

Bases de Données

Module M2104

IUT R&T Villetaneuse

13 février 2020

Objectif d'une base de données

Objectif d'une base de données

Stocker les données communes à plusieurs applications de façon plus efficace qu'un système de fichiers.

Introduction

Base de données versus SGBD

- Une *base de données* est un ensemble de données reliées entre elles, accessibles à plusieurs utilisateurs simultanément.

Introduction

Base de données versus SGBD

- Une *base de données* est un ensemble de données reliées entre elles, accessibles à plusieurs utilisateurs simultanément.
- Un *SGBD* (Système de Gestion de Bases de Données) est un logiciel permettant de :
 - créer des bases de données
 - les interroger
 - les mettre à jour
 - assurer les contrôles d'intégrité, de concurrence et de sécurité.

Introduction

Base de données versus SGBD

- Une *base de données* est un ensemble de données reliées entre elles, accessibles à plusieurs utilisateurs simultanément.
- Un *SGBD* (Système de Gestion de Bases de Données) est un logiciel permettant de :
 - créer des bases de données
 - les interroger
 - les mettre à jour
 - assurer les contrôles d'intégrité, de concurrence et de sécurité.

Objectifs principaux

- intégration
- indépendance
- disponibilité
- sécurité

Intégration & Indépendance

Intégration

- Dans un système de traitement de données **orienté fichier**, l'application dialogue avec un **grand nombre de fichiers** $\Rightarrow$ risque de **duplication** ou de **perte d'informations**
- Dans **l'approche base de données**, il y a :
 - centralisation de toutes les données en un **réservoir unique de données** commun à toutes les applications
 - centralisation de tous les contrôles d'intégrité et de cohérence

Intégration & Indépendance

Intégration

- Dans un système de traitement de données **orienté fichier**, l'application dialogue avec un **grand nombre de fichiers** $\Rightarrow$ risque de **duplication** ou de **perte d'informations**
- Dans **l'approche base de données**, il y a :
 - centralisation de toutes les données en un **réservoir unique de données** commun à toutes les applications
 - centralisation de tous les contrôles d'intégrité et de cohérence

Indépendance

Indépendance physique : le niveau utilisateur est transparent à un changement :

- de support ou de chemin d'accès aux données
- de méthode d'accès aux données.

Disponibilité & Sécurité

Disponibilité

- **Performance** : tout utilisateur doit ignorer l'existence d'utilisateurs concurrents.

Disponibilité & Sécurité

Disponibilité

- **Performance** : tout utilisateur doit ignorer l'existence d'utilisateurs concurrents.

Sécurité

- **Intégrité** : protection contre la modification invalide des données.
- **Confidentialité** : protection contre l'accès illégal aux données.

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

Personne

- nom : String
- prenom : String
- dateNaissance : Date

- + age() : int

Notion de classe

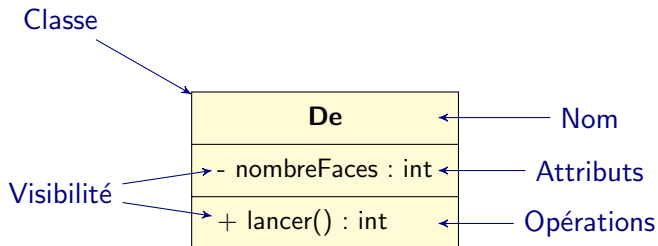
Personne

- nom : String
- prenom : String
- dateNaissance : Date

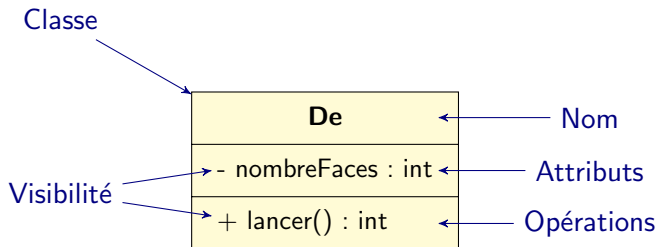
+ age() : int

- 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



Nom

Le **nom** de la classe doit évoquer le **concept** décrit par la classe ; il commence par une majuscule.

Attributs — Opérations ou méthodes

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.

Attributs — Opérations ou méthodes

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.

Opérations ou méthodes

- Les **opérations** décrivent les **actions** qu'un objet peut effectuer : elles répondent à la question : *Que puis-je faire ?*
- Elles peuvent prendre des **valeurs en entrée**, **modifier les attributs** et/ou **produire des résultats**
- L'**implémentation d'une opération** est appelée une **méthode**. Dans le cas général, la visibilité d'une méthode est publique.

Constructeurs

Personne
- nom : String - prenom : String - dateNaissance : Date
+ Personne() + Personne(nom :String, prenom : String, dateNais : Date) + getNom() : String + getPrenom() : String + age() : int

Constructeurs

Personne
- nom : String - prenom : String - dateNaissance : Date
+ Personne() + Personne(nom :String, prenom : String, dateNais : Date) + getNom() : String + getPrenom() : String + age() : int

Un **constructeur** est une opération appelée lors de la création d'un objet.

Notion d'objet

jeanDupont : Personne

nom : Dupont

prenom : Jean

dateNaissance : 29/02/1972

getNom() : String

getPrenom() : String

age() : int

Notion d'objet

jeanDupont : Personne
nom : Dupont prenom : Jean dateNaissance : 29/02/1972
getNom() : String getPrenom() : String age() : int

- Un **objet** est une **instance d'une classe dotée de propriétés** :
 - une **identité**
 - un **état** ou des **propriétés structurelles** : attributs et terminaisons d'associations
 - un **comportement** : ensemble de méthodes qu'il peut invoquer

Notion d'objet

jeanDupont : Personne
nom : Dupont prenom : Jean dateNaissance : 29/02/1972
getNom() : String getPrenom() : String age() : int

- Un **objet** est une **instance d'une classe dotée de propriétés** :
 - une **identité**
 - un **état** ou des **propriétés structurelles** : attributs et terminaisons d'associations
 - un **comportement** : ensemble de méthodes qu'il peut invoquer
- Les objets sont les **éléments d'un programme en cours d'exécution**.

