# Sources de Données

## Ressource R207

IUT R&T Villetaneuse

5 janvier 2022

# Organisation du module

| | |
|---|---|
| Cours | 2 * 1h |
| TD-TP | 2 * (1h30 + 1h30) |
| TP | 3 * 3h |
| Contrôle | de 2h |

`https://lipn.univ-paris13.fr/~petrucci/R207`

# Stockage et accès aux données

# Modèles de stockage de données

- Bases de données relationnelle : tables
- Bases de données NoSQL : associations (clef, valeur)
- Fichiers

# Modèles de stockage de données

- Bases de données relationnelle : tables
- Bases de données NoSQL : associations (clef, valeur)
- Fichiers

## Réglementation

- CNIL
- RGPD

# CSV

```
type,constructeur,capacité
A340,Airbus,228
B747,Boeing,432
```

# CSV

```
type,constructeur,capacité
A340,Airbus,228
B747,Boeing,432
```

| type | constructeur | capacité |
|---|---|---|
| A340 | Airbus | 228 |
| B747 | Boeing | 432 |

# JSON

```
{
   "menu": {
     "id": "menu_file",
     "value": "File",
     "popup": {
       "menuitem": [
         {"value":"New","onclick":"CreateNewDoc()
             "},
         {"value":"Open","onclick":"OpenDoc()"},
         {"value":"Close","onclick":"CloseDoc()"}
       ]
     }
   }
}
```

# Conception d'une base de données

# Conception de BD

## niveau externe

rédaction en français des besoins (cahier des charges)

# Conception de BD

## niveau externe

rédaction en français des besoins (cahier des charges)

## niveau conceptuel

diagramme de classes UML

# Conception de BD

## niveau externe

rédaction en français des besoins (cahier des charges)

## niveau conceptuel

diagramme de classes UML

## niveau logique

schéma relationnel

# Conception de BD

## niveau externe

rédaction en français des besoins (cahier des charges)

## niveau conceptuel

diagramme de classes UML

## niveau logique

schéma relationnel

## niveau physique

scripts SQL

# Exemple

## Cahier des charges

On souhaite développer une application informatique de gestion de vols. Cette application doit permettre de gérer l'affectation des pilotes aux différents vols.

# Exemple

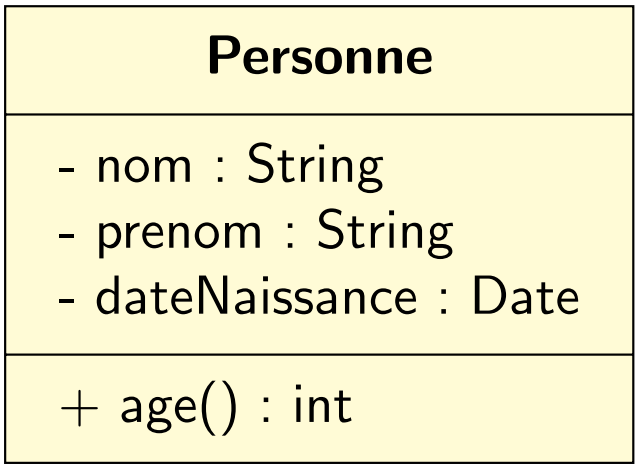
## Cahier des charges

On souhaite développer une application informatique de gestion de vols. Cette application doit permettre de gérer l'affectation des pilotes aux différents vols.

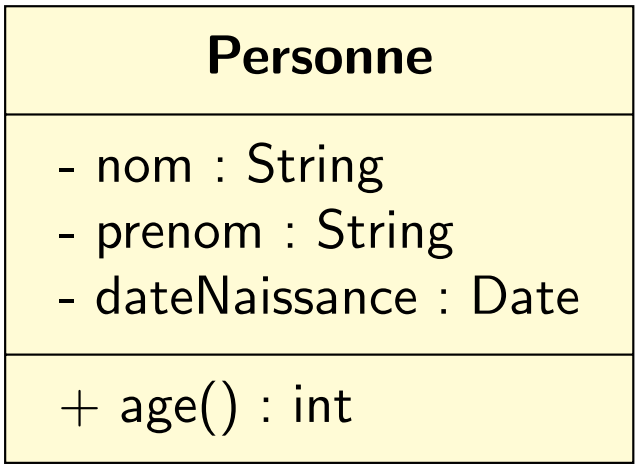
## Contraintes

- Un pilote ne peut pas assurer deux vols en même temps.
- Lors d'un vol, il doit y avoir un pilote dans l'avion.

# Notion de classe

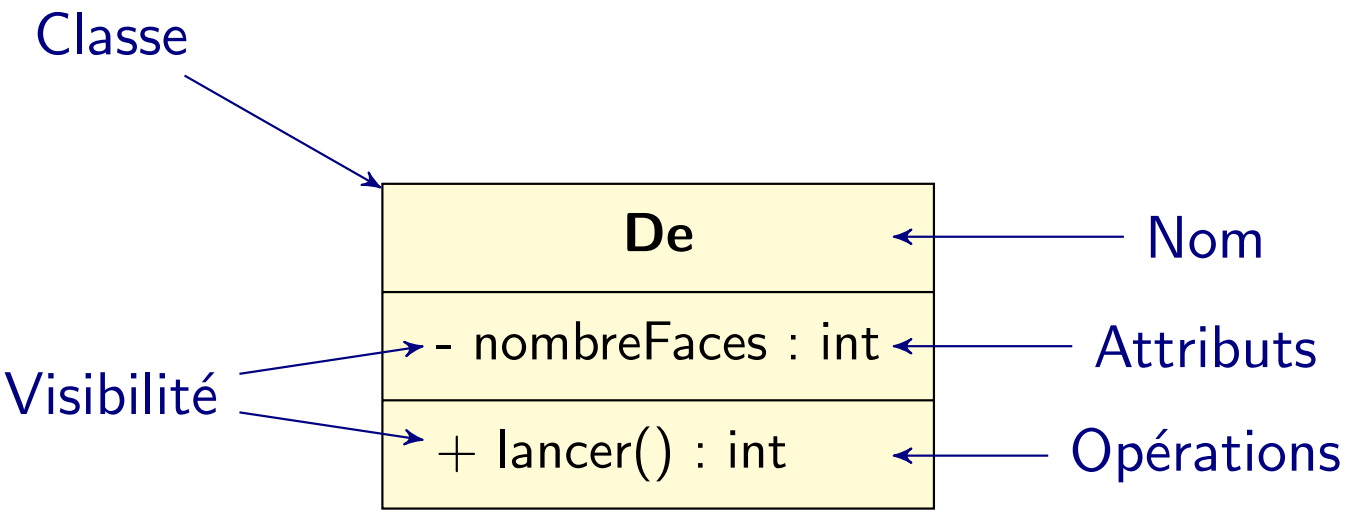


# Notion de classe

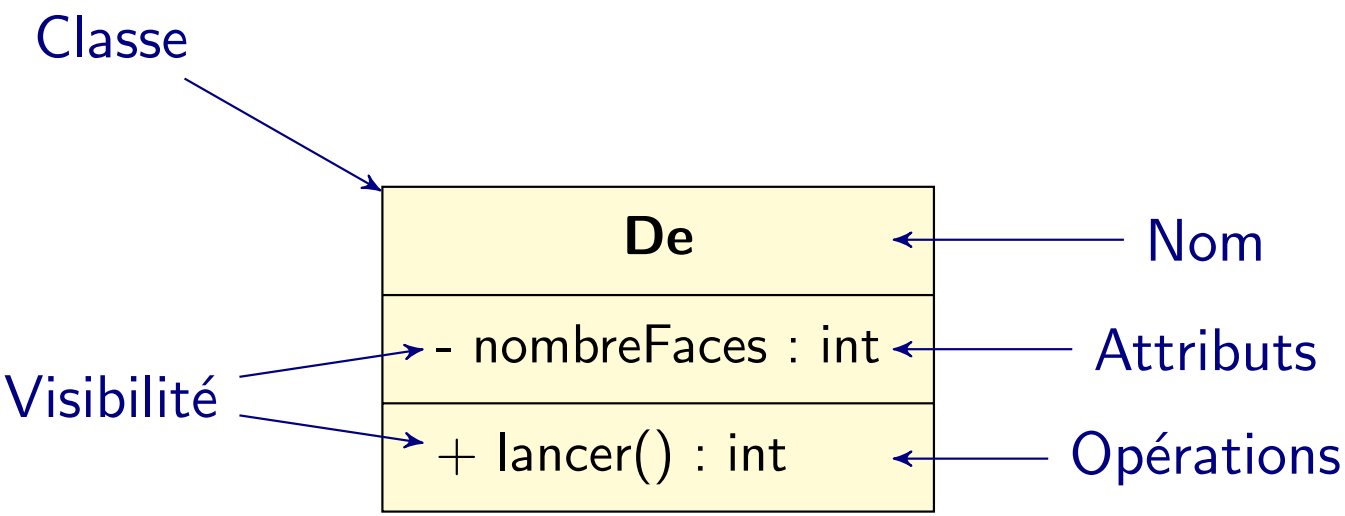



- Une classe est la description formelle d'un ensemble d'objets ayant des propriétés (attributs et méthodes) communes
- Une classe peut être instanciée : une instance d'une classe est un objet

