALTER TABLE Client ADD [COLUMN]
   metier VARCHAR(50),
   lieu_nais VARCHAR(30)
      FOREIGN KEY(lieu_nais)
      REFERENCES Ville(nom_ville)
      ON DELETE SET NULL
      ON UPDATE CASCADE
```

Modification des tables

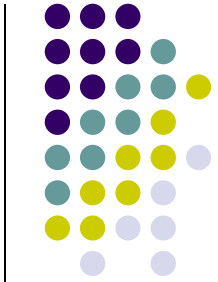


Attention!

avec postgreSQL, la forma **`ADD COLUMN`** ne supporte pas les valeurs par defaut et les contraintes **`NOT NULL`**

Par contre la forma **`ALTER COLUMN`** est pleinement conforme!

Modification des tables

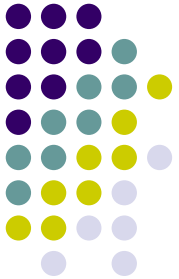


ALTER TABLE nom_table **ALTER** [**COLUMN**]
nom_colonne [**SET DEFAULT** value|**DROP DEFAULT**]

Exemple:
ALTER TABLE Client **ALTER** Adresse
SET DEFAULT 'inconnu'

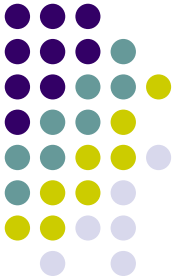
Attention!
Les defauts s'appliquent seulement aux commandes **INSERT** subséquentes.

Modification des tables



ALTER TABLE Client **RENAME** [**COLUMN**]
Date_naissance **TO** Annee_naissance;

SQL / création des tables



Insertion des données avec INSERT INTO

```
CREATE TABLE Pays
        (code VARCHAR (4) DEFAULT 0 NOT NULL,
        nom VARCHAR (30) NOT NULL,
        langue VARCHAR (30) NOT NULL,
        PRIMARY KEY (code))

    INSERT INTO Pays VALUES (0, 'Inconnu', 'Inconnue');
    INSERT INTO Pays VALUES (1, 'France', 'Français');
    INSERT INTO Pays VALUES (2, 'USA', 'Anglais');
    INSERT INTO Pays VALUES (3, 'Allemagne', 'Allemand');
    INSERT INTO Pays VALUES (4, 'Angleterre', 'Anglais');
```

SQL / Modification du schéma

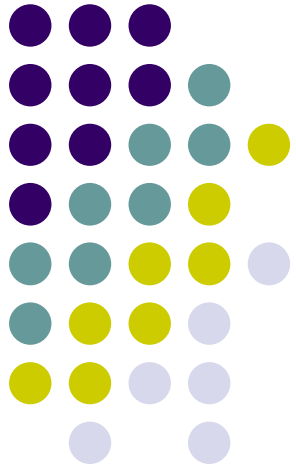
ALTER TABLE *nomTable ACTION description*

Action : ADD, MODIFY, DROP, RENAME

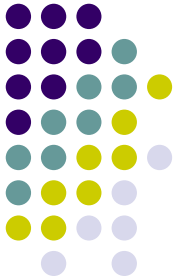
- ALTER TABLE Internaute *ADD* region VARCHAR(10);
- ALTER TABLE Internaute *MODIFY* region VARCHAR(30) NOT NULL;
- ALTER TABLE Internaute *ALTER* region *SET DEFAULT* ’PACA’;
- ALTER TABLE Internaute *DROP* region;

Bases de Données

- Introduction
- Modèle Entité/Association
- Modèle relationnel
- Algèbre relationnelle
- SQL
  - Requêtes



SQL / création d'index



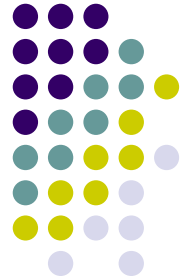
Index pour les clés primaires (systématique),les clés secondaires (UNIQUE) ou les attributs normaux:

CREATE [UNIQUE] INDEX *nomIndex* ON *nomTable (attribut1 [, ...])*

Ex :
CREATE UNIQUE INDEX idxNom ON Artiste (nom, prenom);

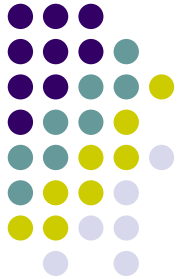
CREATE INDEX idxGenre ON Film (genre);

Exemple:organisme de voyage



- Station (**nomStation**, capacité, lieu, région, tarif)
- Activite (***nomStation***, **libellé**, prix)
- Client (**id**, nom, prénom, ville, région, solde)
- Séjour (***idClient***, **station**, **début**, nbPlaces)

SQL / Requêtes