Association

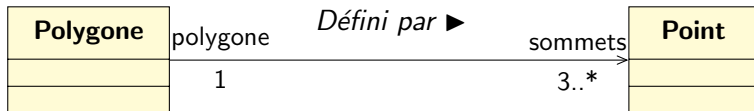
Définition

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

Association

Définition

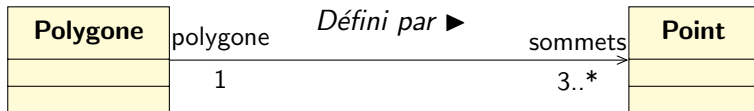
Une **association** est une **relation entre des classes** qui décrit les connexions structurelles entre leurs instances



Association

Définition

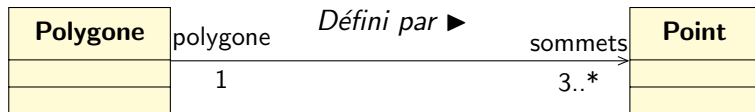
Une **association** est une **relation entre des classes** qui décrit les connexions structurelles entre leurs instances



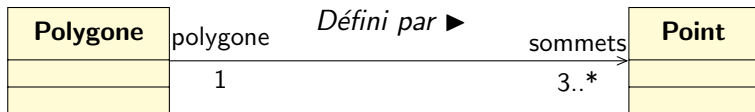
Représentation graphique

- 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

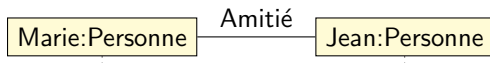
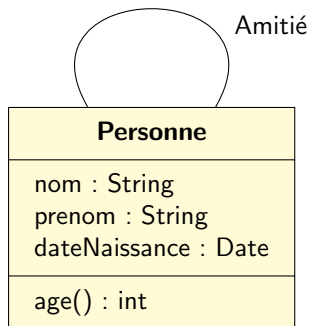


Association binaire

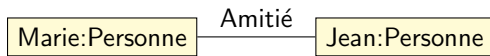
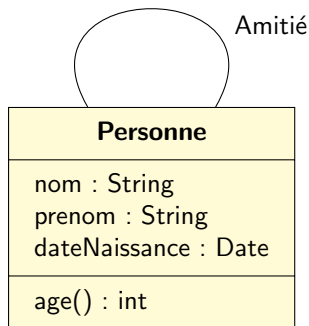
La **multiplicité** sur la terminaison cible fixe le nombre d'objets de la classe cible pouvant être associés à un seul objet donné de la classe source (la classe de l'autre terminaison de l'association) :

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

Association réflexive



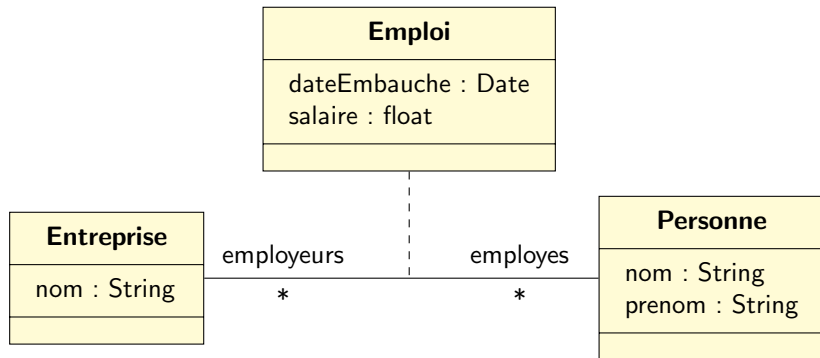
Association réflexive



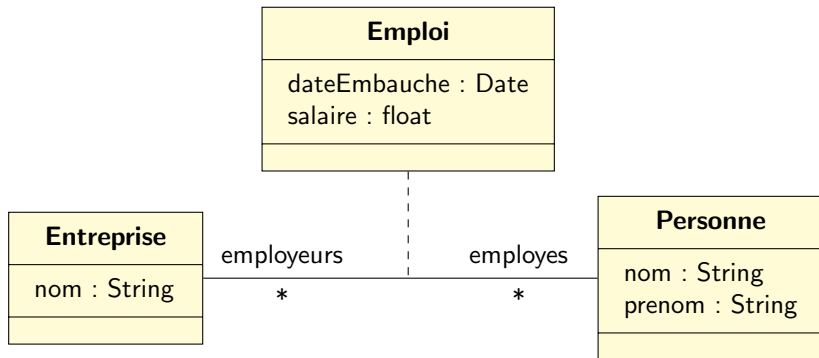
Définition

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**

Élaboration d'un diagramme de classes

- Trouver les **classes** du domaine étudié en collaboration avec un expert du domaine : **concepts** ou **substantifs** du domaine
- Trouver les **associations** entre classes : **verbes** ou **constructions verbales** mettant en relation plusieurs classes (*est composé de, travaille pour...*)
- Trouver les **attributs** des classes : **substantifs** ou **groupes nominaux** (*la masse d'une voiture, le montant d'une transaction...*)