# Représentation graphique



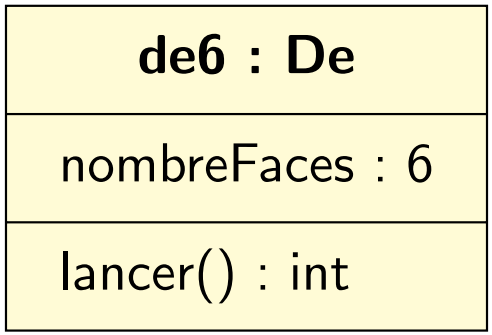
# Représentation graphique



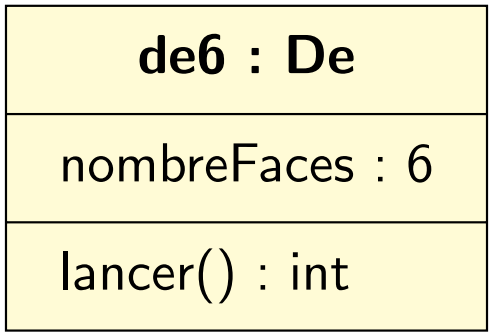

## Attributs

- Les attributs définissent la structure d'un objet : ils répondent à la question : *Qui suis-je ?*
- Chaque attribut est défini par un nom, un type, une visibilité et une valeur qui peut différer d'un objet à un autre.
- Dans le cas général, la visibilité d'un attribut est privée.

# Notion d'objet



# Notion d'objet




Un objet est une instance d'une classe dotée de propriétés :

- une identité
- un état ou des propriétés structurelles : attributs
- un comportement : ensemble de méthodes qu'il peut invoquer

# Association

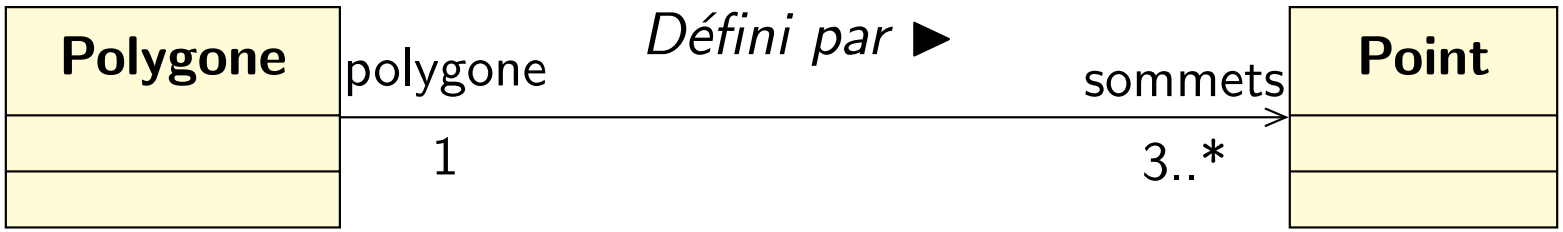
## Définition

Une association est une relation entre des classes qui décrit les connexions structurelles entre leurs instances

# Association

## Définition

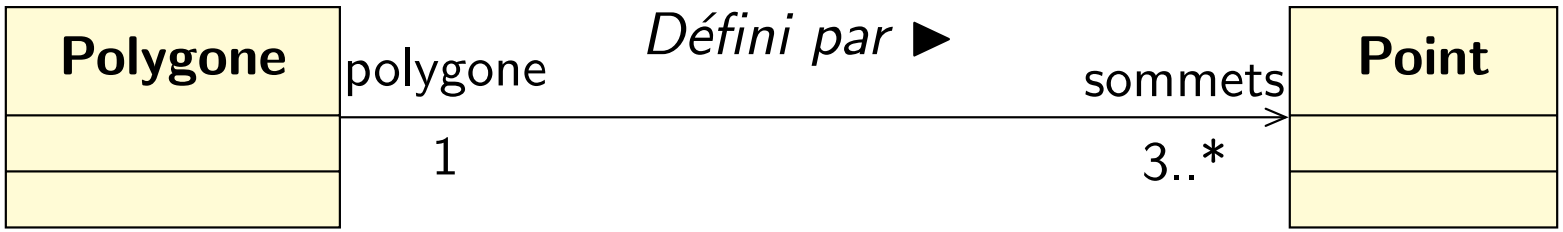
Une association est une relation entre des classes qui décrit les connexions structurelles entre leurs instances



# Association

## Définition

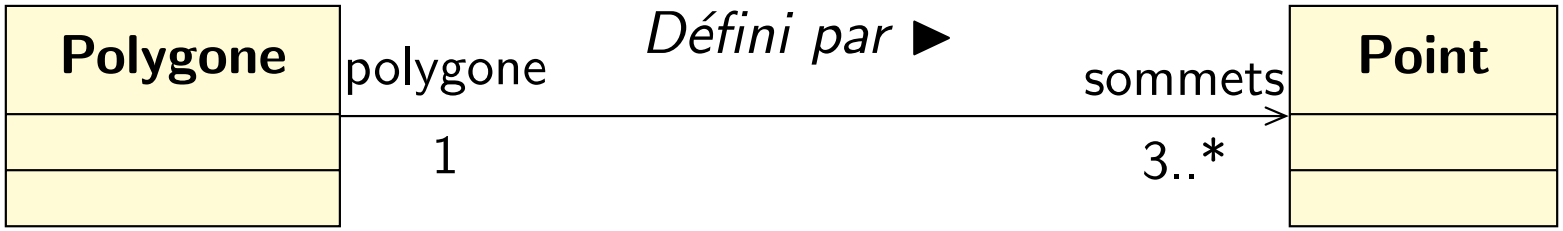
Une association est une relation entre des classes qui décrit les connexions structurelles entre leurs instances



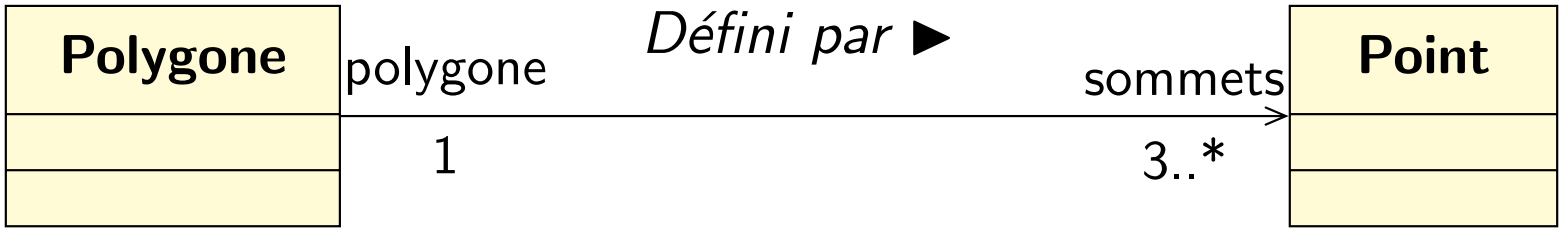

## Représentation graphique

- Une association binaire est matérialisée par un trait plein entre les classes associées.
- Elle peut avoir un nom : celui-ci figure alors au milieu du lien d'association.
- Elle peut avoir un sens de lecture (▶ ou ◀).
- De part et d'autre du lien d'association peuvent figurer des rôles.