```
SELECT nomStation
FROM Station
WHERE region = 'Antilles'
```

- FROM indique la (ou les) tables dans lesquelles on trouve les attributs utiles à la requête.
- SELECT indique la liste des attributs constituant le résultat.
- WHERE indique les conditions que doivent satisfaire les n-uplets de la base pour faire partie du résultat.

SQL / Requêtes

SELECT libelle, prix / 6.56, 'Cours de l'euro = ', 6.56
FROM Activite
WHERE nomStation = 'Santalba'

libelle	prix / 6.56	'Cours de l'euro ='	'6.56'
Kayac	7.62	'Cours de l'euro ='	6.56

SELECT libelle, prix / 6.56 **AS** prixEnEuros,
'Cours de l'euro = ', 6.56 **AS** cours
FROM Activite
WHERE nomStation = 'Santalba'

libelle	prixEnEuros	'Cours de l'euro ='	cours
Kayac	7.69	'Cours de l'euro ='	6.56

SQL / Requêtes

- Doublons :

SELECT libelle
FROM Activite

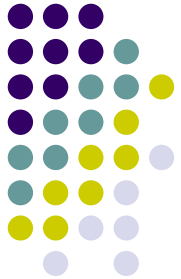
libelle
Voile
Plongee
Plongee
Ski
Piscine
Kayac

SELECT **DISTINCT** libelle
FROM Activite

- Tri du résultat (ascendant; pour descendant ajout DESC)

SELECT *
FROM Station
ORDER BY tarif, nomStation

SQL / Requêtes



- Where :

AND, OR, NOT, <, <=, >, >=, <>, !=,BETWEEN

SELECT nomStation, libelle
FROM Activite
WHERE nomStation = ’Santalba’
AND prix **BETWEEN** 50 AND 120

SQL / Requêtes

- Chaîne de caractères:
  - LIKE : pattern matching
  - ‘_’ : n’importe quel caractère
  - ‘%’ : n’importe quelle chaîne de caractères

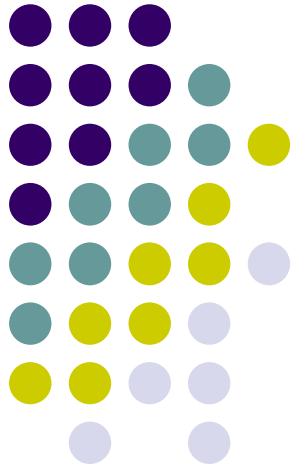
Ex:

```
SELECT nomStation
FROM Station
WHERE nomStation LIKE ’%a’

(se termine par ‘a’)
```

```
SELECT nomStation
FROM Station
WHERE nomStation LIKE ’V_____’

