Introduction aux bases de données

A. OMOR

Qu'est ce que c'est qu'une BD?

- Un Système de Gestion de Bases de Données est:
  - Un ensemble de données liées entre elles
  - Un ensemble de programmes pour accéder les données
  - Un environnement qui est *pratique* et *efficace* à utiliser

- Applications utilisant des bases de données:
  - Gestion du personnel, étudiants, cours, inscriptions, ...
  - Système de réservation de places d'avion, de train, …
  - Gestion des comptes clients des banques,
  - Gestion des lignes de bus de la ville de Lausanne, cartographie
  - E-commerce,
  - Gestion de production, vente des entreprises
  - ...
- Exemple : votre répertoire de téléphone portable est une base de données

Base de données (BD)

- BD = Ensemble de données reliées entre elles et utilisables avec un programme particulier appelé : Système de Gestion de Bases de Données

- Ensemble de données :
  - Structuré
  - Cohérent
    - L'ensemble des données a une signification (pas n'importe quelles données)
  - Partagé
    - Utilisées par plusieurs utilisateurs et/ou types d'utilisateurs
  - Défini pour les besoins d'une application
    - Univers du discours

Système de Gestion de Bases de Données (SGBD)

- SGBD: ensemble de programmes i.e. software permettant de :
  - Définir la BD
    - Spécifier les types de données, la structure, contraintes…
      - ex: structure de Etudiant, de ses champs,…
  - Construire la BD:
    - Stocker les données sur disque
  - Manipuler la BD
    - Récupérer des données stockées (requêtes sur la BD)
      - ex: liste des étudiants
    - Mettre à jour les données
      - ex: changer l'heure de début d'un cours
  - Maintenir/administrer la BD
    - Gestion des données (concurrence, fiabilité,…) et des utilisateurs (droits)

Pourquoi des bases de données?

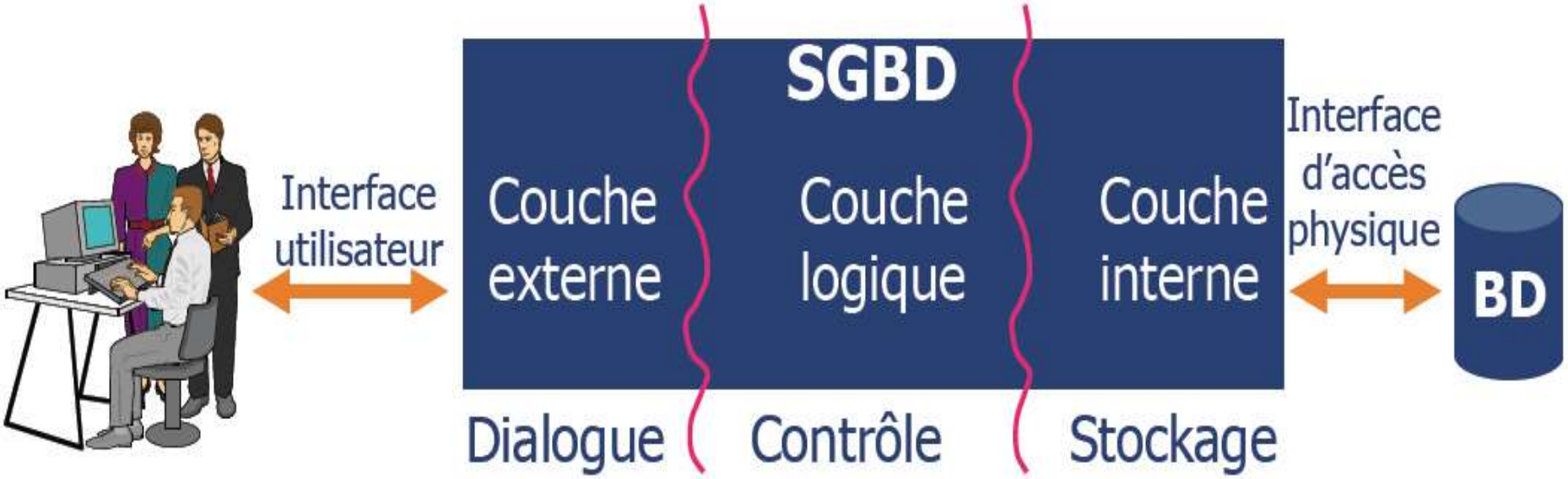
- Ancêtre des SGBDs: Systèmes de fichiers.
- Inconvénients des systèmes de fichiers:
  - Redondance des données et donc inconsistance des données
    - Formats de fichiers multiples et différents, duplication des données dans plusieurs fichiers
  - Difficulté d'accès aux données
    - Nécessaire d'écrire un nouveau programme pour chaque accès différent aux données
  - Pas de gestion de la concurrence
    - Accès concurrent par plusieurs utilisateurs nécessaire pour des raisons de performance
    - Doit être contrôlé pour éviter des erreurs dans les données
      - Exemple: deux personnes mettant à jour le même compte avec un solde différent
  - Pas de gestion de la sécurité "pratique"
    - Difficile de donner des droits aux utilisateurs sur certaines données des fichiers et pas à d'autres.
- Les bases de données offrent des solutions à tous ces problèmes.