# Multiplicité ou cardinalité



# Multiplicité ou cardinalité

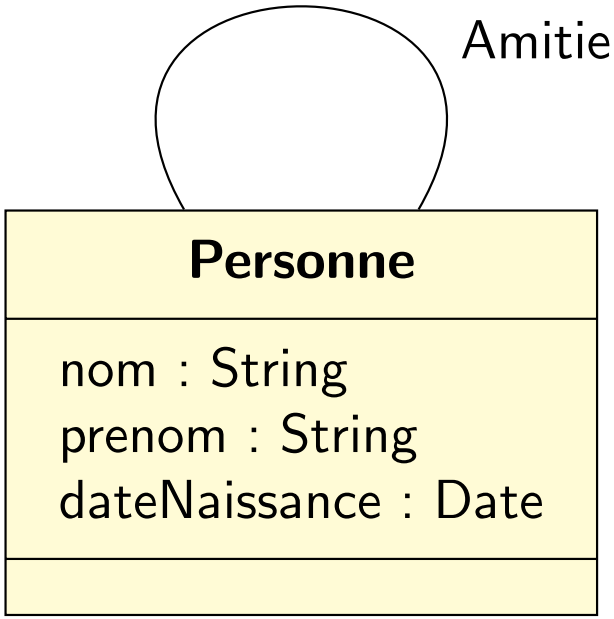



## Association binaire

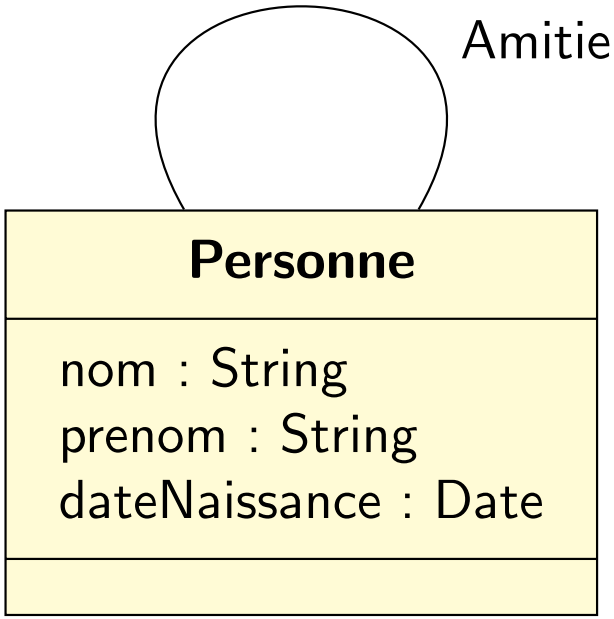
La multiplicité sur la terminaison cible fixe le nombre d'objets de la classe cible pouvant être associés à un seul objet donné de la classe source (la classe de l'autre terminaison de l'association) :

- exactement un : 1 ou 1..1
- plusieurs : $*$ ou 0..$*$
- au moins un : 1..$*$
- de un à six : 1..6

# Association réflexive




# Association réflexive

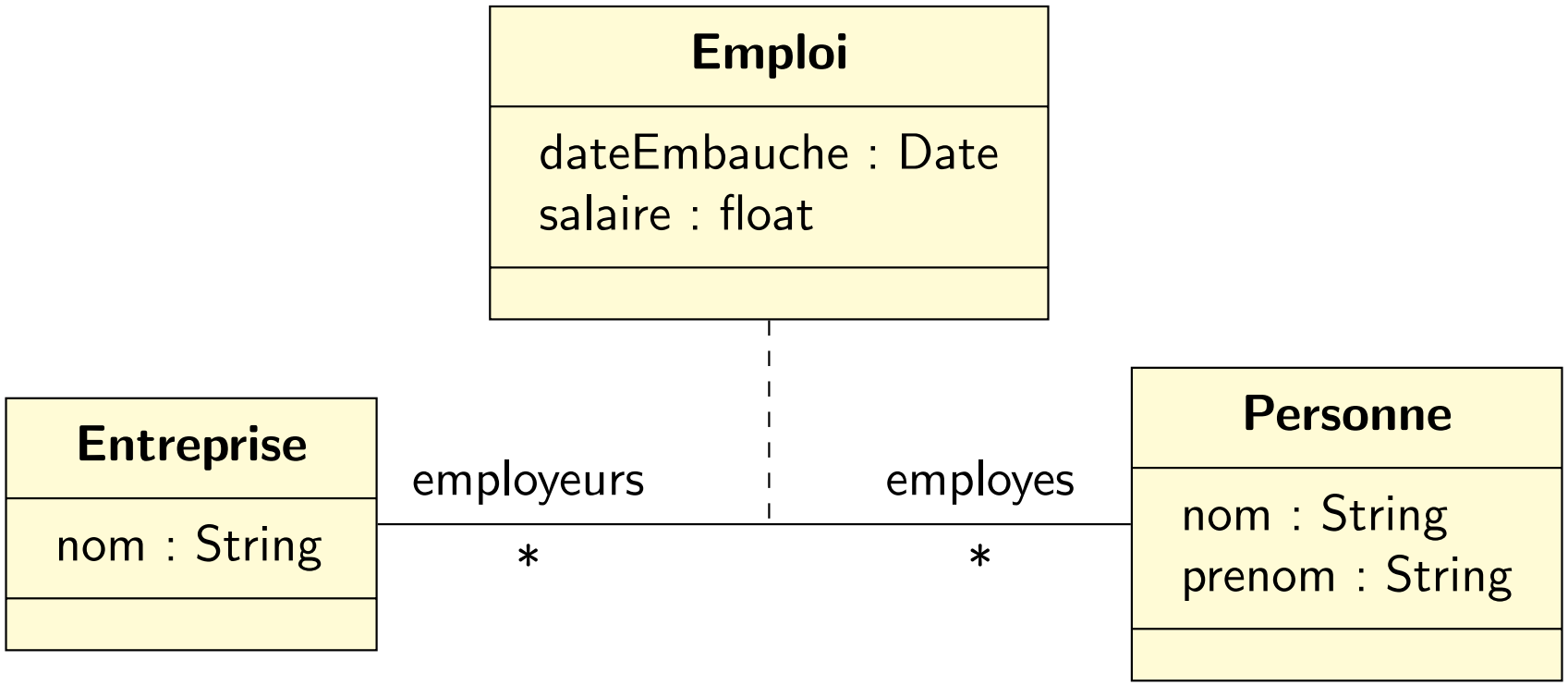




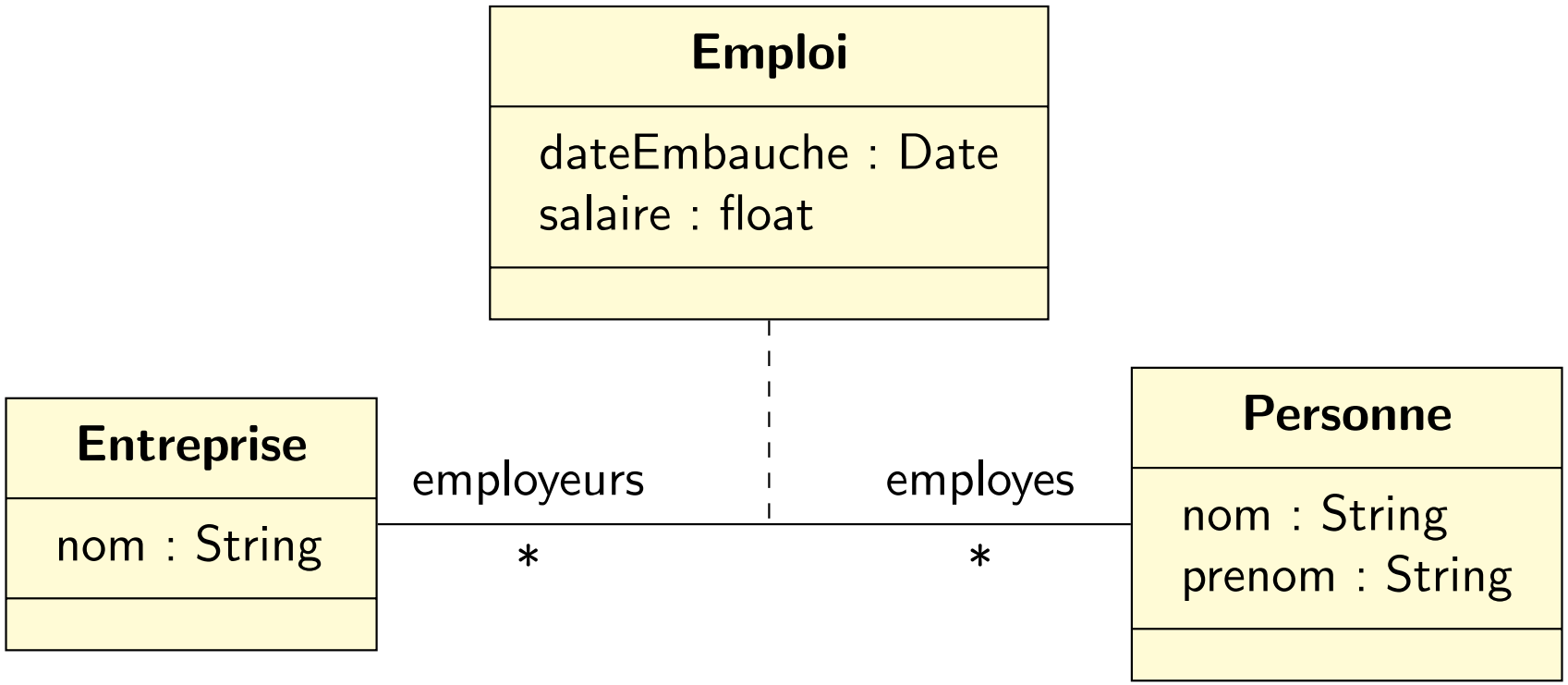
## Définition

Une association est dite réflexive quand les deux extrémités de l'association aboutissent à la même classe.