**Attention : se méfier de certains attributs
qui sont en réalité des associations entre classes**

Diagramme de classes : exemple

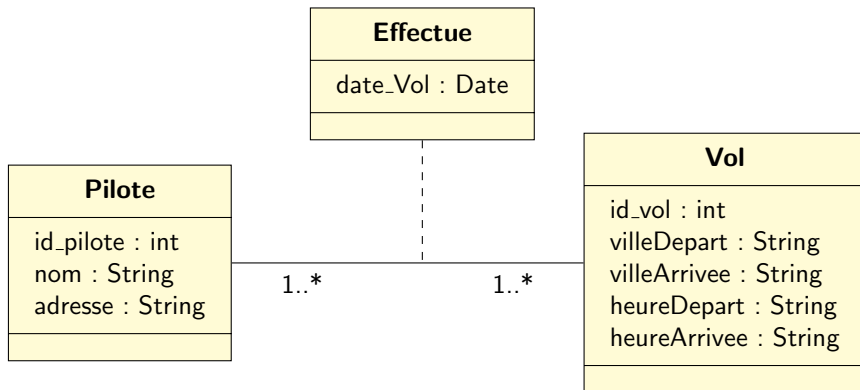
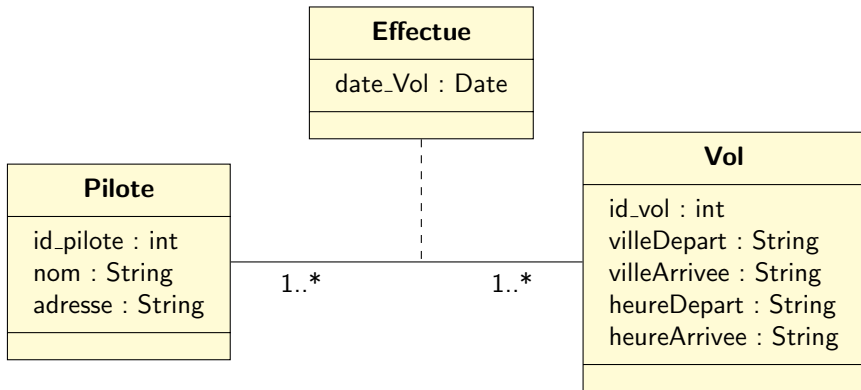


Diagramme de classes : exemple



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.

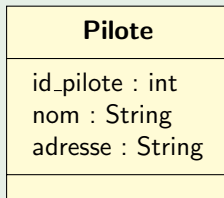


Schéma Relationnel

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 : instantiation

Une **relation** est instanciée par une table.

Schéma Relationnel

Relation : instantiation

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

<u>type</u>	constructeur	capacité
A340	Airbus	228
B747	Boeing	432

AVION

<u>id_avion</u>	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

<u>type</u>	constructeur	capacité
A340	Airbus	228
B747	Boeing	432

AVION

<u>id_avion</u>	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.

Schéma Relationnel : Contrainte d'intégrité de référence

- Lors de l'**insertion** d'un tuple dans **AVION**, le système doit vérifier que la valeur de l'attribut **type** existe bien dans **MODÈLE**
- Lors de la **suppression** d'un tuple dans **MODÈLE**, 4 types de contrôles peuvent être effectués :
 - interdire la suppression si la valeur de **type** existe dans **AVION** (recommandé)
 - supprimer ces valeurs
 - avertir l'utilisateur d'une incohérence
 - remplacer ces valeurs par la valeur nulle

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 $\rightarrow$ 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 → 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

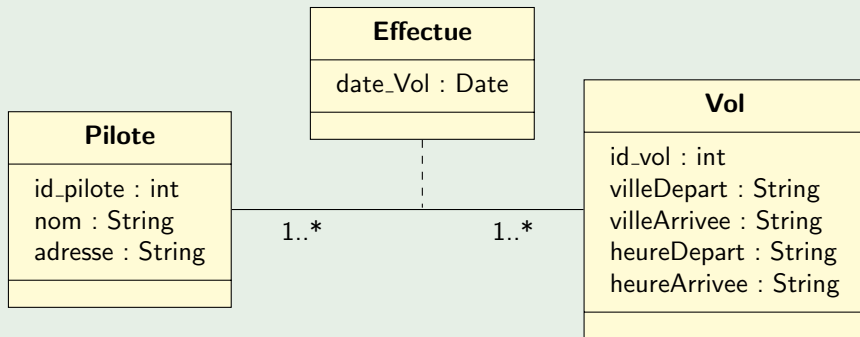


Diagramme de classes → Schéma relationnel : Exemple

Diagramme de classes

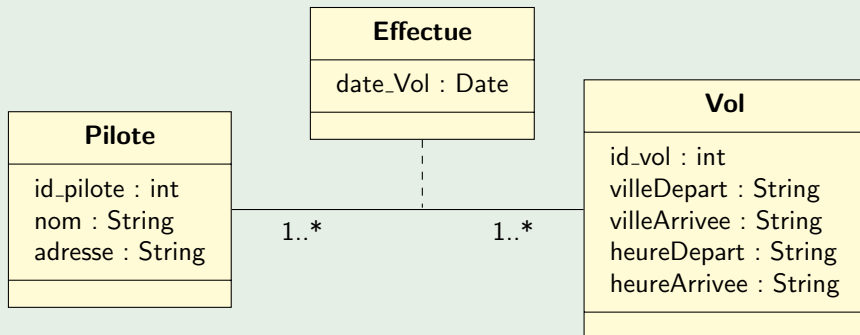


Schéma relationnel

PILOTE(id_pilote, nom, adresse)

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

EFFECTUE(id_pilote, id_vol, date_vol)

Normalisation

Normalisation

Objectifs

- Éviter :
 - la **redondance**, c'est-à-dire la duplication d'informations.
 - les **incohérences** par ajout.
 - la **perte d'informations** par suppression.
- S'assurer qu'un schéma relationnel est correct.

Dépendances Fonctionnelles

Définition

Soit $R(X, Y, Z)$ une relation.

Si à une valeur de X n'est associée qu'une seule valeur de Y , on dit que X **détermine** Y

Notation : $X \rightarrow Y$

On dit encore que Y **dépend fonctionnellement** de X .

1^{ère} Forme Normale

Une relation est en **1^{ère} forme normale** si et seulement si chaque attribut ne contient qu'une seule valeur à chaque instant (on n'affecte pas plusieurs valeurs à un seul attribut).

1^{ère} Forme Normale

Une relation est en 1^{ère} forme normale si et seulement si chaque attribut ne contient qu'une seule valeur à chaque instant (on n'affecte pas plusieurs valeurs à un seul attribut).