Objectif fondamental du SGBD

- Indépendance Programme/données:
  - Indépendance entre données sur disque et utilisateurs (programmes, humains) des données
    - Possibilité pour un administrateur système de modifier ses choix en matière d'organisation des données, pour améliorer les performances, sans que cela ait un impact sur les utilisateurs (leurs requêtes d'interrogation ou de mise à jour, ou leurs programmes d'application qui utilisent la base de données).
    - Possibilité pour un utilisateur de modifier sa vue de la base et ses traitements sans avoir à se soucier des choix qui ont été opérés au niveau interne en matière de fichiers

-> Vision plus fine de l'architecture

Trois couches

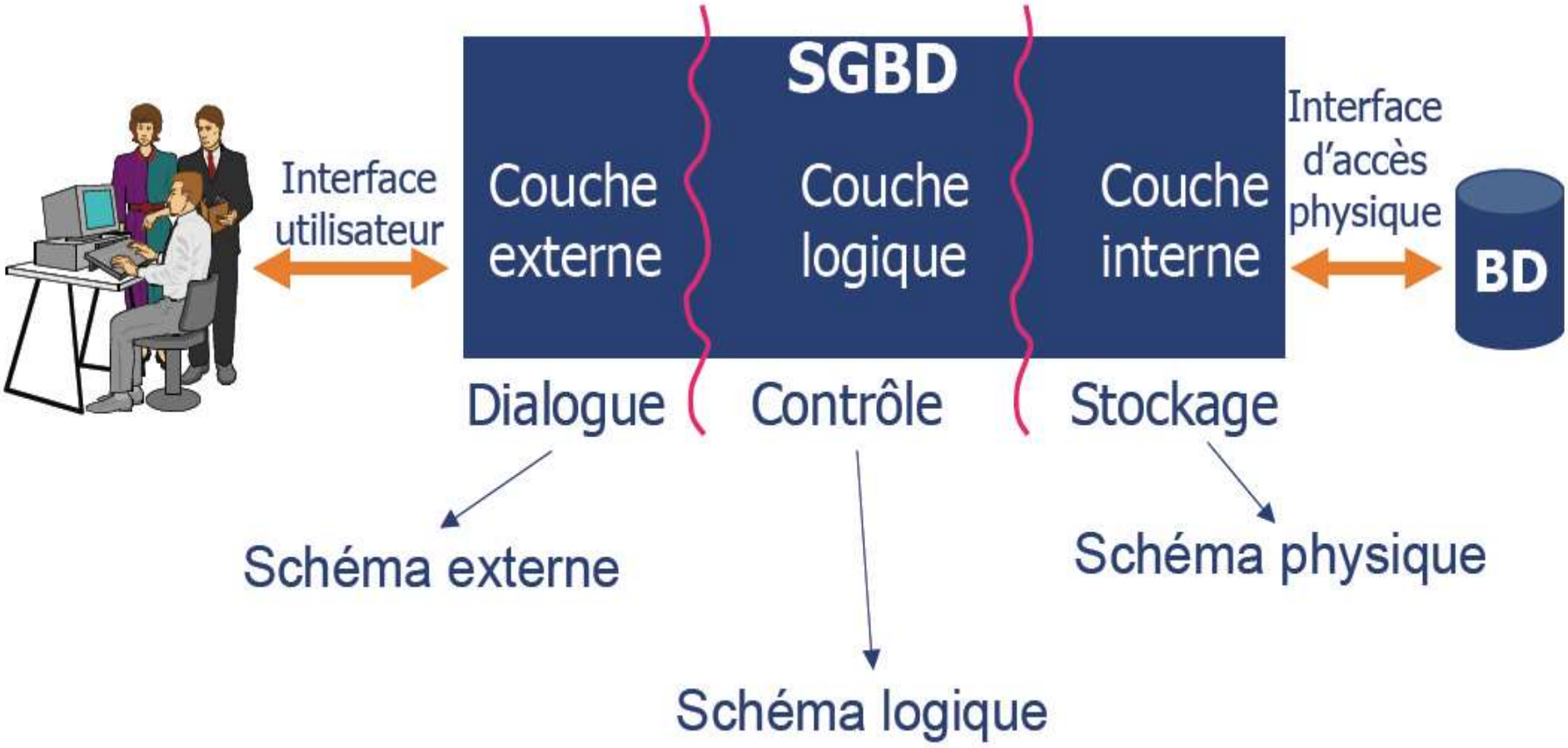


- Trois couches avec des fonctions différentes