# Classe-association



# Classe-association

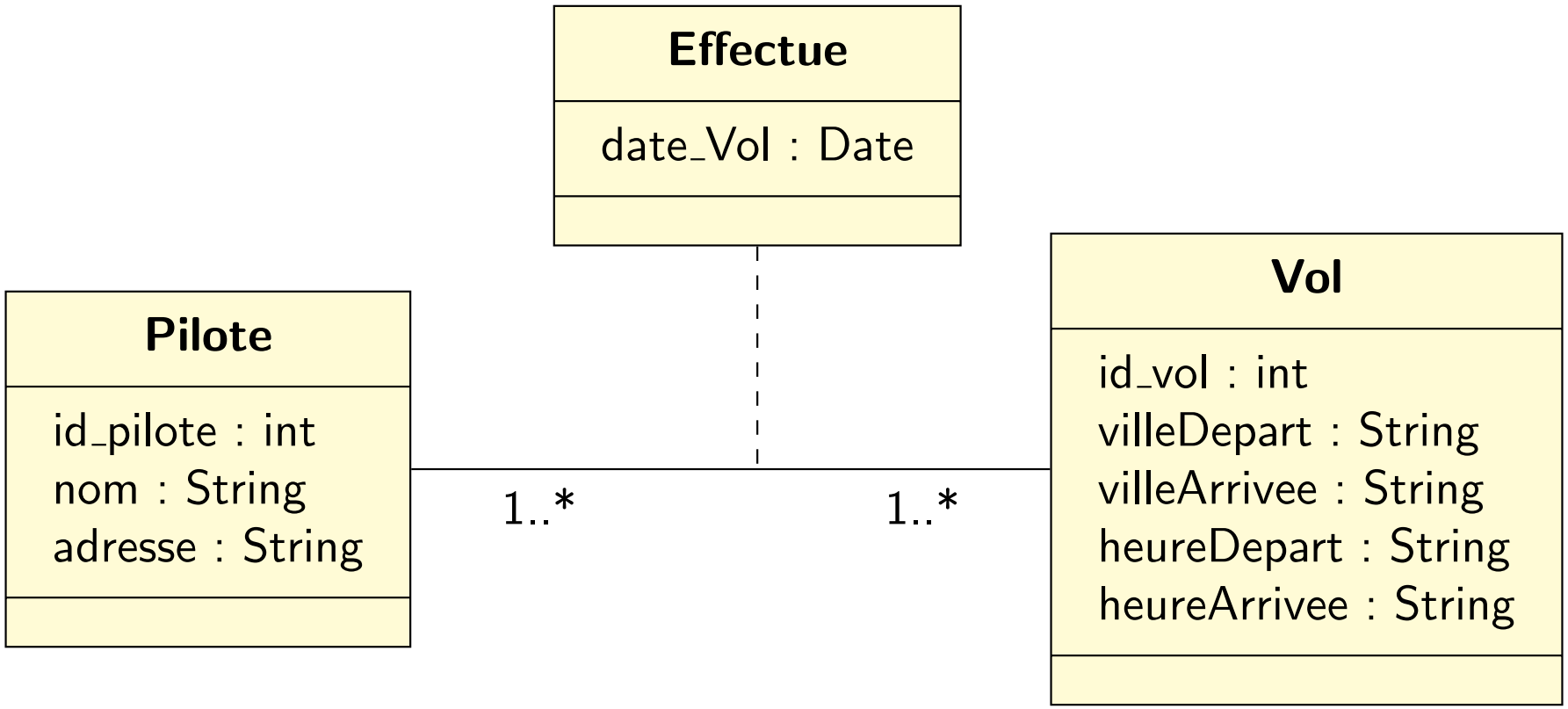



- elle n'existe que pour contenir les attributs d'une association
- elle est reliée à l'association par un trait discontinu

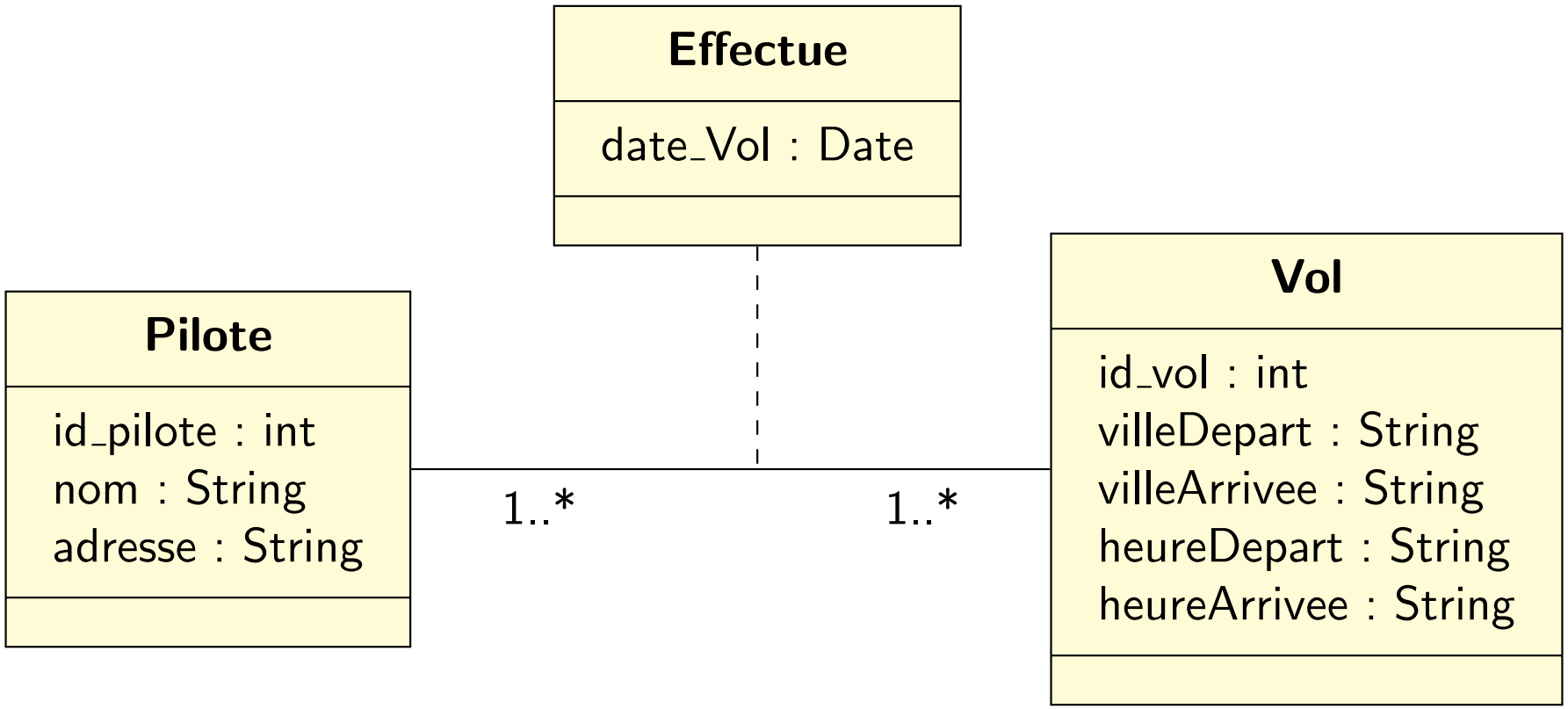
# Diagramme de classes

- diagramme le plus important de la modélisation objet
- permet de modéliser les classes du système et leurs relations indépendamment d'un langage de programmation particulier
- représente graphiquement les classes interconnectées par des associations ou des relations de généralisation
- procure une vue statique du système (on ne tient pas compte du facteur temporel)
- Principaux éléments : les classes et leurs relations ou associations

# Diagramme de classes : exemple



# Diagramme de classes : exemple




Dans une optique *bases de données*, on ne s'intéresse qu'aux attributs des classes.

# Diagramme de classes

## Identifiant

Lorsqu'elle est impliquée dans un projet *base de données*, une classe doit avoir un attribut jouant le rôle d'identifiant unique.

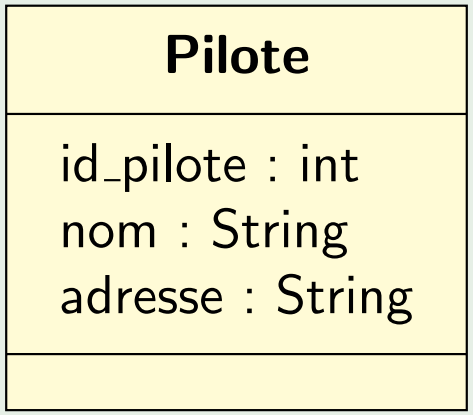
# Diagramme de classes

## Identifiant

Lorsqu'elle est impliquée dans un projet *base de données*, une classe doit avoir un attribut jouant le rôle d'identifiant unique.