AVION

<u>id_avion</u>	type	capacité	compagnie
10	A340	228	Air France
20	B747	432	British Airways Qantas

1^{ère} Forme Normale

Une relation est en 1^{ère} forme normale si et seulement si chaque attribut ne contient qu'une seule valeur à chaque instant (on n'affecte pas plusieurs valeurs à un seul attribut).

AVION

<u>id_avion</u>	type	capacité	compagnie
10	A340	228	Air France
20	B747	432	British Airways Qantas

AVION n'est pas en 1^{ère} forme normale.

1^{ère} Forme Normale

Une relation est en **1^{ère} forme normale** si et seulement si chaque attribut ne contient qu'une seule valeur à chaque instant (on n'affecte pas plusieurs valeurs à un seul attribut).

AVION

<u>id_avion</u>	type	capacité	compagnie
10	A340	228	Air France
20	B747	432	British Airways Qantas

AVION n'est pas en 1^{ère} forme normale.

AVION1

<u>id_avion</u>	type	capacité	compagnie
10	A340	228	Air France
20	B747	432	British Airways
30	B747	432	Qantas

1^{ère} Forme Normale

Une relation est en **1^{ère} forme normale** si et seulement si chaque attribut ne contient qu'une seule valeur à chaque instant (on n'affecte pas plusieurs valeurs à un seul attribut).

AVION	<u>id_avion</u>	type	capacité	compagnie
	10	A340	228	Air France
	20	B747	432	British Airways Qantas

AVION n'est pas en 1^{ère} forme normale.

AVION1	<u>id_avion</u>	type	capacité	compagnie
	10	A340	228	Air France
	20	B747	432	British Airways
	30	B747	432	Qantas

AVION1 est en 1^{ère} forme normale.

2^{ème} Forme Normale

Une relation est en 2^{ème} forme normale si et seulement si :

- elle est en 1^{ère} forme normale
- tout attribut n'appartenant pas à la clé primaire ne dépend pas d'une partie de cette clé.

2^{ème} Forme Normale

Une relation est en 2^{ème} forme normale si et seulement si :

- elle est en 1^{ère} forme normale
- tout attribut n'appartenant pas à la clé primaire ne dépend pas d'une partie de cette clé.

AVION

<u>id_avion</u>	<u>constructeur</u>	<u>type</u>	<u>capacité</u>	<u>compagnie</u>
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways
30	Boeing	B747	432	Qantas

2^{ème} Forme Normale

Une relation est en 2^{ème} forme normale si et seulement si :

- elle est en 1^{ère} forme normale
- tout attribut n'appartenant pas à la clé primaire ne dépend pas d'une partie de cette clé.

AVION

<u>id_avion</u>	<u>constructeur</u>	<u>type</u>	<u>capacité</u>	<u>compagnie</u>
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways
30	Boeing	B747	432	Qantas

Dépendances fonctionnelles :

- (1) id_avion → type
- (2) id_avion → compagnie
- (3) type → capacité
- (4) type → constructeur

2^{ème} Forme Normale

Une relation est en 2^{ème} forme normale si et seulement si :

- elle est en 1^{ère} forme normale
- tout attribut n'appartenant pas à la clé primaire ne dépend pas d'une partie de cette clé.

AVION

<u>id_avion</u>	<u>constructeur</u>	<u>type</u>	<u>capacité</u>	<u>compagnie</u>
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways
30	Boeing	B747	432	Qantas

Dépendances fonctionnelles :

- (1) id_avion → type
- (2) id_avion → compagnie
- (3) type → capacité
- (4) type → constructeur

La relation AVION n'est pas (1 et 2) en 2^{ème} forme normale.

2^{ème} Forme Normale

AVION

<u>id_avion</u>	constructeur	type	capacité	compagnie
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways
30	Boeing	B747	432	Qantas

2^{ème} Forme Normale

AVION

<u>id_avion</u>	constructeur	type	capacité	compagnie
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways
30	Boeing	B747	432	Qantas

Dépendances fonctionnelles :

- (1) id_avion → type
- (2) id_avion → compagnie
- (3) type → capacité
- (4) type → constructeur

2^{ème} Forme Normale

AVION

<u>id_avion</u>	constructeur	type	capacité	compagnie
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways
30	Boeing	B747	432	Qantas

Dépendances fonctionnelles :

- (1) id_avion → type
- (2) id_avion → compagnie
- (3) type → capacité
- (4) type → constructeur

La relation AVION est en 2^{ème} forme normale.

2^{ème} Forme Normale

AVION

<u>id_avion</u>	constructeur	type	capacité	compagnie
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways
30	Boeing	B747	432	Qantas

Dépendances fonctionnelles :

- (1) id_avion → type
- (2) id_avion → compagnie
- (3) type → capacité
- (4) type → constructeur

La relation AVION **est** en 2^{ème} forme normale.

Remarque

la vérification de la deuxième forme normale n'a de sens que dans le cas où la clé primaire est constituée de plusieurs attributs.

3^{ème} Forme Normale

Une relation est en 3^{ème} forme normale si et seulement si :

- elle est en 2^{ème} forme normale et
- tout attribut n'appartenant pas à la clé ne dépend pas d'un attribut non clé.

3^{ème} Forme Normale

Une relation est en 3^{ème} forme normale si et seulement si :

- elle est en 2^{ème} forme normale et
- tout attribut n'appartenant pas à la clé ne dépend pas d'un attribut non clé.

AVION

<u>id_avion</u>	constructeur	type	capacité	compagnie
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways

3^{ème} Forme Normale

Une relation est en 3^{ème} forme normale si et seulement si :

- elle est en 2^{ème} forme normale et
- tout attribut n'appartenant pas à la clé ne dépend pas d'un attribut non clé.

AVION

<u>id_avion</u>	constructeur	type	capacité	compagnie
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways

Dépendances fonctionnelles :

- (1) id_avion → type
- (2) id_avion → compagnie
- (3) type → capacité
- (4) type → constructeur

3^{ème} Forme Normale

Une relation est en 3^{ème} forme normale si et seulement si :

- elle est en 2^{ème} forme normale et
- tout attribut n'appartenant pas à la clé ne dépend pas d'un attribut non clé.