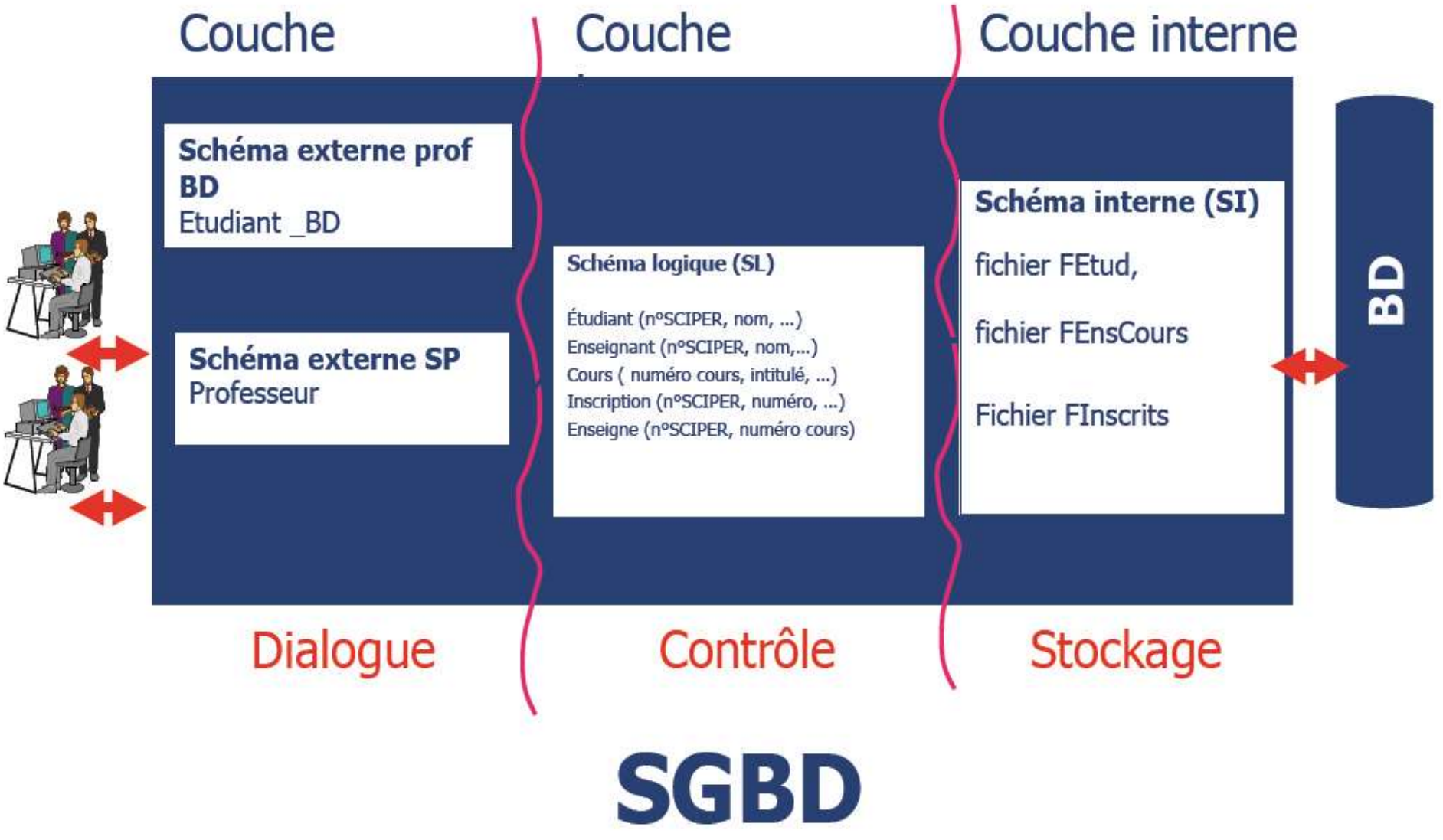
Pour chaque couche

- Modèle de données
  - Ensemble des concepts qui permettent de décrire les données d'une base et les règles d'utilisation de ces concepts.
- Schéma d'une BD
  - Description d'une base de données obtenues en employant un modèle de données.

Modèles


SGBD
Interface utilisateur
Couche externe
Couche logique
Couche interne
Interface d'accès physique
BD
Dialogue
Contrôle
Stockage
Schéma externe
Schéma logique
Schéma physique

Résumé: 3 types de schémas



Cycle de vie d'une base de données

Monde réel

Personne
Voiture

Concepteur

❶ ❶

❹

Schéma conceptuel

❷

Schéma logique

Table Personne
Table Voiture
Table Conduit
...

Schéma physique

❷

BD

Personne
Dupont
Durant
Rochat

Voiture
Ford KA
Peugeot 106
Renault Clio
...

