2.4 Contraintes d'intégrité

La cohérence des données au sein d'une BD est assurée par des contraintes d'intégrité. Ces dernières sont des invariants, c'est à dire des propriétés logiques que les données doivent vérifier à tout instant. parmi ces contraintes, on distingue :

Contrainte de domaine

Tout attribut d'un enregistrement doit respecter le domaine indiqué dans le schéma relationnel.

Attention, certains domaines sont subtils. Par exemple, si une relation possède un attribut "Code Postal", le domaine de cet attribut devra être String plutôt que Entier . Dans le cas contraire, un enregistrement possédant le code postal 03150 serait converti en 3150 (car pour les entiers, 03150 = 3150). Or le code postal 3150 n'existe pas.

Contrainte de relation (ou d'entité)

La contrainte de relation impose que tout enregistrement soit unique : cette contrainte est réalisée par l'existence obligatoire d'une clé primaire. Cette clé primaire est souvent créée de manière artificielle (voir id_auteur dans le paragraphe **Redondance des données**)

Contrainte de référence

La cohérence entre les différentes tables d'une base de données est assurée par les clés étrangères : dans une table, la valeur d'un attribut qui est clé étrangère doit obligatoirement pouvoir être retrouvée dans la table dont cet attribut est clé primaire.

Exercice n° 7

1. Quelle est la contrainte non respectée dans l'exercice 2 ?
2. Quelle est la contrainte non respectée dans l'exercice 4 ?

3 SGBD

Dans une base de données, l'information est stockée dans des fichiers.
Pour y avoir accès, il faut utiliser un logiciel que l'on appelle "système de gestion de base de données" (SGBD). Il en existe plusieurs, des gratuites, des payantes, des libres et des propriétaires.

- les SGBD permettent de gérer la lecture, l'écriture ou la modification des informations contenues dans une base de données
- les SGBD permettent de gérer les autorisations d'accès à une base de données. Il est en effet souvent nécessaire de contrôler les accès par exemple en permettant à l'utilisateur A de lire et d'écrire dans la base de données alors que l'utilisateur B aura uniquement la possibilité de lire les informations contenues dans cette même base de données.
- les fichiers des bases de données sont stockés sur des disques durs dans des ordinateurs, ces ordinateurs peuvent subir des pannes. Il est souvent nécessaire que l'accès aux informations contenues dans une base de données soit maintenu, même en cas de panne matérielle. Les bases de données sont donc dupliquées sur plusieurs ordinateurs afin qu'en cas de panne d'un ordinateur A, un ordinateur B contenant une copie de la base de données présente dans A, puisse prendre le relais. Tout cela est très complexe à gérer, en effet toute modification de la base de données présente sur l'ordinateur A doit entraîner la même modification de la base de données présente sur l'ordinateur B. Cette synchronisation entre A et B doit se faire le plus rapidement possible, il est fondamental d'avoir des copies parfaitement identiques en permanence. C'est aussi les SGBD qui assurent la maintenance des différentes copies de la base de données.
- plusieurs personnes peuvent avoir besoin d'accéder aux informations contenues dans une base données en même temps. Cela peut parfois poser problème, notamment si les 2 personnes désirent modifier la même donnée au même moment (on parle d'accès concurrent). Ces problèmes d'accès concurrent sont aussi gérés par les SGBD.

Comme nous venons de la voir, les SGBD jouent un rôle fondamental. L'utilisation des SGBD explique en partie la supériorité de l'utilisation des bases de données sur des solutions plus simples à mettre en oeuvre comme les fichiers au format CSV.

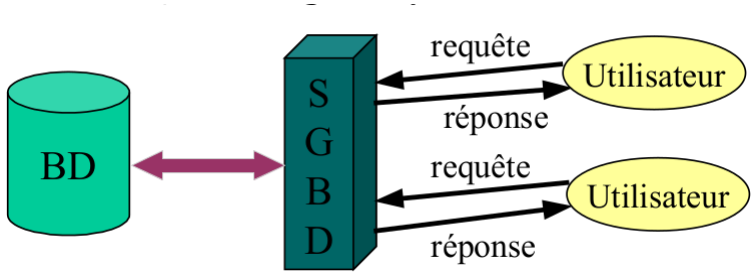
Ce sont des logiciels dont la conception est bien trop complexe pour pouvoir être abordée dans ce cours ; nous nous contenterons d'interagir avec eux par l'intermédiaire de requêtes exprimées dans un langage devenu standard au fil des temps : le langage SQL (pour Structured Query Language).

Définition : Une requête est une demande que l'on fait à une base de données

3.1 Architecture à deux tiers

Sur un réseau informatique, des informations sont en permanence échangées entre deux machines, un logiciel assurant le traitement des informations sur chacune d'entre elles. On distingue le logiciel client installé sur la machine qui envoie des requêtes au logiciel serveur installé sur la machine qui traite les requêtes.