## Exemple

`id_pilote` est un identifiant de la classe pilote.



# Schéma Relationnel

## Schéma Relationnel

fournit sous forme de relations une description simple des classes et de certaines associations du diagramme de classes.

# Schéma Relationnel

## Schéma Relationnel

fournit sous forme de relations une description simple des classes et de certaines associations du diagramme de classes.

## Relation

Une relation $R$ est un ensemble d'attributs.

# Schéma Relationnel

## Schéma Relationnel

fournit sous forme de relations une description simple des classes et de certaines associations du diagramme de classes.

## Relation

Une relation $R$ est un ensemble d'attributs.

## Exemple incomplet

PILOTE(id_pilote, nom, adresse)

VOL(id_vol, ville_départ, ville_arrivée, heure_départ, heure_arrivée)

# Schéma Relationnel

## Schéma Relationnel

fournit sous forme de relations une description simple des classes et de certaines associations du diagramme de classes.

## Relation

Une relation $R$ est un ensemble d'attributs.

## Exemple incomplet

PILOTE(id_pilote, nom, adresse)

VOL(id_vol, ville_départ, ville_arrivée, heure_départ, heure_arrivée)

La relation PILOTE a pour attributs : id_pilote, nom et adresse

# Schéma Relationnel

## Relation : instanciation

Une relation est instanciée par une table.

# Schéma Relationnel

## Relation : instanciation

Une relation est instanciée par une table.

## Exemple

PILOTE

| id_pilote | nom | adresse |
|---|---|---|
| 3 | Garratt | Perth |
| 4 | Durand | Paris |
| 5 | MacMachin | Glasgow |

# Schéma Relationnel

## Clé Primaire

- Une clé primaire d'une relation est un attribut ou un groupe d'attributs de la relation qui identifie un tuple unique.
- Une relation possède une et une seule clé primaire, mais peut contenir plusieurs clés qui pourraient jouer ce rôle (clés candidates).
- Dans le cas d'une relation issue d'une classe, la clé primaire correspond à l'identifiant de la classe.

# Schéma Relationnel

## Clé Primaire

- Une clé primaire d'une relation est un attribut ou un groupe d'attributs de la relation qui identifie un tuple unique.
- Une relation possède une et une seule clé primaire, mais peut contenir plusieurs clés qui pourraient jouer ce rôle (clés candidates).
- Dans le cas d'une relation issue d'une classe, la clé primaire correspond à l'identifiant de la classe.

## Exemple

PILOTE(id_pilote, nom, adresse)

VOL(id_vol, ville_départ, ville_arrivée, heure_départ,heure_arrivée)

# Schéma Relationnel

## Clé Primaire

- Une clé primaire d'une relation est un attribut ou un groupe d'attributs de la relation qui identifie un tuple unique.
- Une relation possède une et une seule clé primaire, mais peut contenir plusieurs clés qui pourraient jouer ce rôle (clés candidates).
- Dans le cas d'une relation issue d'une classe, la clé primaire correspond à l'identifiant de la classe.

## Exemple

PILOTE(id_pilote, nom, adresse)

VOL(id_vol, ville_départ, ville_arrivée, heure_départ,heure_arrivée)

id_pilote est la clé primaire de la relation PILOTE.

# Schéma Relationnel : Contrainte d'Intégrité et valeur nulle

## Contrainte d'intégrité

Tout SGBD relationnel doit vérifier l'unicité et le caractère défini (non nul) des valeurs de la clé primaire.

# Schéma Relationnel : Contrainte d'Intégrité et valeur nulle

## Contrainte d'intégrité

Tout SGBD relationnel doit vérifier l'unicité et le caractère défini (non nul) des valeurs de la clé primaire.

## Valeur nulle

- Lors de l'insertion de tuples dans une relation, il arrive qu'un attribut soit inconnu ou non défini. On introduit alors une valeur conventionnelle, appelée valeur nulle.
- Une clé primaire ne peut pas avoir une valeur nulle.

# Schéma Relationnel : Clé Étrangère

## Clé Étrangère

Une clé étrangère dans une relation est une clé primaire dans une autre relation.

# Schéma Relationnel : Clé Étrangère

## Clé Étrangère

Une clé étrangère dans une relation est une clé primaire dans une autre relation.

## Exemple

MODÈLE

| type | constructeur | capacité |
|---|---|---|
| A340 | Airbus | 228 |
| B747 | Boeing | 432 |

AVION

| id_avion | type | compagnie |
|---|---|---|
| 10 | A340 | Air France |
| 20 | B747 | British Airways |
| 30 | B747 | Qantas |

# Schéma Relationnel : Clé Étrangère

## Clé Étrangère

Une clé étrangère dans une relation est une clé primaire dans une autre relation.

## Exemple

MODÈLE

| type | constructeur | capacité |
|---|---|---|
| A340 | Airbus | 228 |
| B747 | Boeing | 432 |

AVION

| id_avion | type | compagnie |
|---|---|---|
| 10 | A340 | Air France |
| 20 | B747 | British Airways |
| 30 | B747 | Qantas |

type est la clé primaire de la relation MODÈLE et une clé étrangère de la relation AVION.

# Du Diagramme de Classes au Schéma Relationnel

# Diagramme de classes → Schéma relationnel

## Classes

- Chaque classe du diagramme UML devient une relation contenant tous les attributs de la classe.
- Si la classe possède un identifiant, il devient la clé primaire, sinon il faut ajouter une clé primaire arbitraire.
- Les méthodes ne sont pas traduites.

# Diagramme de classes → Schéma relationnel

## Association un à plusieurs

- Ajouter un attribut clé étrangère dans la relation de cardinalité N.
- Cette clé étrangère est la clé primaire de l'autre relation (celle de cardinalité 1).

# Diagramme de classes → Schéma relationnel

## Association un à plusieurs

- Ajouter un attribut clé étrangère dans la relation de cardinalité N.
- Cette clé étrangère est la clé primaire de l'autre relation (celle de cardinalité 1).

## Association un à un

- Il faut rajouter un attribut de type clé étrangère dans la relation dérivée de la classe de cardinalité minimale égale à 1.
- Cette clé étrangère est la clé primaire de l'autre relation.
- Si les deux cardinalités minimales sont à zéro, on choisit arbitrairement la relation recevant la clé étrangère.

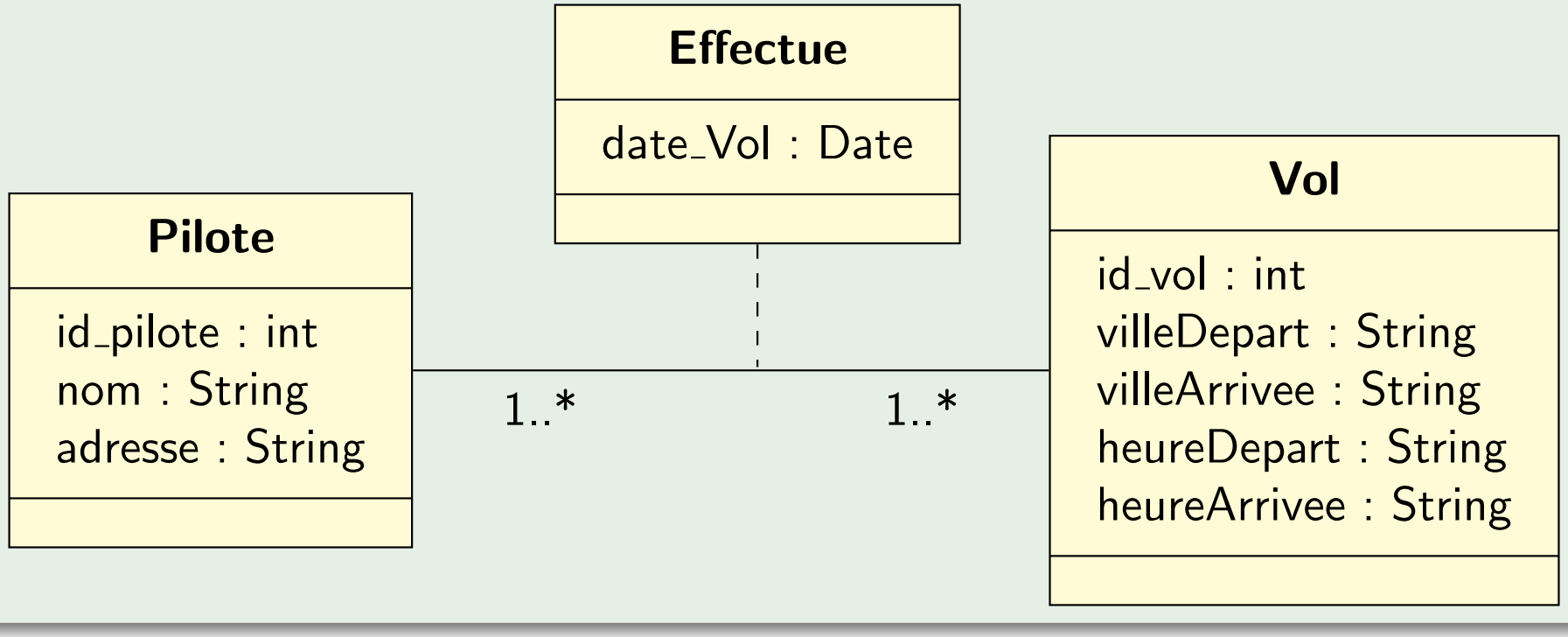
# Diagramme de classes $\rightarrow$ Schéma relationnel

## Association plusieurs à plusieurs

- L'association ou la classe-association devient une relation dont la clé primaire est composée par la concaténation des clés primaires des classes connectés à l'association.
- Ces attributs deviennent clé étrangères dans la nouvelle relation.
- Les attributs de la classe-association doivent être ajoutés à la nouvelle relation.
- Ces attributs peuvent faire partie de la clé étrangère.

