Système d'information et base de données

CM6 : Optimisation des SGBR et du SQL

Mickaël Martin Nevot

V2.2.2

Système d'information et base de données

I. Présentation du cours
II. SI
III. SGBD
IV. Design
V. Droits
VI. Maintenance
VII. Réplication/Sécurité
VIII. Optimisation

Infrastructure réseau

- Interface réseau rapide :
  - **Fibre optique** (avec carte[s] réseau[x] compatible[s])
  - ***Switch*** à la place de hub
- Parallélisation et **répartition de charge** :
  - *Switch*(s) ou « ferme » de serveurs
- Protocole **réseau déterministe** :
  - 100 VG anylan
  - SPX/IPX
  - Etc.
- Adresses (IP, IPX, etc.) fixes

Le protocole TCP est performant sur des réseaux WAN mais moins intéressant que SPX sur des réseaux LAN

Infrastructure réseau

Utilisateurs simultanés	**15**	**15**	**50**	**50**	**300**	**300**
Débit	faible	fort	faible	fort	faible	fort
Architecture réseau	10 Mo/s 1 carte	100 Mo/s 1 carte	100 Mo/s 1 - 2 cartes	100 Mo/s 1 - 4 cartes, *switch* frontal	1 Go/s 1 carte fibre optique, *switch* frontal	1 Go/s 2 - 4 cartes fibre optique, *switch* frontal en cascade

Au dessus de 300 utilisateurs simultanés, il est nécessaire d'avoir du *clustering*

Utiliser des *switch* en cascade permet d'avoir des équipements plus petits et donc moins chers

Serveur « physique »

- **Alimentation redondante** (*hot plug* et *hot swap*)
- Mémoire **RAM ECC** (*error checking and correction*) :
  - Taille idéale de RAM : $256\ Mo + taille_{BD}$

Avec cette taille idéale de RAM, le serveur peut monter toutes les données de la BD en mémoire vive et en avoir encore suffisamment pour fonctionner

- Carte(s) réseau redondante(s) **IPsec** (permettant d'avoir plusieurs adresses logiques)

L'idéal est d'avoir une machine dédiée au SGBDR, et même une seule base de données par machine si cela est possible !

Un PC personnel ne doit pas servir de serveur car l'architecture d'un serveur est orienté vers la parallélisation des flux de données et l'accès rapide aux ressources disque, tandis qu'un PC personnel est plutôt conçu pour du traitement graphique efficace

Serveur « physique »

- Ondulation électrique (**UPS**) *on line* :

  Attention : tous les onduleurs ne sont pas destinés à filtrer le courant électrique et à protéger de la surtension ou des parasites

- Disques durs **RAID SCSI** :
  - Taux idéal d'occupation des disques dur : **inférieur à 67 %**

    Plus un disque dur se remplit, plus les temps d'accès sont long : le phénomène n'est pas linéaire et peut aller jusqu'au blocage

- Support physique de **sauvegarde externe** :
  - Une sauvegarde logique des données ne suffit pas
  - Le RAID n'est pas suffisant en cas de dégât des eaux, incendie, suppression malencontreuse, piratage, etc.

Différents types d'onduleurs

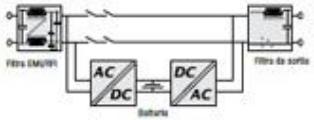
Bien choisir son onduleur

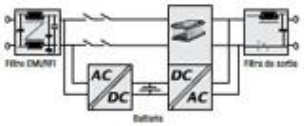
Les pannes électriques sont très souvent à l'origine des pertes de données informatiques. Leur origine peut être diverse : Coupures ou Microcoupures Réseau, Surtensions, Baisses de Tension, Variations de fréquence, Distorsions harmoniques...
Eaton propose un large éventail de solutions d'Onduleurs, basées sur trois technologies différentes, selon le niveau de protection approprié.

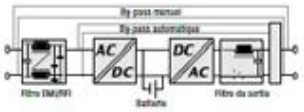
Off-line (grand public)

(In-)line-interactive

On-line (double conversion)