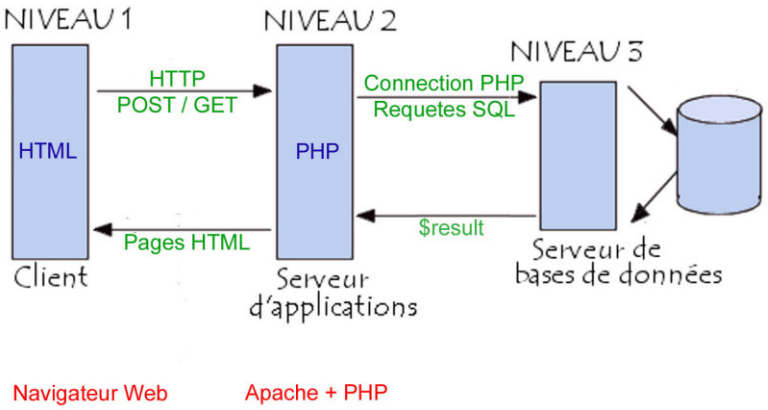
Ce mode de communication est appelé architecture à deux niveaux. C'est l'une des architectures client-serveur possibles.



3.2 Architecture à trois-tiers

L'architecture trois-tiers est une architecture client-serveur qui ajoute un niveau supplémentaire dans l'environnement précédemment décrit. Un serveur de données transmet les informations à un serveur d'application qui, à son tour, transmet les informations traitées vers un client. Ce modèle d'architecture présente plusieurs avantages :

— meilleure prise en compte de l'hétérogénéité des plates-formes ;
— amélioration de la sécurité des données en supprimant le lien entre le client et les données ;
— meilleure répartition des tâches entre les différentes couches logicielles.



4 Requete SQL

4.1 Echauffement

Pour s'échauffer, voici quelques exemples de requêtes (içi interrogations).
A l'aide de la base de donnée de l'exercice 6. Répondre aux requêtes ci-dessous :

1. Quel est le nom de l'équipe de code "GFC" ?
2. Combien y a-t-il d'étapes ?
3. Quelles sont les étapes de plus de 190 km ?
4. Quelles sont les équipes dont le nom comporte la chaine "Team" ?
5. Donner la liste des coureurs de l'équipe Deceuninck.
6. Quels temps à mis Alaphilippe lors de l'étape Sisteron-Orcière ?
7. Donner le dossard des coureurs et le temps mis lors de l'étape partant de Gap.

4.2 DB Browser

Création de la base de données

1. Télécharger puis décompresser l'archive TourDeFrance.zip
2. Importer les trois tables dans **DB Browser**
3. Vérifier le schéma de chaque relation et préciser la clef primaire (CP ou PK)

4.3 Requête interrogative

— Le résultat d'une requête interrogative sera toujours une table/une relation.

— Une requête SQL se termine toujours par ; (un point virgule)

— Une requête interrogative (simple) est de la forme :

```
SELECT <liste d'attribut(s)>  #colonnes que l'on veut garder
FROM <liste de table(s)> ;  #à partir de quelles tables
```

1. Quels sont les noms et prénoms des coureurs ?

```
SELECT Coureurs.nomCoureur,Coureurs.prénomCoureur
FROM Coureurs;
```

Lorsqu'il n'y a pas d'ambiguïté pour la désignation d'un attribut il est possible d'omettre le nom de la table dans sa désignation :

```
SELECT nomCoureur,prénomCoureur
FROM Coureurs;
```

Pour tout garder tous les attributs :

```
SELECT * FROM Coureurs;
```

2. *C'est à vous :* Donner la liste des noms des équipes ?

```
...
```

3. *C'est à vous :* Quelles sont les villes de départ ?

```
...
```

On évite les doublons :

```
SELECT DISTINCT VilleDépart FROM Etapes;
```

4. Donner la liste des étapes dans l'ordre croissant des noms de ville de départ..

```
SELECT * FROM Etapes
ORDER BY VilleDépart ASC;
```

5. *C'est à vous :* Donner la liste des coureurs triés dans l'ordre croissant de leur Nom.

```
...
...
```

ASC (dans l'ordre), DESC (ordre inverse)

6. Quel est l'étape la plus petite ?

```
SELECT *, MIN(km) FROM  Etapes;
```

7. Quel est le nombre de participants ?

```
SELECT COUNT(*) FROM  Coureurs;
```

8. Combien y a t-il de ville de départ ? (Réponse 3)

```
SELECT COUNT(DISTINCT VilleDépart) FROM  Etapes;
```

On pourra utiliser les fonctions suivantes :
- **AVG( ) : moyenne des valeurs**
- **SUM( ) : somme des valeurs**
- **MIN( ) : valeur minimum, MAX( ) : valeur maximum**
- **COUNT(DISTINCT( )**

9. *C'est à vous* : Quelles est la distance moyenne des étapes du tour ?

```
...
```

10. *C'est à vous* : Quelles est la distance totale des étapes du tour en m ?

```
...
```

Possibilité de ne garder que certaines lignes en ajoutant de la clause optionnelle WHERE.

```
SELECT ...  FROM ...
WHERE <condition> ;
```

<condition> est une condition portant sur des attributs et/ou valeurs et peut utiliser différents opérateurs :

- **opérateurs de comparaison <, >, <=,>=,=,<> équivaut à ! =**
- **opérateurs booléens AND, OR, NOT**
- **prédicat d'appartenance à un intervalle : BETWEEN ... AND ...**
- **prédicat de recherche : LIKE**

  Si un tuple donné satisfait la condition alors il est gardé sinon il est écarté.