❸

LMD

utilisateurs

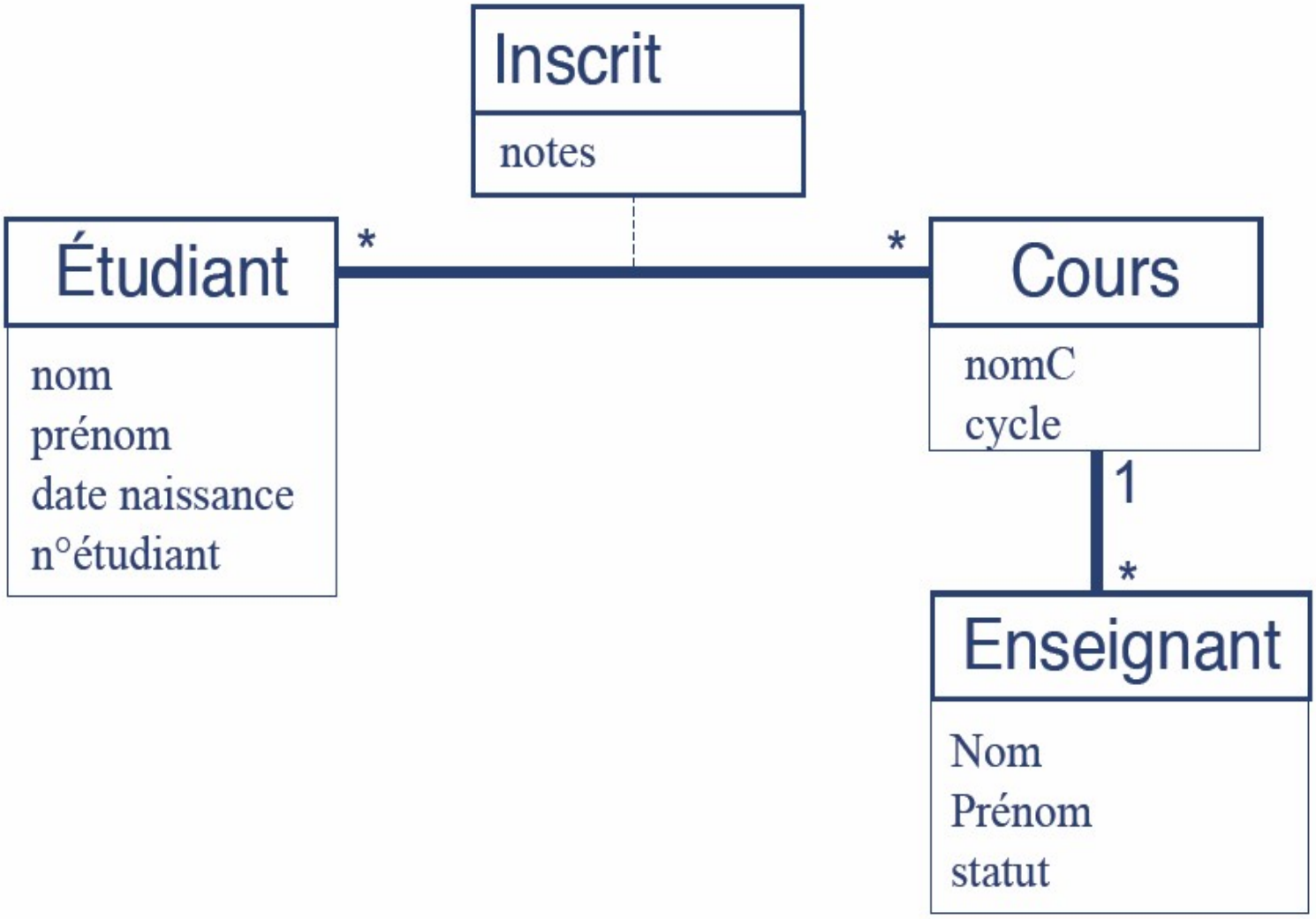
Cycle de vie d'une base de données

- 4 phases:
  1) Conception de la base
     -> Schéma Conceptuel
  2) Implantation des données
     -> Schéma Logique, schéma physique
     -> Population de la BD
  3) Utilisation
     -> Interrogation
     -> Développement des programmes d'application
     -> Mises à jour
  4) Maintenance (correction, évolution)

Phase 1: Conception

- Phase de réflexion et d'analyse sur la manière de structurer les données en fonction des besoins de l'application
- Objectif: Déterminer et décrire le futur contenu de la BD:
  - Quelles sont les infos importantes pour l'application?
  - Quelles sont leurs propriétés ?
- Nécessite un accord des utilisateurs sur la nature et les caractéristiques des informations
- Résultat: **SCHEMA CONCEPTUEL**

Exemple Schéma conceptuel en UML



Phase 2: Implantation

- Transmission de la description des données au SGBD choisi
  - Traduction du schéma conceptuel en schéma logique puis schéma physique
- Au moyen d'un langage de description de données (LDD) spécifique du SGBD choisi
- Insertion des données

Exemple Schéma logique

- Table Etudiant (nom, prenom, datenaissance, n°etudiant)
- Table Cours(nomC, cycle, nomEnseignant)
- Table Inscrit(nomEtudiant, nomC)
- Table Enseignant(nom, prénom, statut)

Phase 3: Utilisation

- Définition des schémas externes
- Rédaction des requêtes d'interrogation
  - Paul est il inscrit au cours de BD?
  - Quelle est la date de naissance de Paul ?
- Rédaction des requêtes de mise à jour