(commence par ‘v’ et a 6 caractères)
```

Bases de Données

- Introduction
- Modèle Entité/Association
- Modèle relationnel
- Algèbre relationnelle
- PL/SQL



PL/SQL(procedural language)

- SQL est un langage non procédural.
- Les traitements complexes sont parfois difficiles à écrire si on ne peut utiliser des variables et les structures de programmation comme les boucles et les alternatives.
- On ressent vite le besoin d'un langage procédural pour lier plusieurs requêtes SQL avec des variables et dans les structures de programmation habituelles.

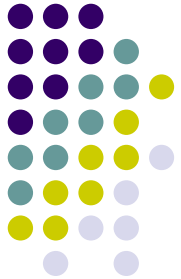
Caractéristiques de PL/SQL

- Extension de SQL : des requêtes SQL cohabitent avec les structures de contrôle habituelles de la programmation structurée (blocs, alternatives, boucles)

- Un programme est constitué de procédures et de fonctions.

- Des variables permettent l'échange d'information entre les requêtes SQL et le reste du programme

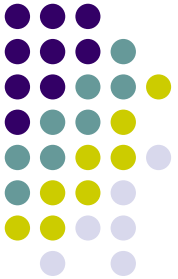
PL/SQL



- PL/SQL est un langage propriétaire de Oracle

- PostgreSQL utilise un langage très proche: PL/pgSQL

Structure d'un bloc PL/SQL



[**DECLARE**

-- déclaration de types, constantes et variables]

BEGIN

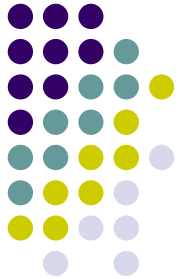
-- instruction PL/SQL

[**EXCEPTION**

-- traitement des erreurs]

END;

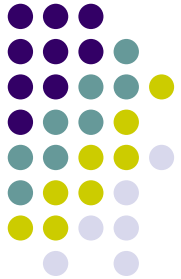
Variables



- Les variables doivent être déclarées avant d'être utilisées

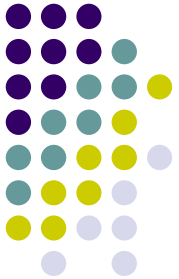
- Identificateur Oracle:
  - 30 caractères au plus
  - Commence par une lettre
  - Peut contenir des chiffres, lettres, _, $ et #

Commentaires



- -- pour une fin de ligne
- /* pour plusieurs lignes */

Déclaration



nomvar
[CONSTANT]
type
[NOT NULL]
[:= valeur | DEFAULT expression]

Déclaration

Types habituels: **`integer, varchar, date, ...`**

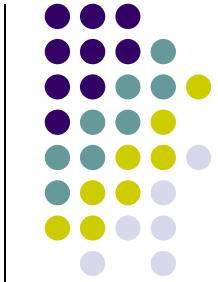
- ***Variables***
  Ex: date_naissance DATE;

  compteur INTEGER := 0; -- initialisation

  compteur INTEGER DEFAULT 0; --initialisation

  id CHAR(5) NOT NULL :=‘AP001’

Déclaration

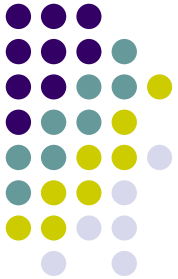


Types habituels: **`integer, varchar, date, ...`**

- ***Constantes***
  Ex: euro CONSTANT REAL := 6.5597

Attention: déclaration multiple interdite!

Déclaration



- Type d'une autre variable : %TYPE

  Ex: credit REAL;

  debit credit%TYPE;

- Type d'un attribut d'une table: %TYPE

  Ex: num_emp EMP.EMPNO%TYPE

- Type d'un n-uplet d'une table: %ROWTYPE

  Ex: un_client client%ROWTYPE

Collection de type RECORD

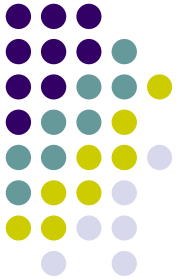
Variable de type 'enregistrement' ou 'ligne' d'une table -> un type structuré décomposable en colonnes élémentaires.

TYPE type_enr **IS RECORD**
(liste_de_types_avec_%TYPE_ou_non |
nom_table%ROWTYPE)

- Ex: **DECLARE**
  TYPE t_nouv_emp **IS RECORD**
  (nom emp.ename%TYPE,
  salaire emp.sal%TYPE,
  comission emp.comm%TYPE);

  rec_emp t_rec_emp;

Affectation



- Affectation simple:

  ex: numero:=0;

  numero:=numero+1;

- Valeurs issues d'une base de données

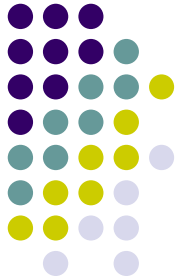
  ex: SELECT numcli INTO numero

  FROM Client WHERE numcli=10;

  SELECT empno, ename INTO num, nom

  FROM emp WHERE ename='King';

Exemple d'utilisation

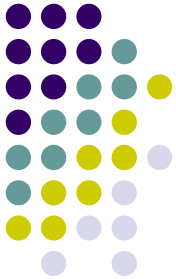


```
employe emp%ROWTYPE;
nom emp.nome.%TYPE;

select * INTO employe
from emp
where matr = 900;

nom := employe.nome;
employe.dept := 20;
…

insert into emp
values employe;
```

Expressions et comparaisons

- Opérateurs arithmétiques: + - / *
- Opérateurs de concaténation: | |
- Opérateurs de comparaison:
  - = < > <= >= <>
  - IS NULL, LIKE, BETWEEN, IN
- Opérateurs logiques:
  - AND, OR, NOT

Structures de contrôle



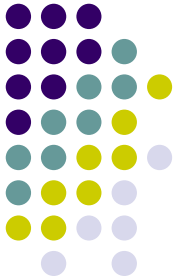
Sélection

```
IF condition THEN
        instruction
END IF;
```

```
IF condition THEN
        instruction1
ELSE
        instruction2;
END IF;
```

```
IF condition1 THEN
        instruction1
ELSEIF condition2 THEN
        instruction2
ELSEIF …
…
ELSE
        instructionN;
END IF;
```

Structures de contrôle

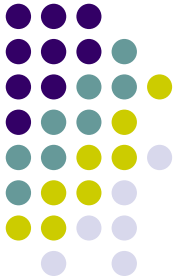


Choix