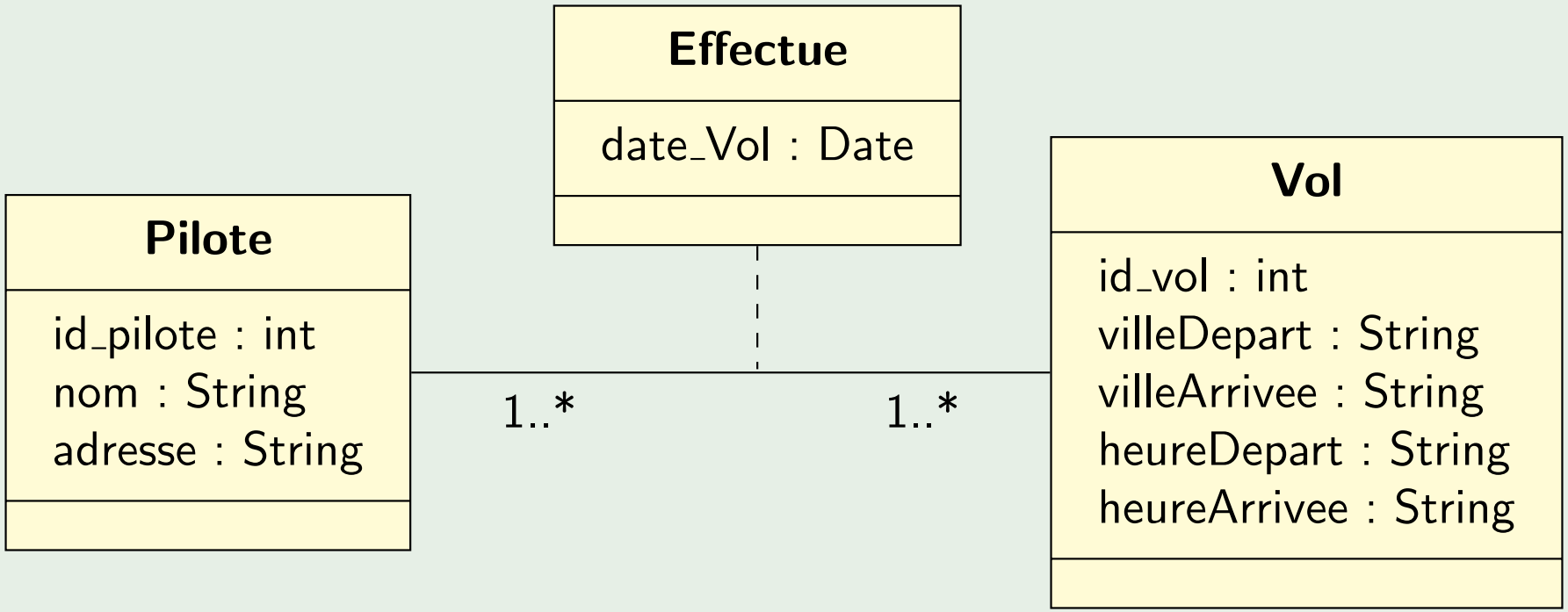
# Diagramme de classes → Schéma relationnel : Exemple

## Diagramme de classes



# Diagramme de classes → Schéma relationnel : Exemple

## Diagramme de classes




## Schéma relationnel

PILOTE(id_pilote, nom, adresse)
VOL(id_vol, ville_départ, ville_arrivée, heure_départ, heure_arrivée)
EFFECTUE(id_pilote, id_vol, date_vol)

# SQL

# SQL — Introduction

- langage déclaratif permettant de :
  - créer, modifier et interroger une base de données relationnelle
  - contrôler la sécurité et l'intégrité de la base
- langage relationnel : on manipule des tables et on obtient des tables
  Une instruction SQL est une requête

# SQL — Introduction

- langage déclaratif permettant de :
  - créer, modifier et interroger une base de données relationnelle
  - contrôler la sécurité et l'intégrité de la base
- langage relationnel : on manipule des tables et on obtient des tables
  Une instruction SQL est une requête

## langage déclaratif

permet de décrire ce que l'on souhaite obtenir sans détailler les moyens de l'obtenir (par opposition à un langage procédural type langage C qui impose de décrire en détail toutes les actions nécessaires).

# Création d'une Table

```
CREATE TABLE nomTable (
    nomColonne type contrainte_colonne,
    . . .,
    contrainte_table,
    . . .
);
```

contrainte_colonne peut être :

- NOT NULL
- UNIQUE
- PRIMARY KEY
- DEFAULT valeur
- CHECK condition

# Création d'une Table

```
CREATE TABLE nomTable (
    nomColonne type contrainte_colonne,
    . . .,
    contrainte_table,
    . . .
);
```

contrainte_colonne peut être :

- NOT NULL
- UNIQUE
- PRIMARY KEY
- DEFAULT valeur
- CHECK condition

contrainte_table est :
FOREIGN KEY (référenceColonne) REFERENCES référenceTable

# Exemples

```
create table Pilote(
    id_pilote smallint primary key,
    nom varchar(20),
    adresse varchar(30)
);
create table Vol(
    id_vol varchar(10) primary key,
    ville_depart varchar(20),
    ville_arrivee varchar(20),
    heure_depart time,
    heure_arrivee time
);
```

# Exemples

```
create table Pilote(
    id_pilote smallint primary key,
    nom varchar(20),
    adresse varchar(30)
);
create table Vol(
    id_vol varchar(10) primary key,
    ville_depart varchar(20),
    ville_arrivee varchar(20),
    heure_depart time,
    heure_arrivee time
);
```

## Remarque

Notez l'absence de virgule à la fin de la dernière ligne.

# Exemples

```
create table Effectue(
    id_pilote smallint references Pilote,
    id_vol varchar(10) references Vol,
    date Date,
    primary key(id_pilote,id_vol,date)
);
```

# Exemples

## Autre façon de déclarer les clés étrangères

```
create table Effectue(
    id_pilote smallint,
    id_vol varchar(10),
    date Date,
    foreign key (id_pilote) references Pilote,
    foreign key (id_vol) references Vol,
    primary key (id_pilote,id_vol,date)
);
```

# Contraintes

Les contraintes permettent d'exprimer des conditions devant être respectées par tous les tuples d'une table.

# Contraintes

Les contraintes permettent d'exprimer des conditions devant être respectées par tous les tuples d'une table.