  - ajouter de nouvelles informations,
  - supprimer des informations périmées,
  - modifier le contenu des informations
- Au moyen d'un langage de manipulation de données (LMD)

Phase 4: Maintenance

- Corrective
  - Supprimer les données erronées, redondantes...
- Évolutive
  - Faire évoluer la structure des données
  - ex: Ajouter la colonne "adresse" à Étudiant

SQL :
Structured Query Language

SQL: Trois langages

- Langage de définition de données (LDD/DDL)
  - création de relations : CREATE TABLE
  - modification de relations: ALTER TABLE
  - suppression de relations: DROP TABLE
  - vues, index …. : CREATE VIEW ...
- Langage de manipulation de données (LMD /DML)
  - insertion de tuples: INSERT
  - mise à jour des tuples: UPDATE
  - suppression de tuples: DELETE
- Langage de requêtes (LMD/DML)
  - SELECT ……. FROM ..... WHERE .....

Structure générale d'une requête : le BLOC

- Structure d'une requête formée de trois clauses:

  SELECT <liste-attributs>
  FROM <liste-tables>
  WHERE <condition>

- SELECT définit le format du résultat cherché
- FROM définit à partir de quelles tables le résultat est calculé
- WHERE définit les prédicats de sélection du résultat

Exemple de requête

SELECT * FROM Pays;

-> tous les attributs de tous les tuples dans la table Pays

PAYS

nom	capitale	population	surface
Irlande	Dublin	3	70
Autriche	Vienne	8	83
Royaume-Uni	Londres	36	244
Suisse	Berne	7	41
USA	Washington	189	441

Sélection

```
SELECT *
FROM R
WHERE condition
```

Exemple de requête de sélection

```
SELECT      *
FROM        Pays
WHERE       population < 20 ;
```