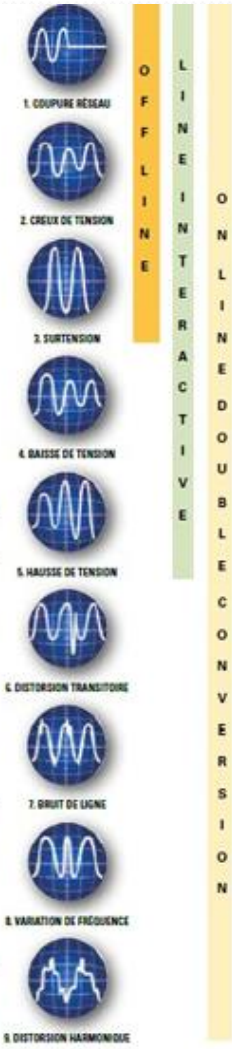




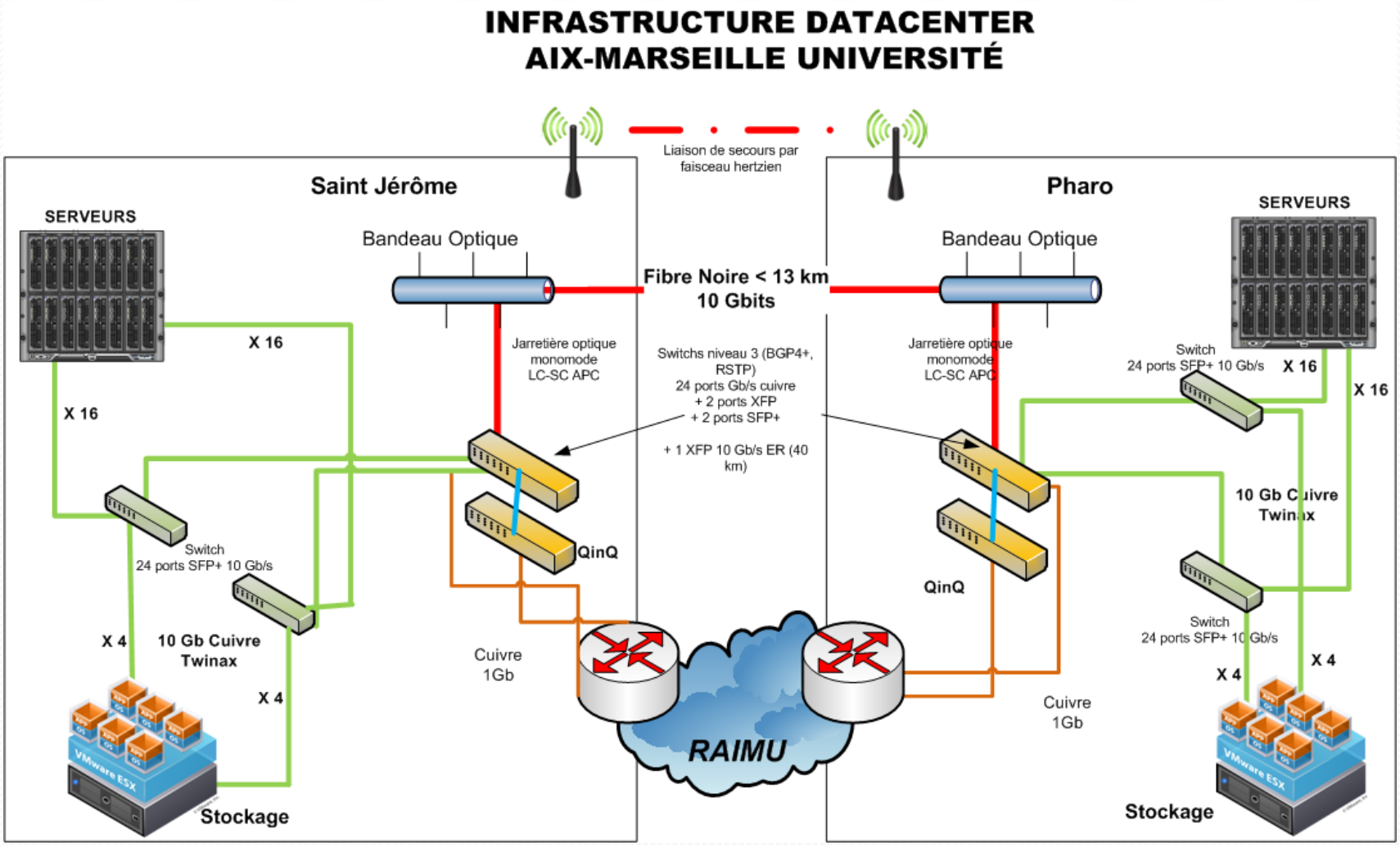
La technologie Off-Line (ou Passive Stand-By) est la plus fréquente pour la protection des PC en environnement peu perturbé. En mode normal, l'onduleur alimente l'application avec le secteur, simplement filtré mais sans aucune conversion d'énergie. Son principe de fonctionnement est séquentiel (sur secteur/sur batterie).
En cas de coupure, de baisse ou hausse de tension, l'onduleur puise son énergie dans sa batterie pour fournir une énergie stabilisée. Son utilisation est inadaptée en cas de perturbations fréquentes (environnements industriels ou fortement perturbés).
Avantage : très économique.

La technologie Line-Interactive est utilisée pour protéger les réseaux et les applications informatiques des entreprises.
En mode normal, l'appareil est géré par un microprocesseur qui surveille la qualité du réseau électrique et réagit aux variations. Un booster et un fader, circuits de compensation de tension, sont activés en cas de variation de l'amplitude de la tension.
Avantage : pallie les baisses ou les hausses de tension prolongées sans sollicitation des batteries.

La technologie double-conversion (On-Line) est adaptée à la protection centralisée de serveurs garantissant une qualité constante quelles que soient les perturbations du secteur.
Dans l'onduleur On-Line, la double conversion permanente élimine les perturbations électriques qui peuvent endommager un ordinateur : le courant est entièrement régénéré par transformation d'alternatif en continu, puis à nouveau de continu en alternatif.
Il est indispensable pour la protection des installations vitales à l'entreprise et assure une protection permanente. L'onduleur On-Line est compatible avec tout type de charge car il ne génère pas de micro-coupure lors du passage sur batterie.
Avantage : technologie la plus performante, application constamment protégée contre tout type de perturbation, régulation permanente de la tension de sortie (amplitude et fréquence), continuité de service grâce au by-pass.



Exemple de data center



SGBDR (MySQL)

- **Collation binaire** :
  - Collation (ou interclassement) : règles définissant la comparaison de caractères pour un jeu de caractères
  - Collation binaire : collation utilisant les codes (ASCII) des caractères pour la comparaison

Sensible à la casse, comme `BINARY CHAR`, `BINARY VARCHAR`, `BLOB`, etc.

- Tailles des ***buffers*** (de chaque *thread*) adaptée :
  - Une pile (`thread_stack` : 64 ko par défaut)
  - Un buffer de connexion (`net_buffer_length`)
  - Un buffer de résultat (`net_buffer_length`)

Les buffers de connexion et de résultat sont dynamiquement élargit jusqu'à `max_allowed_packet` suivant les besoins

Base de données

- Placer les fichiers de données et ceux de journalisation sur des disques durs distincts et différents du disque système
- Précisez une taille de page de données adaptée :

RAM	256 Mo	512 Mo	1 Go	2 Go	4 Go
Disque	4 Go	4 Go	9 Go	18 Go	36 Go
Base	600 Mo	1 Go	3 Go	8 Go	25 Go
Page	**2 Ko**	**4 Ko**	**8 Ko**	**16 Ko**	**32 Ko**

Taille par défaut d'une page de données avec InnoDB : 16 Ko (configurable de 8 Ko à 64 Ko)
Non applicable pour MyISAM (enregistré au format machine)

Schéma relationnel

- **Normaliser** au maximum, dénormaliser à bon escient
- Type et format de champs standardisés

  Des champs comparables devraient avoir le même type et format de données
- **Clef** :
  - Purement informatique (incrémentale ?)
  - Non composée
  - Même taille que celle des mots du processeur

  INTEGER ?