```
CASE expression
    WHEN exp1 THEN instruction1
    WHEN exp2 THEN instruction2
    …
    ELSE instructionN;
END CASE;
```

Expression peut avoir n'importe quel type simple (ne peut pas par exemple être un RECORD)

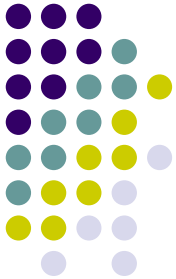
Structures de contrôle



Boucle ‘tant que’

```
WHILE condition LOOP
        instructions;
END LOOP;
```

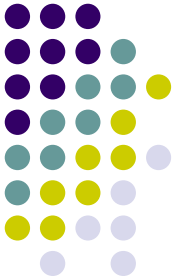
Structures de contrôle



Boucle générale

```
LOOP
        instructions;
        EXIT [WHEN condition];
        instructions;
END LOOP;
```

Structures de contrôle



Boucle ‘pour’

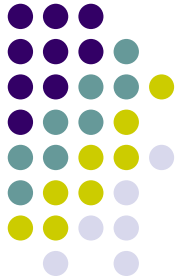
```
FOR compteur IN [REVERSE] inf .. Sup LOOP
    instructions;
END LOOP;
```

Ex: **`for`** i **`IN`** 1..100 **`LOOP`**

somme:=somme+i;

`END LOOP`

Affichage
(paquetage DBMS_OUTPUT)

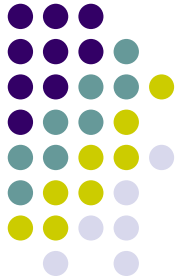


DBMS_OUTPUT.PUT_LINE(‘chaîne’)

- La taille maximum du tampon est de un million de caractères
- La taille maximum d'une ligne est de 255 caractères
- Ex:

```
DBMS_OUTPUT.PUT_LINE(‘bonjour’)
DBMS_OUTPUT.PUT_LINE(‘nom=’||nom)
DBMS_OUTPUT.PUT_LINE(‘num=’TO_CHAR(num));
```

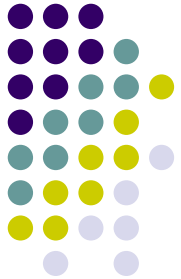
Fonctions



`CREATE [ OR REPLACE ] FUNCTION`
nom ([[modearg] [nomarg] typearg [, ...]])
[**`RETURNS`** type_ret]
{**`AS`** ‘definition’ | **`LANGUAGE`** nomlang|…}

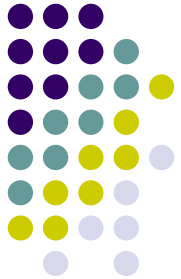
Modearg: IN, OUT, INOUT
Nomarg: nom d’un argument
Typearg: type d’un argument
Typeret: type de retour
Definition: une constante de type chaîne définissant la fonction
Nomlang: langage d’écriture de la fonction

Fonctions



```
CREATE FUNCTION un() RETURNS integer AS $$
    SELECT 1 AS resultat;
$$ LANGUAGE SQL;

-- Autre syntaxe pour les chaînes littérales :
CREATE FUNCTION un() RETURNS integer AS '
SELECT 1 AS resultat;
' LANGUAGE SQL;

SELECT un();
```

Fonctions: +ieurs entrées



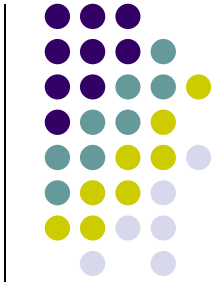
```
CREATE FUNCTION ajoute(integer, integer)
RETURNS integer AS $$
    SELECT $1 + $2;
$$ LANGUAGE SQL;
```

$1: premier argument de la fonction

$2: deuxième argument de la fonction

```
SELECT ajoute(1, 2) AS reponse;
```

Fonctions: void



Fonction qui réalise des actions mais n ’a pas de valeur utile à renvoyer: void!

Ex: **CREATE FUNCTION** nettoie_emp()

RETURNS void AS ‘

DELETE FROM emp

WHERE salaire<0;

‘ **LANGUAGE SQL**;

SELECT nettoie_emp();

Fonctions: update etc.

```
CREATE FUNCTION tf1 (integer, numeric)
RETURNS  numeric AS $$
    UPDATE banque
    SET balance = balance - $2
    WHERE no_compte = $1;

    SELECT 1;
$$ LANGUAGE SQL;
```

- Un utilisateur pourrait exécuter cette fonction pour débiter le compte 17 de 100 000 euros ainsi : SELECT tf1(17, 100.000);

Fonctions: update etc.