PAYS

nom	capitale	population	surface
Irlande	Dublin	3	70
Autriche	Vienne	8	83
Royaume-Uni	Londres	36	244
Suisse	Berne	7	41
USA	Washington	189	441

* = toutes les colonnes

Projection

SELECT A1, A2,…An
FROM R

Exemple de requête de projection

```
SELECT   nom, capitale
FROM     Pays;
```

PAYS			
nom	capitale	population	surface
Irlande	Dublin	3	70
Autriche	Vienne	8	83
Royaume-Uni	Londres	36	244
Suisse	Berne	7	41
USA	Washington	189	441

Projection - selection

SELECT <liste-attributs> ← projection

FROM <liste-tables>

WHERE <condition> ← selection

Requête de sélection + projection

```
SELECT  nom, capitale, population
FROM Pays
WHERE  population < 20 ;
```

PAYS

nom	capitale	population	surface
Irlande	Dublin	3	70
Autriche	Vienne	8	83
Royaume-Uni	Londres	36	244
Suisse	Berne	7	41
USA	Washington	189	441

résultat

nom	capitale	population
Irlande	Dublin	3
Autriche	Vienne	8
Suisse	Berne	7

Produit cartésien

```
SELECT *
FROM R, S
```

Exemple

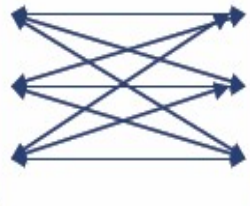
SELECT * FROM R, S

R

A	B
a	b
b	c
c	b

S

C	D	E
c	d	e
b	a	b
a	a	c



A	B	C	D	E
a	b	c	d	e
a	b	b	a	b
a	b	a	a	c
b	c	c	d	e
b	c	b	a	b
b	c	a	a	c
c	b	c	d	e
c	b	b	a	b
c	b	a	a	c

n tuples

m tuples

n x m tuples

Exemple II

JO

année	lieu	pays
1896	Athènes	Grèce
1900	Paris	France
1904	Louis	USA
1908	Londres	RU

PAYS

nom	capitale	population	surface
Irlande	Dublin	3	70
Autriche	Vienne	8	83
RU	Londres	36	244
Suisse	Berne	7	41
USA	Washingtor	189	441

```
SELECT année, lieu, pays, capitale
FROM JO, PAYS
```

Jointure

- En SQL :

```
SELECT *
FROM R, S
WHERE R.A1 = S.A1
  AND R.A2 = S.A2
        …
  AND R.An = S.An
```

- Avec A1, …, An tous les attributs communs à R et S

Jointure de 2 relations

JO

année	lieu	pays
1896	Athènes	Grèce
1900	Paris	France
1904	St.Louis	USA
1908	Londres	Royaume-Uni

```
SELECT année, lieu, pays,
       capitale
FROM JO, PAYS
WHERE JO.pays = PAYS.nom;
```

PAYS

nom	capitale	population	surface
Irlande	Dublin	3	70
Autriche	Vienne	8	83
Royaume-Uni	Londres	36	244
Suisse	Berne	7	41
USA	Washington	189	441

Jointure de 2 relations: résultat

JO

année	lieu	pays
1896	Athènes	Grèce
1900	Paris	France
1904	St Louis	USA
1908	Londres	Royaume-Uni

PAYS

nom	capitale	population	surface
Irlande	Dublin	3	70
Autriche	Vienne	8	83
Royaume-Uni	Londres	36	244
Suisse	Berne	7	41
USA	Washington	189	441

Résultat

année	lieu	pays	capitale
1904	St Louis	USA	Washington
1908	Londres	Royaume-Uni	Londres

```
SELECT année, lieu, pays, capitale
FROM JO, PAYS
WHERE JO.pays = PAYS.nom ;
```

Union

BlocR **UNION** BlocS

```
SELECT *
FROM R
UNION
SELECT *
FROM S
```

Les tuples en double sont éliminés du résultat

Intersection

BlocR **INTERSECT** BlocS

```
SELECT *
FROM R
INTERSECT
SELECT *
FROM S
```

Les tuples en double sont éliminés du résultat

Différence

SQL : MINUS
Manifold: EXCEPT

BlocR **EXCEPT** BlocS

```
SELECT *
FROM R
 EXCEPT
SELECT *
FROM S
```

Les tuples en double sont éliminés du résultat

Remarques

- Si plusieurs attributs ont le même nom, pour résoudre l'ambiguïté, on spécifie la relation auquel l'attribut appartient.
  - Ex : select A, R.B, C
    from R, S