  Pour un système d'exploitation 32 bits, il faut donc un champ de 4 octets
- Champ de données calculable :
  - A éviter
  - Ne pas permettre les valeurs nulles

Schéma relationnel

- `CHAR` au lieu de `VARCHAR` pour les recherches/jointures
- **Contraintes** :
  - Relationnelles → sémantiques (*triggers*) → applicatives

Pas d'enregistrement **orphelin**

- **Pas de jointure** sur plus de deux tables
- **Pas de doublon** (sélection avant insertion au besoin)
- **Pas de** `DISTINCT`
- **Index essentiels**, index textuel pour recherche **plein texte**
- **Table des dates** plutôt que fonctions de calcul temporel

Utiliser le plus souvent possible des ***scripts*** « de nuit »

Formatage applicatif des données

Format	Règles
Date	Retirer tous les caractères non chiffres et mise au format JJ/MM/AAAA
Téléphone	Formater une chaine de chiffre en téléphone : Onze chiffres : 33 1 45 78 45 78 Dix chiffres : 01 45 78 45 78 Sinon : groupes de deux et si impair, alors commence par un chiffre isolé
Nom/ Prénom	Ne garder que les caractères a à z (minuscule), accents et lettres diacritiques et les caractères (espace), – et ' (espace, tiret, apostrophe) avec capitalisation des initiales
E-mail	Mettre en minuscule et utiliser les expression régulière : #^(([a-z0-9!\#$%&\\\'*+/=?^_`{\|}~-]+\.?)*[a-z0-9!\#$%&\\\'*+/=?^_`{|}~-]+)@(([a-z0-9-_]+\.?)*[a-z0-9-_]+)\.[a-z]{2,}$#i
Id	Ne garder que les caractères A à Z (majuscule) et le caractère _ Transformer les accents et lettres diacritiques Tous les autres caractères sont rejetés
...	...

Format français

Optimiseur

- L'optimiseur MySQL n'est pas parfait
- Ne fonctionne que pour SQL (pas en dehors du SGBDR)
- Pour aider l'optimiseur :
  - EXPLAIN (avoir des informations sur une requête) :

```
mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
```

  - FORCE INDEX/IGNORE INDEX
    (chercher dans les index avant les données / le contraire) :

```
mysql> SELECT * FROM t1, t2 FORCE INDEX (index) WHERE t1.i = t2.i;
```

  - STRAIGHT_JOIN (forcer l'ordre des tables à celui donné) :

```
mysql> SELECT STRAIGHT_JOIN * FROM t1, t2 WHERE t1.i = t2.i;
```

Bonnes pratiques

- **Minimiser les données** de résultat :
  - Sélections fines ← Moins d' enregistrements
  - Projections fines ← Moins de champs
  - Clause `LIMIT` ← Moins d'enregistrements
- **Renommage et alias courts**
- **Pas de qualification** des champs non ambigüs
- **Pas de jointure inutile** (utilisation des redondances)
- Un champ seul comme premier opérande d'un opérateur de comparaison
- Pas de joker en début de mot pour une recherche avec `LIKE` :
  - En cas de besoin, faire une recherche sur le mot renversé

Bonne pratiques

- **Éviter négation** (`NOT`) et différence (<>)
- **Pas d'utilisation de** `BLOB`/`Text` (ressources externes)
- **Pas d'utilisation de** `CASE`
- Utiliser un *charset* nécessaire et suffisant pour l'application
- Préciser tous les champs (non nuls) dans un `INSERT`
- Tables toujours dans le même ordre (lexicographique ?)
- **Une seule requête** par session pour les :
  - Données peu sensibles au rafraichissement
  - Opérations sur des données déjà rapatriées (mise en majuscules, abréviations, etc.)

Bonne pratiques

- **Éviter la clause** `ORDER BY`
- Utilisation d'un champ supplémentaire pour le rang
- Utilisation de `UNION ALL` si doublons acceptables
- Faire des **transactions courtes**
- **Éviter les** `CURSOR`
- **Procédures stockées** plutôt que requêtes complexes
- Après mise à jour importantes :
  - Mettre à jour les index
  - Mettre à jour les statistiques (`myisamchk`, etc.)

Transformations usuelles

SELECT *

SELECT col

```
SELECT *
FROM   T_CLIENT
```

```
SELECT CLI_ID, TIT_CODE, CLI_NOM,
       CLI_PRENOM, CLI_ENSEIGNE
FROM   T_CLIENT
```

Transformations usuelles

SELECT *

SELECT col

```
SELECT *
FROM   T_CLIENT
```

```
SELECT CLI_ID, TIT_CODE, CLI_NOM,
       CLI_PRENOM, CLI_ENSEIGNE
FROM   T_CLIENT
```

Transformations usuelles

EXISTS (SELECT *

EXISTS (SELECT col

```
SELECT CHB_ID
FROM   T_CHAMBRE T1
WHERE  NOT EXISTS (SELECT *
                   FROM   TJ_CHB_PLN_CLI T2
                   WHERE  PLN_JOUR = '2000-11-11'
                     AND  T2.CHB_ID = T1.CHB_ID)
```

```
SELECT CHB_ID
FROM   T_CHAMBRE T1
WHERE  NOT EXISTS (SELECT CHB_ID
                   FROM   TJ_CHB_PLN_CLI T2
                   WHERE  PLN_JOUR = '2000-11-11'
                     AND  T2.CHB_ID = T1.CHB_ID)
```