```
CREATE FUNCTION tf1 (integer, numeric)
RETURNS  numeric AS $$
    UPDATE banque
    SET balance = balance - $2
    WHERE no_compte = $1;

    SELECT balance FROM banque
    WHERE no_compte = $1;
$$ LANGUAGE SQL;
```

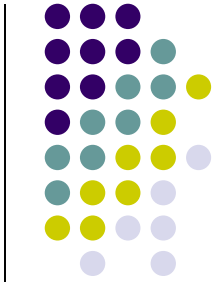
- Un utilisateur pourrait exécuter cette fonction pour débiter le compte 17 de 100 000 euros ainsi : SELECT tf1(17, 100.000);

Fonctions: entrée composite

```
CREATE TABLE emp (
nom text,
salaire numeric,
age integer,
tel varchar(12)
);

CREATE FUNCTION double_salaire(emp) RETURNS numeric AS $$
    SELECT $1.salaire * 2 AS salaire;
$$ LANGUAGE SQL;

    SELECT nom, double_salaire(emp.*) AS reve
    FROM emp
    WHERE emp.tel= '0645342311';
```

emp.*: la ligne courante entière de la table emp

Fonctions: ROW

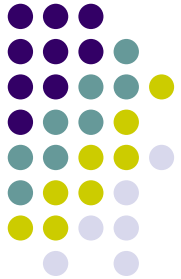


Quelque fois, il est pratique de construire une valeur d'argument composite en direct. Ceci peut se faire avec la construction ROW.

Par exemple, nous pouvons ajuster les données passées à la fonction:

```
SELECT nom,
       double_salaire(ROW(nom, salaire*1.1, age, tel))
       AS reve
FROM emp;
```

Fonctions: renvoie composite



```
CREATE FUNCTION nouvel_emp()
RETURNS  emp AS $$
        SELECT text 'Aucun' AS nom,
                    1000.0 AS salaire,
                    25 AS age,
                   ‘0654321245’ AS Tel;
$$ LANGUAGE SQL;
```

Attention: ordre des colonnes!

As n’a pas d ’interaction.

Fonctions: renvoie composite

```
CREATE FUNCTION nouveau_emp()
RETURNS  emp AS $$
        SELECT
        ROW('Aucun ', 1000.0, 25, '0654321245' )::emp;
$$ LANGUAGE SQL;
```

Attention:

SELECT nouveau_emp(); renvoie une colonne!

SELECT * FROM nouveau_emp(); renvoie 4 colonnes!

Fonctions: renvoie composite

```
CREATE FUNCTION nouveau_emp()
RETURNS  emp AS $$
        SELECT
        ROW('Aucun ', 1000.0, 25, '0654321245' )::emp;
$$ LANGUAGE SQL;
```

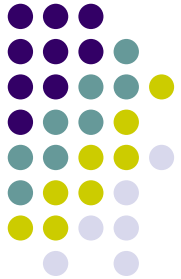
Attention:

SELECT (nouveau_emp()).nom; ou

SELECT nom(nouveau_emp());

renvoie seulement le nom

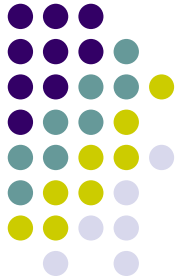
Fonctions: paramètres en sortie



```
CREATE FUNCTION ajoute (IN x int, IN y int,
OUT sum int) AS '
        SELECT $1 + $2'
LANGUAGE SQL;

SELECT ajoute(3,7);
```

resultat: ajoute(): 10

Fonctions:
sortie +ieurs colonnes

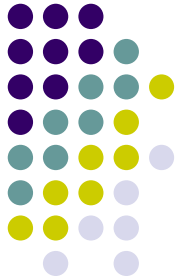


```
CREATE FUNCTION ajoute_et_produit (IN x
  int, IN y int, OUT sum int, OUT product int)
  AS '
      SELECT $1 + $2, $1*$2
' LANGUAGE SQL;
```

SELECT ajoute_et_produit(3,7);

resultat: sum 10 product 21

Création de type



CREATE TYPE produit_ajout **AS** (somme int, produit int)

CREATE FUNCTION ajoute_n_produit (int, int)
RETURNS produit_ajout **AS** '
    SELECT $1 + $2, $1*$2
'**LANGUAGE** SQL;

Renvoyer un ensemble

```
CREATE FUNCTION recup_client (varchar(10))
RETURNS SETOF Client AS $$
SELECT * FROM Client WHERE
   nom_client=$1
$$ LANGUAGE SQL;
```