AVION	<u>id_avion</u>	constructeur	type	capacité	compagnie
	10	Airbus	A340	228	Air France
	20	Boeing	B747	432	British Airways

Dépendances fonctionnelles :

- (1) id_avion → type
- (2) id_avion → compagnie
- (3) type → capacité
- (4) type → constructeur

La relation AVION n'est pas (3 et 4) en 3^{ème} forme normale.

3^{ème} Forme Normale

MODÈLE

<u>type</u>	constructeur	capacité
A340	Airbus	228
B747	Boeing	432

AVION

<u>id_avion</u>	type	compagnie
10	A340	Air France
20	B747	British Airways
30	B747	Qantas

3^{ème} Forme Normale

MODÈLE

<u>type</u>	constructeur	capacité
A340	Airbus	228
B747	Boeing	432

AVION

<u>id_avion</u>	type	compagnie
10	A340	Air France
20	B747	British Airways
30	B747	Qantas

Dépendances fonctionnelles :

id_avion → type,
id_avion → compagnie,
type → constructeur,
type → capacité

3^{ème} Forme Normale

MODÈLE

<u>type</u>	constructeur	capacité
A340	Airbus	228
B747	Boeing	432

AVION

<u>id_avion</u>	type	compagnie
10	A340	Air France
20	B747	British Airways
30	B747	Qantas

Dépendances fonctionnelles :

id_avion → type,
id_avion → compagnie,
type → constructeur,
type → capacité

MODELE et AVION sont en 3^{ème} forme normale.

Règles sur les Relations

Tuples et attributs

- ① chaque **tuple** d'une relation est **unique**
- ② l'ordre des tuples dans une relation n'a pas de signification
- ③ chaque **attribut** prend ses valeurs dans **un seul domaine**
- ④ un **domaine** peut correspondre à **plusieurs attributs**

Règles sur les Relations

Tuples et attributs

- 1 chaque **tuple** d'une relation est **unique**
- 2 l'ordre des tuples dans une relation n'a pas de signification
- 3 chaque **attribut** prend ses valeurs dans **un seul domaine**
- 4 un **domaine** peut correspondre à **plusieurs attributs**

Domaine

Un **domaine** est un **ensemble de valeurs** identifiées par un nom.

Définition du domaine :

- en intension (exemples : entier, réel, chaîne de caractères, ...)
- en extension (exemple : villes = {Paris, Marseille, Lyon})

Algèbre relationnelle

Algèbre Relationnelle

L'**algèbre relationnelle** utilise une collection d'opérateurs qui agissent sur des relations et produisent des relations comme résultat.

Algèbre Relationnelle

L'**algèbre relationnelle** utilise une collection d'opérateurs qui agissent sur des relations et produisent des relations comme résultat.

Opérateurs

- unaires : **projection**, **sélection**
- ensemblistes : **union**, **produit**, **différence**, **intersection**
- binaires (ou n-aires) : **jointure**, **division**

Projection

La **projection** d'une relation R sur un ensemble d'attributs $\{A_i\}$ de R est un sous-ensemble de R réduit aux sous-tuples ne contenant que les attributs de $\{A_i\}$.

Suppression dans les tuples de tous les attributs n'appartenant pas à $\{A_i\}$, puis suppression des doublons.

Notation : $\Pi_{A_1, \dots, A_n}(\text{nom-relation})$

Exemple

Quels sont les différents types d'avions ?

AVION	id_avion	constructeur	type	capacité	compagnie
	10	Airbus	A340	228	Air France
	20	Boeing	B747	432	British Airways
	30	Boeing	B747	432	Qantas

Exemple

Quels sont les différents types d'avions ?

AVION

id_avion	constructeur	type	capacité	compagnie
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways
30	Boeing	B747	432	Qantas

AVION

id_avion	constructeur	type	capacité	compagnie
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways
30	Boeing	B747	432	Qantas

Exemple

Quels sont les différents types d'avions ?

AVION	id_avion	constructeur	type	capacité	compagnie
	10	Airbus	A340	228	Air France
	20	Boeing	B747	432	British Airways
	30	Boeing	B747	432	Qantas

AVION	id_avion	constructeur	type	capacité	compagnie
	10	Airbus	A340	228	Air France
	20	Boeing	B747	432	British Airways
	30	Boeing	B747	432	Qantas

$\Pi_{\text{type}}(\text{AVION})$	type
	A340
	B747

Sélection

La **sélection** sur une relation R suivant une condition C portant sur des attributs de R est un sous-ensemble de R dont les tuples satisfont C .

Notation : $\sigma_C(R)$

Exemple

Quels avions peuvent accueillir au moins 300 passagers ?

AVION

id_avion	constructeur	type	capacité	compagnie
10	Airbus	A340	228	Air France
20	Boeing	B747	432	British Airways
30	Boeing	B747	432	Qantas

Exemple

Quels avions peuvent accueillir au moins 300 passagers ?

AVION	id_avion	constructeur	type	capacité	compagnie
	10	Airbus	A340	228	Air France
	20	Boeing	B747	432	British Airways
	30	Boeing	B747	432	Qantas

AVION	id_avion	constructeur	type	capacité	compagnie
	10	Airbus	A340	228	Air France
	20	Boeing	B747	432	British Airways
	30	Boeing	B747	432	Qantas

$$\sigma_{\text{capacité} \geq 300}(\text{AVION})$$

Union

L'**union** est une opération ensembliste portant sur 2 relations R_1 et R_2 **de même schéma**. $R_1 \cup R_2$ est la relation de même schéma contenant les tuples de R_1 et ceux de R_2 : Copie des tuples de R_1 et de R_2 , puis suppression des doublons.