Transformations usuelles

EXISTS (SELECT *

EXISTS (SELECT col

```
SELECT CHB_ID
FROM   T_CHAMBRE T1
WHERE  NOT EXISTS (SELECT *
                   FROM   TJ_CHB_PLN_CLI T2
                   WHERE  PLN_JOUR = '2000-11-11'
                     AND  T2.CHB_ID = T1.CHB_ID)
```

```
SELECT CHB_ID
FROM   T_CHAMBRE T1
WHERE  NOT EXISTS (SELECT CHB_ID
                   FROM   TJ_CHB_PLN_CLI T2
                   WHERE  PLN_JOUR = '2000-11-11'
                     AND  T2.CHB_ID = T1.CHB_ID)
```

Transformations usuelles

SELECT COUNT(*)

SELECT COUNT(col)

```
SELECT COUNT (*)
FROM T_CHAMBRE
```

```
SELECT COUNT (CHB_ID)
FROM T_CHAMBRE
```

Transformations usuelles

SELECT COUNT(*)

SELECT COUNT(col)

```
SELECT COUNT (*)
FROM T_CHAMBRE
```

```
SELECT COUNT (CHB_ID)
FROM T_CHAMBRE
```

Transformations usuelles

SELECT DISTINCT col

SELECT col

```
SELECT DISTINCT CHB_NUMERO, CHB_ETAGE
FROM T_CHAMBRE
```

```
SELECT  CHB_NUMERO, CHB_ETAGE
FROM T_CHAMBRE
```

Transformations usuelles

SELECT DISTINCT col

SELECT col

```
SELECT DISTINCT CHB_NUMERO, CHB_ETAGE
FROM T_CHAMBRE
```

```
SELECT  CHB_NUMERO, CHB_ETAGE
FROM T_CHAMBRE
```

Transformations usuelles

SELECT DISTINCT

EXISTS (SELECT *

```
SELECT DISTINCT CLI_NOM, CLI_PRENOM
FROM   T_CLIENT C
       JOIN TJ_CHB_PLN_CLI J
            ON C.CLI_ID = J.CLI_ID
WHERE  PLN_JOUR = '2000-11-11'
```

```
SELECT CLI_NOM, CLI_PRENOM
FROM   T_CLIENT C
WHERE  EXISTS (SELECT *
               FROM   TJ_CHB_PLN_CLI J
               WHERE  C.CLI_ID = J.CLI_ID
                 AND  PLN_JOUR = '2000-11-11')
```

Transformations usuelles

SELECT DISTINCT

EXISTS (SELECT *

```
SELECT DISTINCT CLI_NOM, CLI_PRENOM
FROM   T_CLIENT C
       JOIN TJ_CHB_PLN_CLI J
            ON C.CLI_ID = J.CLI_ID
WHERE  PLN_JOUR = '2000-11-11'
```

```
SELECT CLI_NOM, CLI_PRENOM
FROM   T_CLIENT C
WHERE  EXISTS (SELECT *
               FROM   TJ_CHB_PLN_CLI J
               WHERE  C.CLI_ID = J.CLI_ID
                 AND  PLN_JOUR = '2000-11-11')
```

Transformations usuelles

BETWEEN

LIKE

```
SELECT *
FROM   T_CLIENT
WHERE  CLI_NOM BETWEEN 'D' AND 'E '
```

```
SELECT *
FROM   T_CLIENT
WHERE  CLI_NOM LIKE 'D%'
```

Transformations usuelles

BETWEEN

LIKE

```
SELECT *
FROM   T_CLIENT
WHERE  CLI_NOM BETWEEN 'D' AND 'E '
```

```
SELECT *
FROM   T_CLIENT
WHERE  CLI_NOM LIKE 'D%'
```

Transformations usuelles

<, >

BETWEEN

```
SELECT *
FROM   T_FACTURE
WHERE  FAC_DATE > '2000-06-18'
  AND  FAC_DATE < '2000-07-15'
```

```
SELECT *
FROM   T_FACTURE
WHERE  FAC_DATE BETWEEN '2000-06-18'
                AND '2000-07-14'
```

Transformations usuelles

<, >

BETWEEN

```
SELECT *
FROM   T_FACTURE
WHERE  FAC_DATE > '2000-06-18'
  AND  FAC_DATE < '2000-07-15'
```

```
SELECT *
FROM   T_FACTURE
WHERE  FAC_DATE BETWEEN '2000-06-18'
                AND '2000-07-14'
```

Transformations usuelles

BETWEEN

IN

```
SELECT *
FROM T_CHAMBRE
WHERE CHB_NUMERO BETWEEN 11  AND 14
```

```
SELECT *
FROM   T_CHAMBRE
WHERE  CHB_NUMERO IN (11, 12, 13, 14)
```

Transformations usuelles

BETWEEN

IN

```
SELECT *
FROM T_CHAMBRE
WHERE CHB_NUMERO BETWEEN 11  AND 14
```

```
SELECT *
FROM   T_CHAMBRE
WHERE  CHB_NUMERO IN (11, 12, 13, 14)
```

Transformations usuelles

EXISTS/NOT EXISTS

IN/NOT IN

```
SELECT CHB_ID
FROM   T_CHAMBRE T1
WHERE  NOT EXISTS (SELECT *
                   FROM   TJ_CHB_PLN_CLI T2
                   WHERE  PLN_JOUR = '2000-11-11'
                     AND  T2.CHB_ID = T1.CHB_ID)
```

```
SELECT CHB_ID
FROM   T_CHAMBRE
WHERE  CHB_ID NOT IN (SELECT CHB_ID
                      FROM   TJ_CHB_PLN_CLI
                      WHERE  PLN_JOUR = '2000-11-11')
```

Transformations usuelles

EXISTS/NOT EXISTS

IN/NOT IN

```
SELECT CHB_ID
FROM   T_CHAMBRE T1
WHERE  NOT EXISTS (SELECT *
                   FROM   TJ_CHB_PLN_CLI T2
                   WHERE  PLN_JOUR = '2000-11-11'
                     AND  T2.CHB_ID = T1.CHB_ID)
```

```
SELECT CHB_ID
FROM   T_CHAMBRE
WHERE  CHB_ID NOT IN (SELECT CHB_ID
                      FROM   TJ_CHB_PLN_CLI
                      WHERE  PLN_JOUR = '2000-11-11')
```