11. Donner le nom et prénom des coureurs de codeEquipe "COF".

```
SELECT  nomCoureur,prénomCoureur FROM  Coureurs
WHERE codeEquipe="COF";
```

12. Donner les étapes dont le kilométrage est compris en 160 et 190km.

```
SELECT  * FROM  Etapes
WHERE km>=160 AND km<=190;
```

On peut aussi utiliser :

```
SELECT  * FROM  Etapes
WHERE km BETWEEN 160 AND 190;
```

13. Donner les coureurs dont le nom commence par un C.

```
SELECT  * FROM  Coureurs
WHERE nomCoureur LIKE 'c%';
```

14. Donner les coureurs dont le nom fini par un E

```
SELECT  * FROM  Coureurs
WHERE nomCoureur LIKE '%E';
```

15. Donner les équipes dont le nom contient "team".

```
SELECT  * FROM  Equipes
WHERE nomEquipe LIKE '%team%';
```

16. *C'est à vous :* Donner les coureurs dont le nom ET le prénom contiennent un a. (On ordonnera les résultats par ordre croissant des noms)

```
...
...
...
```

17. Observer le résultat de :

```
SELECT  * FROM  Coureurs,Equipes;
```

Vocabulaire : On dit que l'on fait le **produit cartésien** de deux relations.

18. *C'est à vous :*

— Combien y a t-il d'enregistrements dans la requête précédentes ?
— Combien y a t-il d'enregistrements dans la relation Coureurs ?
— Combien y a t-il d'enregistrements dans la relation Équipes ?

On va supprimer certaines lignes :

```
SELECT  * FROM  Coureurs,Equipes
WHERE Coureurs.codeEquipe=Equipes.codeEquipe;
```

On va supprimer certaines colonnes :

```
SELECT  nomCoureur,prénomCoureur,nomEquipe    #Attributs à afficher
FROM  Coureurs,Equipes                        #table à utiliser
WHERE Coureurs.codeEquipe=Equipes.codeEquipe; #Jointure
```

Remarque : la jointure fait le lien entre la clef primaire et la clé étrangère.

On va supprimer certaines colonnes :

```
SELECT  nomCoureur,prénomCoureur,nomEquipe    #Attributs à afficher
FROM  Coureurs,Equipes                        #table à utiliser
WHERE Coureurs.codeEquipe=Equipes.codeEquipe; #Jointure
AND Equipes.codeEquipe='COF'                  #contrainte
```

On peut aussi écrire :

```
SELECT  nomCoureur,prénomCoureur,nomEquipe             #Attributs à afficher
FROM  Equipes                                          #table à utiliser
JOIN Coureurs ON Coureurs.codeEquipe=Equipes.codeEquipe #Jointure
WHERE Equipes.codeEquipe='COF';                        #contrainte
```

19. *C'est à vous :* Qu'est ce que la requête précédente a permis d'afficher ?

...

20. Donner le dossard des coureurs et le temps mis lors de l'étape 4.

```
SELECT  Temps.dossard,nomCoureur, tempsRéalisé,numéroEtape
FROM  Temps,Coureurs
WHERE Temps.dossard=Coureurs.dossard AND  numéroEtape=4;
```

```
SELECT  * FROM  Coureurs
JOIN  Temps ON Temps.dossard=Coureurs.dossard
WHERE   Temps.numéroEtape=4;
```

21. Donner le dossard des coureurs et le temps mis lors de l'étape partant de Gap.

```
SELECT  Temps.dossard,nomCoureur, tempsRéalisé,Temps.numéroEtape
FROM  Temps,Coureurs,Etapes
WHERE Temps.dossard=Coureurs.dossard
AND  Temps.numéroEtape=Etapes.numéroEtape
AND Etapes.villeDépart='Gap'
```

```
SELECT  Temps.dossard,nomCoureur, tempsRéalisé,Temps.numéroEtape
FROM  Temps
JOIN Coureurs ON Temps.dossard=Coureurs.dossard
JOIN Etapes ON Temps.numéroEtape=Etapes.numéroEtape
WHERE Etapes.villeDépart='Gap';
```

22. *C'est à vous :* Donner la liste des coureurs de l'équipe Deceuninck.

```
...
...
...
```

```
...
...
...
```

23. *C'est à vous :* Quels temps à mis Alaphilippe lors de l'étape Sisteron-Orcière ?

```
...
...
...
...
...
```

```
...
...
...
...
```