Union

L'**union** est une opération ensembliste portant sur 2 relations R_1 et R_2 de **même schéma**. $R_1 \cup R_2$ est la relation de même schéma contenant les tuples de R_1 et ceux de R_2 : Copie des tuples de R_1 et de R_2 , puis suppression des doublons.

Quels sont tous les pilotes ?

P_AF

id	nom	adresse
1	Dupond	Nice
2	Smith	Londres
3	Garratt	Perth

P_BA

id	nom	adresse
3	Garratt	Perth
4	Durand	Paris
5	MacMachin	Glasgow

Union

L'**union** est une opération ensembliste portant sur 2 relations R_1 et R_2 de **même schéma**. $R_1 \cup R_2$ est la relation de même schéma contenant les tuples de R_1 et ceux de R_2 : Copie des tuples de R_1 et de R_2 , puis suppression des doublons.

Quels sont tous les pilotes ?

P_AF

id	nom	adresse
1	Dupond	Nice
2	Smith	Londres
3	Garratt	Perth

P_BA

id	nom	adresse
3	Garratt	Perth
4	Durand	Paris
5	MacMachin	Glasgow

P_AF $\cup$ P_BA

id	nom	adresse
1	Dupond	Nice
2	Smith	Londres
3	Garratt	Perth
4	Durand	Paris
5	MacMachin	Glasgow

Produit

Le **produit** est une opération ensembliste portant sur 2 relations R_1 et R_2 . $R_1 \times R_2$ est la relation ayant pour schéma la concaténation des schémas de R_1 et R_2 , et pour tuples toutes les combinaisons de tuples de R_1 et R_2 . L'opération *produit* n'est que très rarement utilisée.

Produit

Le **produit** est une opération ensembliste portant sur 2 relations R_1 et R_2 . $R_1 \times R_2$ est la relation ayant pour schéma la concaténation des schémas de R_1 et R_2 , et pour tuples toutes les combinaisons de tuples de R_1 et R_2 . L'opération *produit* n'est que très rarement utilisée.

 R_1

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

 R_2

D	E
d1	e1
d2	e2

Produit

Le **produit** est une opération ensembliste portant sur 2 relations R_1 et R_2 . $R_1 \times R_2$ est la relation ayant pour schéma la concaténation des schémas de R_1 et R_2 , et pour tuples toutes les combinaisons de tuples de R_1 et R_2 . L'opération *produit* n'est que très rarement utilisée.

 R_1

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

 R_2

D	E
d1	e1
d2	e2

 $R_1 \times R_2$

A	B	C	D	E
a1	b1	c1	d1	e1
a1	b1	c1	d2	e2
a2	b2	c2	d1	e1
a2	b2	c2	d2	e2
a3	b3	c3	d1	e1
a3	b3	c3	d2	e2

Différence

La **différence** est une opération ensembliste portant sur 2 relations R_1 et R_2 **de même schéma**. $R_1 \setminus R_2$ est la relation ayant le même schéma que R_1 et R_2 , et pour tuples ceux appartenant à R_1 et pas à R_2 .

Différence

La **différence** est une opération ensembliste portant sur 2 relations R_1 et R_2 **de même schéma**. $R_1 \setminus R_2$ est la relation ayant le même schéma que R_1 et R_2 , et pour tuples ceux appartenant à R_1 et pas à R_2 .

Quels sont les pilotes d'Air France ne travaillant pas pour British Airways ?

P_AF

id	nom	adresse
1	Dupond	Nice
2	Smith	Londres
3	Garratt	Perth

P_BA

id	nom	adresse
3	Garratt	Perth
4	Durand	Paris
5	MacMachin	Glasgow

Différence

La **différence** est une opération ensembliste portant sur 2 relations R_1 et R_2 **de même schéma**. $R_1 \setminus R_2$ est la relation ayant le même schéma que R_1 et R_2 , et pour tuples ceux appartenant à R_1 et pas à R_2 .

Quels sont les pilotes d'Air France ne travaillant pas pour British Airways ?

P_AF

id	nom	adresse
1	Dupond	Nice
2	Smith	Londres
3	Garratt	Perth

P_BA

id	nom	adresse
3	Garratt	Perth
4	Durand	Paris
5	MacMachin	Glasgow

$P_{AF} \setminus P_{BA}$

id	nom	adresse
1	Dupond	Nice
2	Smith	Londres

Intersection

L'**intersection** est une opération ensembliste portant sur 2 relations R_1 et R_2 de même schéma. $R_1 \cap R_2$ est la relation ayant le même schéma que R_1 et R_2 , et pour tuples ceux appartenant à la fois à R_1 et à R_2 .

Intersection

L'**intersection** est une opération ensembliste portant sur 2 relations R_1 et R_2 de même schéma. $R_1 \cap R_2$ est la relation ayant le même schéma que R_1 et R_2 , et pour tuples ceux appartenant à la fois à R_1 et à R_2 .

Quels sont les pilotes travaillant pour Air France et British Airways ?

P_AF

id	nom	adresse
1	Dupond	Nice
2	Smith	Londres
3	Garratt	Perth

P_BA

id	nom	adresse
3	Garratt	Perth
4	Durand	Paris
5	MacMachin	Glasgow

Intersection

L'**intersection** est une opération ensembliste portant sur 2 relations R_1 et R_2 de même schéma. $R_1 \cap R_2$ est la relation ayant le même schéma que R_1 et R_2 , et pour tuples ceux appartenant à la fois à R_1 et à R_2 .

Quels sont les pilotes travaillant pour Air France et British Airways ?

P_AF

id	nom	adresse
1	Dupond	Nice
2	Smith	Londres
3	Garratt	Perth

P_BA

id	nom	adresse
3	Garratt	Perth
4	Durand	Paris
5	MacMachin	Glasgow

$P_{AF} \cap P_{BA}$

id	nom	adresse
3	Garratt	Perth

Jointure

La **jointure** est une opération ensembliste portant sur 2 relations R_1 et R_2 . $R_1 \bowtie_{\text{attributs}} R_2$ est la relation dont les attributs sont ceux de R_1 et ceux de R_2 , et dont les tuples sont obtenus en composant un tuple de R_1 et un tuple de R_2 ayant la même valeur pour les attributs précisés.