Transformations usuelles

= ANY

IN

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE = ANY (SELECT CHB_COUCHAGE
                          FROM   T_CHAMBRE
                          WHERE  CHB_ETAGE ='RDC')
```

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE IN (SELECT CHB_COUCHAGE
                       FROM   T_CHAMBRE
                       WHERE  CHB_ETAGE ='RDC')
```

Transformations usuelles

= ANY

IN

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE = ANY (SELECT CHB_COUCHAGE
                          FROM   T_CHAMBRE
                          WHERE  CHB_ETAGE ='RDC')
```

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE IN (SELECT CHB_COUCHAGE
                       FROM   T_CHAMBRE
                       WHERE  CHB_ETAGE ='RDC')
```

Transformations usuelles

NOT IN

<> ALL

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE
      NOT IN (SELECT CHB_COUCHAGE
              FROM   T_CHAMBRE
              WHERE  CHB_ETAGE ='RDC')
```

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE <> ALL (SELECT CHB_COUCHAGE
                           FROM   T_CHAMBRE
                           WHERE  CHB_ETAGE ='RDC')
```

Transformations usuelles

NOT IN

<> ALL

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE
      NOT IN (SELECT CHB_COUCHAGE
              FROM   T_CHAMBRE
              WHERE  CHB_ETAGE ='RDC')
```

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE <> ALL (SELECT CHB_COUCHAGE
                           FROM   T_CHAMBRE
                           WHERE  CHB_ETAGE ='RDC')
```

Transformations usuelles

Requêtes imbriquées ANY/ALL

Requêtes imbriquées Agrégat

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE > ALL (SELECT CHB_COUCHAGE
                          FROM   T_CHAMBRE
                          WHERE  CHB_ETAGE ='RDC')
```

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE > (SELECT MAX(CHB_COUCHAGE)
                      FROM   T_CHAMBRE
                      WHERE  CHB_ETAGE ='RDC')
```

Transformations usuelles

Requêtes imbriquées ANY/ALL

Requêtes imbriquées Agrégat

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE > ALL (SELECT CHB_COUCHAGE
                          FROM   T_CHAMBRE
                          WHERE  CHB_ETAGE ='RDC')
```

```
SELECT CHB_ID, CHB_COUCHAGE
FROM T_CHAMBRE
WHERE CHB_COUCHAGE > (SELECT MAX(CHB_COUCHAGE)
                      FROM   T_CHAMBRE
                      WHERE  CHB_ETAGE ='RDC')
```

Transformations usuelles

ORDER BY Nombre

ORDER BY Champ

```
SELECT LIF_ID, (LIF_QTE * LIF_MONTANT)
FROM   T_LIGNE_FACTURE
ORDER BY 1, 2
```

```
SELECT LIF_ID,
       (LIF_QTE * LIF_MONTANT) AS LIF_MONTANT
FROM   T_LIGNE_FACTURE
ORDER BY LIF_ID, LIF_MONTANT
```

Transformations usuelles

ORDER BY Nombre

ORDER BY Champ

```
SELECT LIF_ID, (LIF_QTE * LIF_MONTANT)
FROM   T_LIGNE_FACTURE
ORDER BY 1, 2
```

```
SELECT LIF_ID,
       (LIF_QTE * LIF_MONTANT) AS LIF_MONTANT
FROM   T_LIGNE_FACTURE
ORDER BY LIF_ID, LIF_MONTANT
```

Transformations usuelles

JOIN

```
SELECT *
FROM   T_CLIENT C
       JOIN T_FACTURE F
            ON F.CLI_ID = C.CLI_ID
WHERE  EXTRACT(YEAR FROM F.FAC_DATE) = 2000
```

Jointure prédicative

```
SELECT *
FROM   T_CLIENT C, T_FACTURE F
WHERE  EXTRACT(YEAR FROM F.FAC_DATE) = 2000
  AND  F.CLI_ID = C.CLI_ID
```

Transformations usuelles

JOIN

```
SELECT *
FROM   T_CLIENT C
       JOIN T_FACTURE F
            ON F.CLI_ID = C.CLI_ID
WHERE  EXTRACT(YEAR FROM F.FAC_DATE) = 2000
```

Jointure prédicative

```
SELECT *
FROM   T_CLIENT C, T_FACTURE F
WHERE  EXTRACT(YEAR FROM F.FAC_DATE) = 2000
  AND  F.CLI_ID = C.CLI_ID
```

Transformations usuelles

LEFT OUTER JOIN

Requêtes imbriquées

```
SELECT DISTINCT C.CHB_ID
FROM    T_CHAMBRE C
        LEFT OUTER JOIN TJ_CHB_PLN_CLI P
             ON C.CHB_ID = P.CHB_ID
                AND PLN_JOUR = '2000-11-11'
WHERE P.CHB_ID IS  NULL
```

```
SELECT CHB_ID
FROM   T_CHAMBRE
WHERE  CHB_ID NOT IN (SELECT CHB_ID
                      FROM   TJ_CHB_PLN_CLI
                      WHERE  PLN_JOUR = '2000-11-11')
```

Transformations usuelles

LEFT OUTER JOIN

Requêtes imbriquées

```
SELECT DISTINCT C.CHB_ID
FROM    T_CHAMBRE C
        LEFT OUTER JOIN TJ_CHB_PLN_CLI P
             ON C.CHB_ID = P.CHB_ID
                AND PLN_JOUR = '2000-11-11'
WHERE P.CHB_ID IS  NULL
```

```
SELECT CHB_ID
FROM   T_CHAMBRE
WHERE  CHB_ID NOT IN (SELECT CHB_ID
                      FROM   TJ_CHB_PLN_CLI
                      WHERE  PLN_JOUR = '2000-11-11')
```

Transformations usuelles

EXCEPT

LEFT OUTER JOIN