## Contraintes de Domaine

Les contraintes de domaine expriment les valeurs que peuvent prendre un attribut :

- `NOT NULL` : l'attribut doit posséder une valeur
- `DEFAULT` : valeur par défaut de l'attribut (quand il n'est pas défini)
- `UNIQUE` : deux tuples ne peuvent pas avoir la même valeur pour cet attribut
- `CHECK` : spécifie une condition devant être satisfaite par tous les tuples de la table

# Exemple

```
create table Modele(
    type varchar(20) primary key,
    constructeur varchar(30) not null,
    capacite smallint check(capacite>0)
);
```

# Suppression/Modification

Suppression d'une table : `DROP TABLE nom_table;`

# Suppression/Modification

Suppression d'une table : `DROP TABLE nom_table;`

Modification d'une table : `ALTER TABLE nom_table SET...;`

# Insertion de Tuples

```
INSERT INTO table [(column [,...])] {
    DEFAULT VALUES | VALUES (expression [,...])}
```

# Insertion de Tuples

```
INSERT INTO table [(column [,...])] {
    DEFAULT VALUES | VALUES (expression [,...])}
```

## Exemple

```
insert into Avion values (2,'B707','Qantas');
insert into Avion values (3,'B737','British Airways');
insert into Avion values (4,'A320','Air France');
insert into Avion (id_avion,type) values (5,'A320');
insert into Avion values (6,'A320',null);
insert into Avion values (8,'B737','Air France');
```

# Insertion de Tuples

## Résultat

```
select * from Avion;

 id_avion | type | compagnie
----------+------+-------------
        2 | B707 | Qantas
        3 | B737 | British Airways
        4 | A320 | Air France
        5 | A320 |
        6 | A320 |
        8 | B737 | Air France
```

# Mise à Jour

```
UPDATE table SET col=expression [,...]
    [WHERE condition]
```

# Mise à Jour

```
UPDATE table SET col=expression [,...]
    [WHERE condition]
```

## Exemple

```
update Effectue set date_vol='2020-02-07'
    where id_vol='BA302';
```

# Suppression

```
DELETE FROM table [WHERE condition]
```

# Suppression

```
DELETE FROM table [WHERE condition]
```

## Exemple

```
delete from Avion where Avion.id_avion=6;
```

# SELECT

```
SELECT [ALL | DISTINCT [ON (expression [,...])]]
    * | expression [AS nom_sortie] [,...]
    [FROM table [,...]]
    [WHERE condition]
```

# Projection

Pour faire une projection, on utilise `select` en désignant les attributs sur lesquels la projection est effectuée.
On obtient plusieurs fois la même ligne si les mêmes attributs figurent en plusieurs exemplaires dans la projection.

# Projection

Pour faire une projection, on utilise `select` en désignant les attributs sur lesquels la projection est effectuée.
On obtient plusieurs fois la même ligne si les mêmes attributs figurent en plusieurs exemplaires dans la projection.

## Quels sont les différents types d'avions ?

```
select Avion.type from Avion;

 type
------
 B707
 B737
 A320
 A320
 B737
```

# Projection sans Doublon

```
select distinct Avion.type from Avion;

 type
------
 B707
 B737
 A320
```

# Sélection : La clause `WHERE`

- La clause `WHERE` permet de spécifier un critère de sélection, appelé *prédicat*. Si un tuple satisfait le prédicat, il fera partie du résultat.
- Le prédicat est une expression logique composée d'une suite de conditions combinées par les opérateurs logiques `AND`, `OR` ou `NOT`.
- Un élément d'une expression peut prendre une des formes suivantes :
  - comparaison à une valeur : `=`, `!=`, `<>`, `<`, `>`, `<=`, `>=`
  - comparaison à une fourchette de valeurs : `between`
  - comparaison à une liste de valeurs : `in`
  - comparaison à un filtre : `like`, `~`
  - test sur l'indétermination d'une valeur : `is null`
  - test *tous* : `all`
  - test *au moins un* : `any`

# Exemple

## Quels sont les vols partant entre 14h et 18h ?

| id_vol | ... | heure_depart | heure_arrivee |
|---|---|---|---|
| AF1232 | ... | 15:30:00 | 22:00:00 |
| BA302 | ... | 09:15:00 | 16:54:00 |
| QT17 | ... | 14:30:00 | 15:30:00 |

# Exemple

## Quels sont les vols partant entre 14h et 18h ?

```
 id_vol | ... | heure_depart | heure_arrivee
--------+-----+---------------+---------------
 AF1232 | ... | 15:30:00      | 22:00:00
 BA302  | ... | 09:15:00      | 16:54:00
 QT17   | ... | 14:30:00      | 15:30:00
```

```
select Vol.* from Vol where Vol.heure_depart
    between '14:00:00' and '18:00:00';
```

```
 id_vol | ... | heure_depart | heure_arrivee
--------+-----+---------------+---------------
 AF1232 | ... | 15:30:00      | 22:00:00
 QT17   | ... | 14:30:00      | 15:30:00
```

# Autres Exemples

```
select Vol.* from Vol
    where Vol.ville_depart in ('Londres','Paris');

select Pilote.* from Pilote where Pilote.nom ~ '^D';

select Pilote.* from Pilote where Pilote.adresse is null;
```

# Tri des Tuples

- Pour trier le résultat, on utilise la clause `ORDER BY` suivie éventuellement de `ASC` (tri ascendant) ou `DESC` (tri descendant).
- Si rien n'est précisé, les tuples apparaissent dans l'ordre dans lequel ils ont été trouvés.

# Tri des Tuples

- Pour trier le résultat, on utilise la clause `ORDER BY` suivie éventuellement de `ASC` (tri ascendant) ou `DESC` (tri descendant).
- Si rien n'est précisé, les tuples apparaissent dans l'ordre dans lequel ils ont été trouvés.

**Quels sont les avions n'appartenant pas à la compagnie Qantas triés par numéro d'avion décroissant ?**

```
select Avion.* from Avion
    where Avion.compagnie!='Qantas'
    order by Avion.id_avion desc;
```

# Jointure

La jointure permet d'extraire des informations reliées mais situées dans différentes tables.

# Jointure

La jointure permet d'extraire des informations reliées mais situées dans différentes tables.

**Quels sont les noms des pilotes qui ont assuré le vol Londres–Sydney de 09 :15 ?**

Pour effectuer une jointure, il suffit de :

- citer les attributs recherchés dans la clause `select` :
  select Pilote.nom ...
- lister dans la clause `from` les tables concernées par la jointure :
  select Pilote.nom from Pilote,Effectue,Vol ...
- préciser dans la clause `where` la condition portant sur les attributs sur lesquels la jointure est faite :

```
select Pilote.nom from Pilote,Effectue,Vol
    where Pilote.id_pilote=Effectue.id_pilote
        and Effectue.id_vol=Vol.id_vol ...
```

# Jointure

**Quels sont les noms des pilotes qui ont assuré le vol Londres–Sydney de 09 :15 ?**

- préciser dans la clause `where` les conditions particulières à la requête :

```
select Pilote.nom from Pilote,Effectue,Vol
  where Pilote.id_pilote=Effectue.id_pilote
      and Effectue.id_vol=Vol.id_vol
      and Vol.ville_depart='Londres'
      and Vol.ville_arrivee='Sydney'
      and Vol.heure_dep='09:15:00';
```

# Jointure Naturelle : NATURAL JOIN

Permet de ne pas spécifier les attributs sur lesquels la jointure est effectuée : SQL choisit automatiquement les attributs de même nom dans les tables comme attributs de liaison.

# Jointure Naturelle : NATURAL JOIN

Permet de ne pas spécifier les attributs sur lesquels la jointure est effectuée : SQL choisit automatiquement les attributs de même nom dans les tables comme attributs de liaison.

## Quels sont les différentes combinaisons d'avions et de pilotes utilisées sur les vols ?

```
select distinct Pilote.nom,Vol.id_vol
    from Pilote natural join Effectue natural join Vol;

    nom     | id_vol
------------+--------
 Dupond     | AF1232
 Smith      | QT17
 Garratt    | QT17
 MacMachin  | BA302
 Garratt    | BA302
```

# Auto-Jointure

Permet de traîter une requête comportant un critère comparant la valeur d'un attribut avec celle du même attribut dans un autre tuple de la même table. Pour distinguer les deux versions de la table, on utilise des *alias*.

# Auto-Jointure

Permet de traîter une requête comportant un critère comparant la valeur d'un attribut avec celle du même attribut dans un autre tuple de la même table. Pour distinguer les deux versions de la table, on utilise des *alias*.

## Quels sont les numéros des vols dont l'heure de départ est après celle du vol Sydney-Perth ?

```
select tard.id_vol,tard.heure_depart,SP.heure_depart
    from Vol as tard join Vol as SP
        on tard.heure_depart>SP.heure_depart
    where SP.ville_depart='Sydney'
        and SP.ville_arrivee='Perth';

 id_vol  | heure_depart  | heure_depart
---------+---------------+--------------
 AF1232  | 15:30:00      | 14:30:00
```

# Union

L'opérateur `UNION` effectue l'union des résultats de deux requêtes `select` en éliminant les doublons parmi les tuples.
Les attributs sélectionnés dans les deux `select` doivent être identiques.

# Union

L'opérateur `UNION` effectue l'union des résultats de deux requêtes `select` en éliminant les doublons parmi les tuples.
Les attributs sélectionnés dans les deux `select` doivent être identiques.

**Quels sont les avions d'`Air France` assurant des vols depuis `Paris` et ceux de `British Airways` assurant des vols depuis `Londres` ?**

```
select Avion.* from Avion,Vol,Effectue
    where Avion.id_avion=Effectue.id_avion
        and Vol.id_vol=Effectue.id_vol
        and Avion.compagnie='Air France'
        and Vol.ville_depart='Paris'
union
select Avion.* from Avion,Vol,Effectue
    where Avion.id_avion=Effectue.id_avion
        and Vol.id_vol=Effectue.id_vol
        and Avion.compagnie='British Airways'
        and Vol.ville_depart='Londres';
```

# Intersection

L'opérateur `INTERSECT` effectue l'intersection des résultats de deux requêtes `select`. On obtient une table contenant les tuples communs aux deux tables de départ.
Les attributs sélectionnés dans les deux `select` doivent être identiques.