Jointure

La **jointure** est une opération ensembliste portant sur 2 relations R_1 et R_2 . $R_1 \bowtie_{\text{attributs}} R_2$ est la relation dont les attributs sont ceux de R_1 et ceux de R_2 , et dont les tuples sont obtenus en composant un tuple de R_1 et un tuple de R_2 ayant la même valeur pour les attributs précisés.

AVION

id.	type	compagnie
10	A340	Air France
20	B747	British Airways
30	B747	Qantas

MODÈLE

type	constructeur	capa.
A340	Airbus	228
B747	Boeing	432

$\text{AVION} \bowtie_{\text{type}} \text{MODÈLE}$

id_avion	type	compagnie	constructeur	capa.
10	A340	Air France	Airbus	228
20	B747	British Airways	Boeing	432
30	B747	Qantas	Boeing	432

Jointure naturelle

La **jointure naturelle** est une opération ensembliste portant sur 2 relations R_1 et R_2 :

$R_1 \bowtie R_2$ est la relation dont les attributs sont ceux de R_1 et ceux de R_2 , et dont les tuples sont obtenus en composant un tuple de R_1 et un tuple de R_2 ayant la même valeur pour les attributs communs aux deux relations.

Jointure extérieure

La **jointure extérieure** est similaire à la jointure naturelle, mais conserve également les occurrences d'une relation qui n'ont pas de correspondant dans l'autre relation. Elle associe aux attributs non renseignés la valeur nulle.

Jointure extérieure

La **jointure extérieure** est similaire à la jointure naturelle, mais conserve également les occurrences d'une relation qui n'ont pas de correspondant dans l'autre relation. Elle associe aux attributs non renseignés la valeur nulle.

Exemple

AVION

id_avion	type	compagnie
20	B747	British Airways
30	B747	Qantas

MODÈLE

type	constructeur	capacité
A340	Airbus	228
B747	Boeing	432

AVION $\bowtie_{\text{ext}}$ MODÈLE

id_avion	type	compagnie	constructeur	capacité
20	B747	British Airways	Boeing	432
30	B747	Qantas	Boeing	432
⊥	A340	⊥	Airbus	228

Combinaisons d'Opérations

Quelle est la capacité des avions de Qantas ?

AVION

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

MODÈLE

type	constructeur	capacité
A340	Airbus	228
B747	Boeing	432

Combinaisons d'Opérations

Quelle est la capacité des avions de Qantas ?

r_1

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

MODÈLE

type	constructeur	capacité
A340	Airbus	228
B747	Boeing	432

$$r_1 = \sigma_{\text{compagnie}=\text{Qantas}}(\text{AVION}),$$

Combinaisons d'Opérations

Quelle est la capacité des avions de Qantas ?

r_2

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

MODÈLE

type	constructeur	capacité
A340	Airbus	228
B747	Boeing	432

$$r_1 = \sigma_{\text{compagnie}=\text{Qantas}}(\text{AVION}), \quad r_2 = \Pi_{\text{type}} r_1$$

Combinaisons d'Opérations

Quelle est la capacité des avions de Qantas ?

r_2

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

MODÈLE

type	constructeur	capacité
A340	Airbus	228
B747	Boeing	432

r_3

type	constructeur	capacité
B747	Boeing	432

$$r_1 = \sigma_{\text{compagnie}=\text{Qantas}}(\text{AVION}), \quad r_2 = \Pi_{\text{type}} r_1$$

$$r_3 = r_2 \bowtie \text{MODÈLE},$$

Combinaisons d'Opérations

Quelle est la capacité des avions de Qantas ?

r_2

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

MODÈLE

type	constructeur	capacité
A340	Airbus	228
B747	Boeing	432

r_4

type	constructeur	capacité
B747	Boeing	432

$$r_1 = \sigma_{\text{compagnie}=\text{Qantas}}(\text{AVION}), \quad r_2 = \Pi_{\text{type}} r_1$$

$$r_3 = r_2 \bowtie \text{MODÈLE}, \quad r_4 = \Pi_{\text{capacité}}(r_3)$$

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).

familles d'opérations

3 familles d'opérations

- **LDD** = **Langage de Définition des Données** : description de la structure de la base de données (tables, attributs)
- **LMD** = **Langage de Manipulation de Données** : manipulation des tables
- **LCD** = **Langage de Contrôle des Données** : gestion du contrôle et de la sécurité de la base de données

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)  
);
```

Dépendances

Les clauses **ON UPDATE** et **ON DELETE** permettent d'indiquer les répercussions à effectuer lors de mises à jour ou de suppressions de tables ou de clés primaires :

- **NO ACTION** : les clauses `update` et `delete` ne sont pas exécutées pour que l'intégrité référentielle soit préservée
- **CASCADE** : toutes les clés étrangères sont mises à jour lors de la modification de la clé primaire et tous les enregistrements ayant la clé étrangère sont supprimés lors de la suppression de la clé primaire
- **SET NULL, SET DEFAULT** : la clé étrangère prend la valeur `NULL` (ou la valeur par défaut) lors de la modification ou la suppression de la clé primaire

Exemple

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

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.

Contrairement à ce qui se passe en algèbre relationnelle, 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.

Contrairement à ce qui se passe en algèbre relationnelle, 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 Doubleton

 $\Pi_{type} Avion$

```
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

Quels sont les noms des pilotes qui ont assuré le vol Londres–Paris 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–Paris 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='Paris'
        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

Jointures Extérieures

Clauses **LEFT OUTER**, **RIGHT OUTER** et **FULL OUTER**. On conserve les tuples qui ne vérifient pas la condition pour l'une ou l'autre table.

Exemple

Quels sont les pilotes n'effectuant pas de vol ?