```
SELECT CHB_ID
FROM   T_CHAMBRE
EXCEPT
SELECT CHB_ID
FROM   TJ_CHB_PLN_CLI
WHERE  PLN_JOUR = '2000-11-11'
```

```
SELECT DISTINCT C.CHB_ID
FROM   T_CHAMBRE C
       LEFT OUTER JOIN TJ_CHB_PLN_CLI P
            ON C.CHB_ID = P.CHB_ID
               AND PLN_JOUR = '2000-11-11'
WHERE P.CHB_ID IS  NULL
```

Transformations usuelles

EXCEPT

```
SELECT CHB_ID
FROM   T_CHAMBRE
EXCEPT
SELECT CHB_ID
FROM   TJ_CHB_PLN_CLI
WHERE  PLN_JOUR = '2000-11-11'
```

LEFT OUTER JOIN

```
SELECT DISTINCT C.CHB_ID
FROM   T_CHAMBRE C
       LEFT OUTER JOIN TJ_CHB_PLN_CLI P
            ON C.CHB_ID = P.CHB_ID
               AND PLN_JOUR = '2000-11-11'
WHERE P.CHB_ID IS  NULL
```

Transformations usuelles

INNER JOIN

```
SELECT DISTINCT C.CHB_ID
FROM   T_CHAMBRE C
       INNER JOIN TJ_CHB_PLN_CLI P
             ON C.CHB_ID = P.CHB_ID
WHERE  PLN_JOUR = '2000-11-11'
```

INTERSECT

```
SELECT CHB_ID
FROM   T_CHAMBRE
INTERSECT
SELECT CHB_ID
FROM   TJ_CHB_PLN_CLI
WHERE  PLN_JOUR = '2000-11-11'
```

Transformations usuelles

INNER JOIN

INTERSECT

```
SELECT DISTINCT C.CHB_ID
FROM   T_CHAMBRE C
       INNER JOIN TJ_CHB_PLN_CLI P
             ON C.CHB_ID = P.CHB_ID
WHERE  PLN_JOUR = '2000-11-11'
```

```
SELECT CHB_ID
FROM   T_CHAMBRE
INTERSECT
SELECT CHB_ID
FROM   TJ_CHB_PLN_CLI
WHERE  PLN_JOUR = '2000-11-11'
```

Transformations usuelles

FULL OUTER JOIN

UNION

```
SELECT COALESCE(OBJ_NOM, MAC_NOM) AS NOM,
       COALESCE(OBJ_PRIX, MAC_PRIX) AS PRIX
FROM   T_OBJET O
       FULL OUTER JOIN T_MACHINE M
            ON O.OBJ_NOM = M.MAC_NOM
               AND O.OBJ_PRIX = M.MAC_PRIX
ORDER  BY NOM, PRIX
```

```
SELECT OBJ_NOM AS NOM, OBJ_PRIX AS PRIX
FROM   T_OBJET
UNION
SELECT MAC_NOM AS NOM, MAC_PRIX AS PRIX
FROM   T_MACHINE
ORDER  BY NOM, PRIX
```

Transformations usuelles

FULL OUTER JOIN

UNION

```
SELECT COALESCE(OBJ_NOM, MAC_NOM) AS NOM,
       COALESCE(OBJ_PRIX, MAC_PRIX) AS PRIX
FROM   T_OBJET O
       FULL OUTER JOIN T_MACHINE M
            ON O.OBJ_NOM = M.MAC_NOM
               AND O.OBJ_PRIX = M.MAC_PRIX
ORDER  BY NOM, PRIX
```

```
SELECT OBJ_NOM AS NOM, OBJ_PRIX AS PRIX
FROM   T_OBJET
UNION
SELECT MAC_NOM AS NOM, MAC_PRIX AS PRIX
FROM   T_MACHINE
ORDER  BY NOM, PRIX
```

Transformations usuelles

UNION

```
SELECT CHB_NUMERO, 0 AS ETAGE, CHB_COUCHAGE
FROM   T_CHAMBRE
WHERE CHB_ETAGE = 'RDC'
UNION
SELECT CHB_NUMERO, 1 AS ETAGE, CHB_COUCHAGE
FROM   T_CHAMBRE
WHERE CHB_ETAGE = '1er'
UNION
SELECT CHB_NUMERO, 2 AS ETAGE, CHB_COUCHAGE
FROM   T_CHAMBRE
WHERE CHB_ETAGE = '2e'
ORDER  BY  ETAGE, CHB_COUCHAGE
```

CASE

```
SELECT CHB_NUMERO, CASE CHB_ETAGE
                        WHEN 'RDC' THEN 0
                        WHEN '1er' THEN 1
                        WHEN '2e'  THEN 2
                   END AS ETAGE, CHB_COUCHAGE
FROM   T_CHAMBRE
ORDER  BY  ETAGE, CHB_COUCHAGE
```

Transformations usuelles

UNION

CASE

```
SELECT CHB_NUMERO, 0 AS ETAGE, CHB_COUCHAGE
FROM   T_CHAMBRE
WHERE CHB_ETAGE = 'RDC'
UNION
SELECT CHB_NUMERO, 1 AS ETAGE, CHB_COUCHAGE
FROM   T_CHAMBRE
WHERE CHB_ETAGE = '1er'
UNION
SELECT CHB_NUMERO, 2 AS ETAGE, CHB_COUCHAGE
FROM   T_CHAMBRE
WHERE CHB_ETAGE = '2e'
ORDER  BY  ETAGE, CHB_COUCHAGE
```

```
SELECT CHB_NUMERO, CASE CHB_ETAGE
                        WHEN 'RDC' THEN 0
                        WHEN '1er' THEN 1
                        WHEN '2e'  THEN 2
                   END AS ETAGE, CHB_COUCHAGE
FROM   T_CHAMBRE
ORDER  BY  ETAGE, CHB_COUCHAGE
```

Transformations usuelles

UNION