Conclusion

- Structure d'une requête formée de trois clauses:

  SELECT <liste-attributs>
  FROM <liste-tables>
  WHERE <condition>

- SELECT définit le format du résultat cherché
- FROM définit à partir de quelles tables le résultat est calculé
- **WHERE** définit les prédicats de sélection du résultat

- Requête peut être UNION, DIFFERENCE ou INTERSECTION de deux requêtes

Ecriture des conditions

Opérateurs de comparaison

- = égal
- <> différent
- > plus grand que
- >= plus grand ou égal
- < plus petit que
- <= plus petit ou égal

- WHERE surface = 200
- WHERE capitale <> 'Paris'
- WHERE population > 8
- WHERE population >= 8
- WHERE surface < 83
- WHERE surface <= 83

Opérateurs logiques

- Tous les prédicats : AND
  ex: *WHERE population<10 AND surface<500*

- Un des prédicats : OR
  ex: *WHERE population<10 OR surface<500*

- Négation de la condition : NOT
  ex:
```
SELECT      P1.nom, P2.nom, P1.capitale
FROM        PAYS P1, PAYS P2
WHERE       P1.capitale = P2.capitale
AND NOT P1.nom = P2.nom ;
```

Expressions logiques

- Combinaisons:

WHERE (ensoleillement > 80% **AND** pluviosité < 200)
OR température > 30

WHERE ensoleillement > 80% **AND**
(pluviosité < 200 **OR** température > 30)

Appartenance à un ensemble: IN

- WHERE monnaie = 'Pound' **OR**
  monnaie = 'Schilling' **OR**
  monnaie = 'Franc'
- Équivalent à:

  WHERE monnaie **IN** ('Pound', 'Schilling', 'Franc')
- ***NOT IN****: non appartenance à un ensemble*

Comparaison à un ensemble: ALL

- SELECT * FROM Employee
  WHERE salary >= 1400
  AND salary >= 3000;
- Équivalent à:

```
SELECT * FROM Employee
WHERE salary >= ALL ( 1400, 3000);
```

Valeur dans un intervalle: BETWEEN

- WHERE population >= 50 **AND**
  population <= 60

- Équivalent à:

  WHERE population **BETWEEN** 50 **AND** 60

NOT BETWEEN

Conditions partielles (wildcards)

% : zéro ou n'importe quel caractère (x caractères)

- WHERE pays LIKE '%lande'
  - -> Irlande, Islande, Finlande, Hollande
- WHERE pays LIKE '%ran%'
  - -> Iran, France

_ : exactement un caractère

- WHERE pays LIKE 'I_lande'
  - -> Irlande, Islande

NOT LIKE

Valeurs calculées

- SELECT nom, population, surface, natalité
  FROM PAYS
  WHERE (population * 1000 / surface) < 50
  AND (population * natalité / surface) > 0

- SELECT nom, (population * 1000 / surface)
  FROM PAYS

Valeurs numériques: + - * /
Chaines de caractères: ¦ ¦

‘NULL’ <> 0 !!!

- Null = valeur inconnue ou non définie:

Pays (nom , population)

```
SELECT  nom
FROM    PAYS
WHERE  population IS NULL
```

- -> pays dont on a pas entré la valeur de population

NULL = Pas de valeur

Requêtes avec blocs emboîtés

BD Exemple : Livraisons

P (np , nomp , couleur , poids , prix) *les produits*

U (nu , nomu , ville , pays) *les usines*

F (nf , nomf , type , ville , pays) *les fournisseurs*

PUF (np, nu, nf , quantité) *les livraisons*

np, nu, nf dans PUF sont des identifiants externes sur P, U et F (respectivement)

Jointure

Nom et couleur des produits livrés par le fournisseur 1

Solution 1 : la jointure déclarative

```
SELECT nomp, couleur FROM  P, PUF
WHERE PUF.np = P.np AND nf = 1 ;
```

Jointure par blocs emboîtés : IN (∈)

Nom et couleur des produits livrés par le fournisseur 1

Solution 1 : la jointure déclarative

```
SELECT nomp, couleur FROM  P, PUF
WHERE PUF.np = P.np AND nf = 1 ;
```

Solution 2 : la jointure procédurale (par emboîtement)