On sélectionne tous les pilotes et leurs vols même s'ils n'ont pas effectué de vols :

```
select Pilote.nom, Effectue.id_vol from Pilote
left outer join Effectue
on Pilote.id_pilote=Effectue.id_pilote;
```

nom	id_vol
Dupond	AF1232
Garratt	QT17
Smith	QT17
Garratt	BA302
MacMachin	BA302
Durand	

Exemple (suite)

Quels sont les pilotes n'effectuant pas de vol ?

On ne garde ensuite que les pilotes associés à aucun vol :

```
select Pilote.nom from Pilote left outer join Effectue
      on Pilote.id_pilote=Effectue.id_pilote
      where id_vol is null;
```

nom

Durand

Auto-Jointure

Permet de traiter 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 traiter 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');
```

Fonctions sur les Attributs

Fonctions portant sur les chaînes de caractères :

- **CHAR_LENGTH**(chaîne) : renvoie la longueur de chaîne.
- **POSITION**(chaîne IN source) : cherche la chaîne de caractères dans la chaîne source. Si elle est trouvée, sa position est retournée, sinon 0 est renvoyé.
- **SUBSTRING**(source FROM début FOR longueur) : extrait la sous-chaîne de source commençant à la position début et ayant longueur caractères.
- **UPPER**(chaîne) : convertit la chaîne en majuscules.
- **LOWER**(chaîne) : convertit la chaîne en minuscules.
- **EXTRACT**(élément FROM source) : extrait un élément d'une chaîne source telle que date ou heure.
- chaîne1 || chaîne2 : concaténation des deux chaînes.

Exemple

```
select Avion.compagnie ,  
       position('an' in Avion.compagnie),  
       char_length(Avion.compagnie),  
       substring(Avion.compagnie from 5 for 3)  
from Avion;
```

compagnie	position	char_length	substring
Qantas	2	6	as
Air France	7	10	Fra
Air France	7	10	Fra
British Airways	0	15	ish

Exemple

```
select Pilote.nom, Pilote.prenom,  
       upper(nom)||' '||lower(prenom) as nom_prenom  
from Pilote  
order by nom_prenom asc;
```

nom	prenom	nom_prenom
Dupond	Laurent	DUPOND laurent
Durand	Frédéric	DURAND frédéric
Garratt	Jo	GARRATT jo
MacMachin	Allan	MACMACHIN allan
Smith	Wendy	SMITH wendy

Autres fonctions

De nombreuses fonctions sont disponibles. Leur liste peut-être obtenue (dans psql) en tapant `\df`.

Exemple

```
select Modele.type,sqrt(float8(capacite))
      from Modele where Modele.capacite is not null;
```

type		sqrt
-----+-----		
A340		15.0996688705415
B747		20.784609690826528
A320		12.24744871391589
B707		13.74772708486752
B737		14.491376746189438

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
```

```
-----  
241.800000000000000000
```

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

Vues

Une vue est le nom donné à une requête.

L'utilisation de **vues** permet de donner de la base de données une vision adaptée à l'utilisateur (en évitant par exemple de faire apparaître des données sensibles)

Une vue est **dynamique** : c'est une sorte de table virtuelle. Une fois créée, la vue est accessible comme toute autre table.

Création d'une Vue

La création d'une vue s'effectue à l'aide de la clause **CREATE VIEW**.

Création d'une Vue

La création d'une vue s'effectue à l'aide de la clause **CREATE VIEW**.

Créer une vue de nom AF2SG des avions de la compagnie Air France ayant Singapour pour destination.

```
create view AF2SG as
  select avion.id_avion, avion.type, vol.id_vol
     from avion natural join effectue natural join vol
  where avion.compagnie='Air France'
        and vol.ville_arrivee='Singapour';
```

Utilisation d'une vue

Les **requêtes** s'effectuent comme sur les tables :

Utilisation d'une vue

Les **requêtes** s'effectuent comme sur les tables :

```
select * from AF2SG;
```

id_avion	type	id_vol
4	A320	AF1232

Utilisation d'une vue

Les **requêtes** s'effectuent comme sur les tables :

```
select * from AF2SG;
```

id_avion	type	id_vol
4	A320	AF1232

Il ne peut y avoir **ni insertion ni suppression** directes, car la vue est définie par une requête. Ce n'est donc pas une table à part entière.

Pour contourner cette difficulté, on peut utiliser une **règle** (voir plus loin)

Suppression d'une Vue

```
DROP VIEW name [ CASCADE | RESTRICT ]
```

RESTRICT : si la vue intervient dans la définition d'une autre vue ou dans une contrainte d'intégrité, la commande est rejetée.

CASCADE : si la vue est supprimée, toutes les vues et contraintes où la vue intervient seront supprimées.

Suppression d'une Vue

```
DROP VIEW name [ CASCADE | RESTRICT ]
```

RESTRICT : si la vue intervient dans la définition d'une autre vue ou dans une contrainte d'intégrité, la commande est rejetée.

CASCADE : si la vue est supprimée, toutes les vues et contraintes où la vue intervient seront supprimées.

```
drop view AF2SG;
```

Règles

Les **règles** permettent de spécifier quelle **action** doit être effectuée lors de la **réception d'un événement** donné.

Syntaxe

```
CREATE RULE nom_règle AS ON événement TO objet DO action
```

L'**objet** peut être une table ou une vue éventuellement assortie d'une condition dans une clause **WHERE**.

L'événement est une action sur l'objet comme une insertion, une suppression ou une mise à jour. L'**action** peut être :

- une ou plusieurs requêtes
- ne rien faire : **NOTHING**
- ce qu'il faut faire à la place : **INSTEAD action**

Règles — Exemple

Lors de l'insertion dans la vue AF2SG, créer les tuples idoines dans avion et vol

```
create rule ins_af2sg as on insert to AF2SG
do instead (
    insert into Avion
        values(new.id_avion,new.type,'Air France');
    insert into Vol(id_vol,ville_arrivee)
        values(new.id_vol,'Paris');
```

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 **recupérer** 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()
```