```
SELECT LIF_ID, (LIF_QTE * LIF_MONTANT)
FROM   T_LIGNE_FACTURE
WHERE  LIF_REMISE_POURCENT IS NULL
  AND  LIF_REMISE_MONTANT IS NULL
UNION
SELECT LIF_ID, (LIF_QTE * LIF_MONTANT)
               - LIF_REMISE_MONTANT
FROM   T_LIGNE_FACTURE
WHERE  LIF_REMISE_POURCENT IS NULL
  AND  LIF_REMISE_MONTANT IS NOT NULL
UNION
SELECT LIF_ID, (LIF_QTE * LIF_MONTANT)
               * (1 - LIF_REMISE_POURCENT/100)
FROM   T_LIGNE_FACTURE
WHERE  LIF_REMISE_POURCENT IS NOT NULL
  AND  LIF_REMISE_MONTANT IS NULL
UNION
SELECT LIF_ID, (LIF_QTE * LIF_MONTANT)
               * (1 - LIF_REMISE_POURCENT/100)
               - LIF_REMISE_MONTANT
FROM   T_LIGNE_FACTURE
WHERE  LIF_REMISE_POURCENT IS NOT NULL
  AND  LIF_REMISE_MONTANT IS NOT NULL
```

COALESCE

```
SELECT LIF_ID,
       (LIF_QTE * LIF_MONTANT)
        * (1 - COALESCE(LIF_REMISE_POURCENT, 0)/100)
        - COALESCE(LIF_REMISE_MONTANT, 0) AS TOTAL_LIGNE
FROM T_LIGNE_FACTURE
```

Transformations usuelles

UNION

```
SELECT LIF_ID, (LIF_QTE * LIF_MONTANT)
FROM   T_LIGNE_FACTURE
WHERE  LIF_REMISE_POURCENT IS NULL
  AND  LIF_REMISE_MONTANT IS NULL
UNION
SELECT LIF_ID, (LIF_QTE * LIF_MONTANT)
               - LIF_REMISE_MONTANT
FROM   T_LIGNE_FACTURE
WHERE  LIF_REMISE_POURCENT IS NULL
  AND  LIF_REMISE_MONTANT IS NOT NULL
UNION
SELECT LIF_ID, (LIF_QTE * LIF_MONTANT)
               * (1 - LIF_REMISE_POURCENT/100)
FROM   T_LIGNE_FACTURE
WHERE  LIF_REMISE_POURCENT IS NOT NULL
  AND  LIF_REMISE_MONTANT IS NULL
UNION
SELECT LIF_ID, (LIF_QTE * LIF_MONTANT)
               * (1 - LIF_REMISE_POURCENT/100)
               - LIF_REMISE_MONTANT
FROM   T_LIGNE_FACTURE
WHERE  LIF_REMISE_POURCENT IS NOT NULL
  AND  LIF_REMISE_MONTANT IS NOT NULL
```

COALESCE

```
SELECT LIF_ID,
       (LIF_QTE * LIF_MONTANT)
        * (1 - COALESCE(LIF_REMISE_POURCENT, 0)/100)
        - COALESCE(LIF_REMISE_MONTANT, 0) AS TOTAL_LIGNE
FROM T_LIGNE_FACTURE
```

Transformations usuelles

Requêtes imbriquées corrélées

JOIN

```
SELECT FAC_ID,   (SELECT MAX(LIF_QTE * LIF_MONTANT)
                  FROM T_LIGNE_FACTURE L
                  WHERE F.FAC_ID = L.FAC_ID)
FROM   T_FACTURE F
ORDER BY FAC_ID
```

```
SELECT F.FAC_ID,  MAX(LIF_QTE * LIF_MONTANT)
FROM   T_FACTURE F
       JOIN T_LIGNE_FACTURE L
            ON F.FAC_ID = L.FAC_ID
GROUP BY F.FAC_ID
ORDER BY F.FAC_ID
```

Transformations usuelles

Requêtes imbriquées corrélées

JOIN

```
SELECT FAC_ID,   (SELECT MAX(LIF_QTE * LIF_MONTANT)
                  FROM T_LIGNE_FACTURE L
                  WHERE F.FAC_ID = L.FAC_ID)
FROM   T_FACTURE F
ORDER BY FAC_ID
```

```
SELECT F.FAC_ID,  MAX(LIF_QTE * LIF_MONTANT)
FROM   T_FACTURE F
       JOIN T_LIGNE_FACTURE L
            ON F.FAC_ID = L.FAC_ID
GROUP BY F.FAC_ID
ORDER BY F.FAC_ID
```

Transformations usuelles

Requêtes imbriquées non corrélées

Requêtes imbriquées corrélées

```
SELECT DISTINCT VILLE_ETP
FROM   T_ENTREPOT
WHERE  RAYON_RYN IN
   (SELECT RAYON_RYN
    FROM   T_ENTREPOT
    WHERE  RAYON_RYN NOT IN
       (SELECT RAYON_RYN
        FROM   T_ENTREPOT
        WHERE  RAYON_RYN NOT IN
           (SELECT RAYON_RYN
            FROM T_RAYON)))
GROUP   BY VILLE_ETP
HAVING  COUNT (*) =
   (SELECT COUNT(DISTINCT RAYON_RYN)
    FROM   T_RAYON)
```

```
SELECT DISTINCT VILLE_ETP
FROM   T_ENTREPOT AS ETP1
WHERE  NOT EXISTS
      (SELECT *
       FROM   T_RAYON RYN
       WHERE  NOT EXISTS
             (SELECT *
              FROM   T_ENTREPOT AS ETP2
              WHERE  ETP1.VILLE_ETP = ETP2.VILLE_ETP
                AND (ETP2.RAYON_RYN = RYN.RAYON_RYN)))
```

Transformations usuelles

Requêtes imbriquées non corrélées

Requêtes imbriquées corrélées

```
SELECT DISTINCT VILLE_ETP
FROM   T_ENTREPOT
WHERE  RAYON_RYN IN
   (SELECT RAYON_RYN
    FROM   T_ENTREPOT
    WHERE  RAYON_RYN NOT IN
       (SELECT RAYON_RYN
        FROM   T_ENTREPOT
        WHERE  RAYON_RYN NOT IN
           (SELECT RAYON_RYN
            FROM T_RAYON)))
GROUP  BY VILLE_ETP
HAVING  COUNT (*) =
   (SELECT COUNT(DISTINCT RAYON_RYN)
    FROM   T_RAYON)
```