Outer query

```
SELECT nomp, couleur
FROM P
WHERE np  IN
```

Nested query

```
( SELECT np
FROM  PUF
WHERE nf = 1) ;
```

1. Ensemble des produits livrés par le fournisseur 1

2. Nom et couleur des produits du fournisseur 1

Jointure par blocs emboîtés : IN (∈)

- SELECT nomp, couleur FROM P
  WHERE np **IN**
  (SELECT np
  FROM PUF
  WHERE nf = 1) ;
- **IN** compare chaque valeur de np avec l'ensemble (ou multi-ensemble) de valeurs retourné par la sous-requête
- **IN** peut aussi comparer un tuple de valeurs:
  - SELECT nu FROM U
    WHERE (ville, pays) IN (SELECT ville, pays FROM F)

Composition de conditions

Nom des fournisseurs qui approvisionnent une usine de Londres ou de Paris en un produit rouge

```
SELECT nomf FROM F
WHERE nf  IN
    (SELECT nf  FROM PUF
     WHERE np  IN
         (SELECT np FROM P
         WHERE couleur = 'rouge')
       AND  nu  IN
         (SELECT nu FROM U
         WHERE ville = 'Londres'
                    OR ville = 'Paris') ) ;
```

```
SELECT nomf
  FROM  PUF, P, F, U
  WHERE couleur = 'rouge'
      AND PUF.np = P.np
      AND PUF.nf = F.nf
      AND PUF.nu = U.nu
      AND  (U.ville = 'Londres'
            OR U.ville = 'Paris');
```

Traitement des résultats

Fonctions sur des colonnes

- Attributs calculés
  - Ex: SELECT nom, population*1000/surface FROM PAYS
- Opérateurs sur attributs numériques
  - SUM: somme des valeurs des tuples sélectionnés
  - AVG: moyenne
- Opérateurs sur tous types d'attributs
  - MIN: minimum
  - MAX: maximum
  - COUNT: nombre de tuples sélectionnés

PAYS				
nom	capitale	population	surface	continent
Irlande	Dublin	3	70	Europe
Autriche	Vienne	8	83	Europe
R-Uni	Londres	36	244	Europe
Suisse	Berne	7	41	Europe
USA	Washingtor	189	441	Amerique

```
SELECT  MIN(population),
        MAX(population),
        AVG(population),
        SUM(surface),
        COUNT(*)
FROM PAYS
WHERE continent = 'Europe'
```

Retourne un tuple avec:

- la pop. du plus petit pays d'Europe,
- la pop. du plus grand,
- la moyenne des pop. de tous les pays d'Europe,
- la somme des surfaces des pays d'Europe
- le nombre de pays d'Europe

DISTINCT

PAYS

nom	capitale	population	surface	continent
Irlande	Dublin	3	70	Europe
Autriche	Vienne	8	83	Europe
R-Uni	Londres	36	244	Europe
Suisse	Berne	7	41	Europe
USA	Washington	189	441	Amerique

Suppression des doubles:

SELECT **DISTINCT** continent

FROM PAYS

Europe
Amerique

ORDER BY

PAYS

nom	capitale	population	surface	continent
Irlande	Dublin	3	70	Europe
Autriche	Vienne	8	83	Europe
R-Uni	Londres	36	244	Europe
Suisse	Berne	7	41	Europe
USA	Washington	189	441	Amerique

Tri des tuples du résultat

```
SELECT continent, population, nom
FROM PAYS
WHERE surface > 60
ORDER BY continent, nom DESC
```

ASC/DESC

résultat

continent	population	nom
Europe	36	Royaume-Uni
Europe	3	Irlande
Europe	8	Autriche
Amerique	189	USA

GROUP BY

PAYS				
nom	capitale	population	surface	continent
Irlande	Dublin	3	70	Europe
Autriche	Vienne	8	83	Europe
Royaume-Uni	Londres	36	244	Europe
Suisse	Berne	7	41	Europe
USA	Washington	189	441	Amerique

Partition de l'ensemble des tuples en groupes homogènes

```
SELECT    continent, MIN(population), MAX(population),
                AVG(population), SUM(surface), COUNT(*)
FROM  PAYS
GROUP BY continent ;
```

résultat

continent	MIN(pop)	MAX(pop)	AVG(pop)	SUM(surf)	COUNT
Europe	3	36	13,5	438	4
Amerique	189	189	189	441	1

un tuple par continent