# Intersection

L'opérateur `INTERSECT` effectue l'intersection des résultats de deux requêtes `select`. On obtient une table contenant les tuples communs aux deux tables de départ.
Les attributs sélectionnés dans les deux `select` doivent être identiques.

**Quels sont les avions assurant à la fois des vols depuis `Londres` et depuis `Paris` ?**

```
select Avion.* from Avion,Vol,Effectue
    where Avion.id_avion=Effectue.id_avion
        and Vol.id_vol=Effectue.id_vol
        and Vol.ville_depart='Londres'
intersect
select Avion.* from Avion,Vol,Effectue
    where Avion.id_avion=Effectue.id_avion
        and Vol.id_vol=Effectue.id_vol
        and Vol.ville_depart='Paris';
```

# Différence

L'opérateur `EXCEPT` effectue la différence des résultats de deux requêtes `select`.
On obtient une table contenant les tuples de la première requête qui n'apparaissent pas dans la deuxième.
Les attributs sélectionnés dans les deux `select` doivent être identiques.

# Différence

L'opérateur `EXCEPT` effectue la différence des résultats de deux requêtes `select`.
On obtient une table contenant les tuples de la première requête qui n'apparaissent pas dans la deuxième.
Les attributs sélectionnés dans les deux `select` doivent être identiques.

## Quels sont les avions assurant des vols depuis Londres mais pas depuis Paris ?

```
select Avion.* from Avion,Vol,Effectue
    where Avion.id_avion=Effectue.id_avion
        and Vol.id_vol=Effectue.id_vol
        and Vol.ville_depart='Londres'
except
select Avion.* from Avion,Vol,Effectue
    where Avion.id_avion=Effectue.id_avion
        and Vol.id_vol=Effectue.id_vol
        and Vol.ville_depart='Paris';
```

# ALL et ANY

- La requête se place dans la clause `where` avec un opérateur de comparaison suivi de `ALL` ou `ANY`.
- `ALL` : la condition est vraie si et seulement si elle est vraie pour toutes les valeurs produites.
- `ANY` : la condition est vraie si et seulement si elle est vraie pour au moins une valeur produite.

# Exemples

## Quels sont les types d'avions du constructeur Boeing dont la capacité est supérieure à celle d'au moins un avion du constructeur Airbus ?

```
select distinct Avion.type from Avion natural join Modele
    where Modele.constructeur='Boeing'
        and Modele.capacite > ANY
            (select Modele.capacite from Modele
                where Modele.constructeur='Airbus');
```

# Exemples

## Quels sont les types d'avions du constructeur Boeing dont la capacité est supérieure à celle d'au moins un avion du constructeur Airbus ?

```
select distinct Avion.type from Avion natural join Modele
    where Modele.constructeur='Boeing'
        and Modele.capacite > ANY
            (select Modele.capacite from Modele
                where Modele.constructeur='Airbus');
```

## Quels sont les types d'avions du constructeur Boeing dont la capacité est supérieure à celle de tous les avions du constructeur Airbus ?

```
select distinct Avion.type from Avion natural join Modele
    where Modele.constructeur='Boeing'
        and Modele.capacite > ALL
            (select Modele.capacite from Modele
                where Modele.constructeur='Airbus');
```

# EXISTS

```
... where exists (select...);
```

La condition est vérifiée si la requête imbriquée renvoie au moins un tuple.

# EXISTS

```
... where exists (select...);
```

La condition est vérifiée si la requête imbriquée renvoie au moins un tuple.

## Quels sont les avions qui assurent au moins un vol depuis Londres ?

```
select * from Avion
    where exists (select * from Effectue natural join Vol
                    where Effectue.id_avion=Avion.id_avion
                        and Vol.ville_depart='Londres');
```

# EXISTS

```
... where exists (select...);
```

La condition est vérifiée si la requête imbriquée renvoie au moins un tuple.

## Quels sont les avions qui assurent au moins un vol depuis Londres ?

```
select * from Avion
    where exists (select * from Effectue natural join Vol
                    where Effectue.id_avion=Avion.id_avion
                        and Vol.ville_depart='Londres');
```

## Quels sont les avions qui n'assurent pas de vol depuis Londres ?

```
select * from Avion
    where not exists (select * from Effectue natural join Vol
                        where Effectue.id_avion=Avion.id_avion
                            and Vol.ville_depart='Londres');
```

# Détermination des Groupes

- Un groupe est un sous-ensemble des tuples d'une table ayant la même valeur pour un attribut.
- Un groupe est déterminé par la clause `GROUP BY` suivie du nom de l'attribut sur lequel s'effectue le regroupement.
- La clause `GROUP BY` réarrange la table résultat d'un `SELECT` par groupes.
- Lorsqu'une clause `GROUP BY` est précisée, on peut utiliser des fonctions portant sur les groupes.

# Fonctions sur les Groupes

- `COUNT` : compte le nombre d'occurrences de l'attribut.
- `SUM` : calcule la somme des valeurs de l'attribut.
- `AVG` : calcule la moyenne des valeurs de l'attribut.
- `MAX` : recherche la plus grande valeur de l'attribut.
- `MIN` : recherche la plus petite valeur de l'attribut.

# Exemples

## Combien y a-t-il d'avions ?

```
select count(*) from avion;

 count
-------
     5
```

# Exemples

## Combien y a-t-il d'avions ?

```
select count(*) from avion;

 count
-------
     5
```

## Combien y a-t-il d'avions de chaque type ?

```
select type,count(*) from avion group by type;

 type | count
------+-------
 B707 |     1
 B737 |     2
 A320 |     2
```

# Exemples

## Quelle est la capacité moyenne des modèles d'avions ?

```
select avg(capacite) from modele;

          avg
----------------------
 241.8000000000000000
```

# Clause HAVING

- La clause HAVING est l'équivalent du WHERE appliqué aux groupes.
- Le critère spécifié dans la clause HAVING porte sur la valeur d'une fonction calculée sur un groupe.

# Clause HAVING

- La clause HAVING est l'équivalent du WHERE appliqué aux groupes.
- Le critère spécifié dans la clause HAVING porte sur la valeur d'une fonction calculée sur un groupe.

## Quelles sont les compagnies possédant au moins deux avions ?

```
select compagnie, count(*) from avion
    group by compagnie having count(*)>=2;

 compagnie   | count
-------------+-------
 Air France  |     2
```

# Client Python d'un SGBD

# Objectif

Rédiger et implémenter une API permettant aux applications :

- d'accéder aux bases de données.
- d'exécuter des requêtes SQL.
- de récupérer puis traiter les résultats de ces requêtes.

# La librairie psycopg2

Il faut importer la librairie psycopg2 pour pouvoir accéder aux fonctions de postgresql :

```
import psycopg2
```

# La librairie psycopg2

Il faut importer la librairie `psycopg2` pour pouvoir accéder aux fonctions de `postgresql` :

```
import psycopg2
```

- Plusieurs connexions à des bases de données peuvent avoir lieu simultanément. Chacune est identifiée par un objet.
- Une connexion à la base s'obtient en appelant la fonction `psycopg2.connect`.
- La manipulation de la base se fait via un curseur obtenu avec la méthode `cursor` de l'objet de connection.
- Pour exécuter une requête, on utilise la méthode `execute` du curseur avec la requête SQL en paramètre.

# La librairie psycopg2

Plusieurs attributs et méthodes permettent d'examiner le résultat d'une requête :

- L'attribut `description` du curseur décrit le résultat de la requête.
- L'attribut `rowcount` contient le nombre de lignes du résultat.
- La méthode `fetchall` renvoie l'intégralité de la table résultat.
- La méthode `fetchone` renvoie une ligne de la table.

# Exemple — (début)

```
#!/usr/bin/env python3
import psycopg2

# début : connection et création du curseur
connection = psycopg2.connect(database="mabase",
                              user="utilisateur",
                              host="nom_serveur",
                              password="motdepasse",
                              port="5432")
curseur = connection.cursor()

# requête
requete = "SELECT * FROM avion"
curseur.execute(requete)
```

# Exemple — (suite)

```
# écriture des entêtes de colonnes
nom_col = [col[0] for col in curseur.description]
print(nom_col)

# récupération du résultat
resultat = curseur.fetchall()
# écriture du contenu de la table résultat
for ligne in resultat:
  print(ligne)
```

# Fin de connexion

Lorsque l'on a terminé, on peut libérer la mémoire occupée par le curseur avec sa méthode `close`.
Enfin, la méthode `close` de la connection ferme l'accès à la base de données.

```
# fin
curseur.close()
connection.close()
```