```
SELECT DISTINCT VILLE_ETP
FROM   T_ENTREPOT AS ETP1
WHERE  NOT EXISTS
      (SELECT *
       FROM   T_RAYON RYN
       WHERE  NOT EXISTS
             (SELECT *
              FROM   T_ENTREPOT AS ETP2
              WHERE  ETP1.VILLE_ETP = ETP2.VILLE_ETP
                AND (ETP2.RAYON_RYN = RYN.RAYON_RYN)))
```

Antipatterns : conception BD

- **« Tribules » de métadonnées** :
  - Objectif : augmenter les performance d'une très grosse table
  - *Antipattern* : séparation en plusieurs tables uni-compatibles
  - Solution : ajouter une table d'association
- **Relation – champ – valeur** :
  - Objectif : avoir une table avec un nombre variable de champs
  - *Antipattern* : tous les champs dans une autre table en tant qu'enregistrement
  - Solution : autant de tables que nécessaire dont la clef primaire est clef étrangère dans une table de base

Antipatterns : conception BD

- **Associations polymorphiques** :
  - Objectif : avoir plusieurs « parents »
  - *Antipattern* : table sans clef étrangère et faisant systématiquement des jointures sur les tables parentes
  - Solution : une table de base dont la clef primaire est référencée par les autres tables
- **Arbres naïfs** :
  - Objectif : sélectionner/enregistrer des données hiérarchiques
  - *Antipattern* : champs avec l'identifiant du parent
  - Solution : table de fermeture (transitive)

Antipatterns : création de BD

- **Type énuméré** :
  - Objectif : restreindre les valeur d'un champ à une énumération
  - Antipattern : utiliser le type `ENUM` !
  - Solution : utiliser une table de recherche
- **Erreurs d'arrondis** :
  - Objectif : enregistrer des nombres réels exactement
  - Antipattern : utiliser le type `FLOAT` !
  - Solution : utiliser `NUMERIC` !

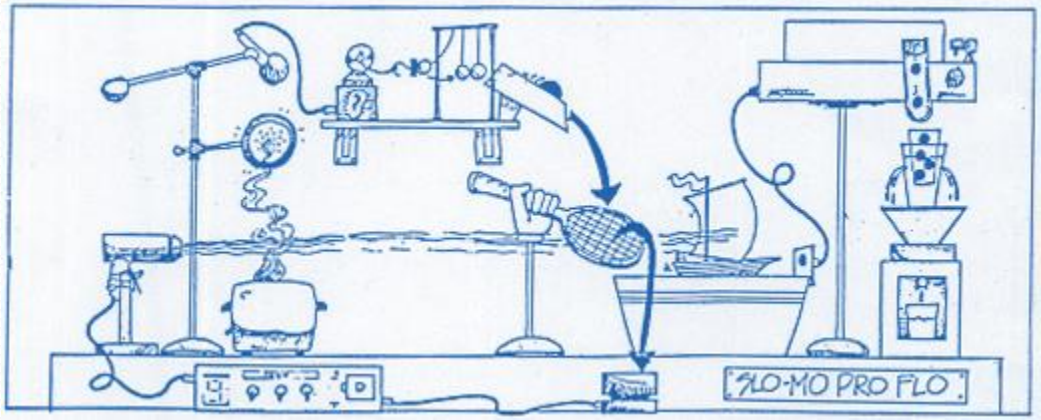
> *10.0 times 0.1 is hardly ever 1.0*
> – Brian Kernighan

Antipatterns : requêtes SQL

- `NULL` :
  - Objectif : manipuler des valeurs « manquantes »
  - *Antipattern* : utiliser `NULL` comme une valeur ordinaire
  - Solution : changer `NULL` en valeur ordinaire avec `COALESCE()`
- **Ordre aléatoire** :
  - Objectif : sélectionne un enregistrement aléatoire
  - *Antipattern* : … `ORDER BY RAND() LIMIT 1`
  - Solution : `SELECT ROUND(RAND() * (SELECT COUNT(*) …)`
    `SELECT * FROM Bugs LIMIT 1 OFFSET`

Antipatterns : requêtes SQL

- **Machine de Rube Goldberg** :
  - Objectif : générer un rapport complet
  - *Antipattern* : essayer de générer toutes les informations du rapport en une seule requête
  - Solution : écrire des requêtes séparées



Antipatterns : applicatif

- **Paramétrage intempestif** :
  - Objectif : construire une requête paramétrée
  - *Antipattern* : une liste de paramètre avec une seule variable
  - Solution : un paramètre = une variable, attention aux injections
- **Effets de bord fantômes** :
  - Objectif : exécuter des tâches applicatives avec des opérations de base de données
  - *Antipattern* : exécuter opérations extérieures avec des *triggers* ou des procédures stockées
  - Solution : ne pas utiliser la base de données pour cela !

Aller plus loin

- Algorithme/fonction Soundex
- Transformations usuelles de la clause `HAVING COUNT`
- Optimisation des tables autoréférencées

Liens

- Documents électroniques :
  - http://sqlpro.developpez.com/cours/optimiser
  - http://www.elliptic.fr/doc/mysql
  - http://dev.mysql.com/doc/refman/5.5/en
- Documents classiques :
  - Maurice Libes. *Administration et exploitation du SGBDR MySQL*.
  - Cyril Gruau. *Conception d'une base de données*.
  - Jean-Marc Petit. *Administration des bases de données*.
  - Bill Karwin. *SQL Antipatterns Strike Back*.

Crédits

Auteur

Mickaël Martin Nevot
mmartin.nevot@gmail.com

Carte de visite électronique

Relecteurs

- Christophe Delagarde

Cours en ligne sur : www.mickael-martin-nevot.com