Bases de Données relaionnelles

SQL

Ilham Alloui

(*Source principale : Vincent Couturier*)

www.polytech.univ-savoie.fr

SQL – Langage d'Interrogation de Données (LID)

- ***Rappels SQL***
- ***Projection***
- ***Restriction/sélection***
- ***Jointures***
- ***Vues***

Langage de requête : SQL

- Structured query language (SQL), ou *langage structuré de requêtes*, : pseudo-langage informatique (de type requête) standard et normalisé, destiné à interroger ou à manipuler une base de données relationnelle
- Une relation est définie par un ensemble d'attributs dont les valeurs représentent des tuples

Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton

tuple

Principes SQL

- Langage fondé sur la logique des prédicats et le modèle relationnel
- Langage non procédural
  - permet de décrire le résultat sans spécifier l'ordre d'exécution des différentes opérations
- Langage ensembliste
- Une requête SQL est équivalente à une suite d'opérations relationnelles (e.g. projection, restriction, jointure)
- SQL est une norme depuis fin 1986

Sélection/Restriction

Sélection

Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton

Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton

Patients de la ville de Paris

Projection

Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton

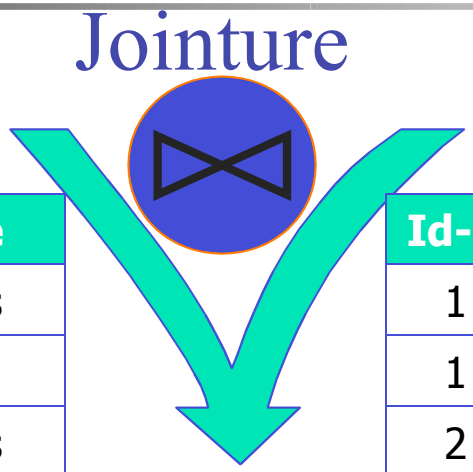
Projection

Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton

Nom et prénom des patients

Jointure



Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton

Visites

Id-D	Id-P	Id-V	Date	Prix
1	2	1	15 juin	250
1	1	2	12 août	180
2	2	3	13 juillet	350
2	3	4	1 mars	250

Id-P	Nom	Prénom	Ville	Id-D	Id-P	Id-V	Date	Prix
1	Lebeau	Jacques	Paris	1	1	2	12 août	180
2	Troger	Zoe	Evry	1	2	1	15 juin	250
2	Troger	Zoe	Evry	2	2	3	13 juillet	350
3	Doe	John	Paris	2	3	4	1 mars	250

Patients et leurs visites

SQL LID : langage d'interrogation

- Expression d'une requête d'interrogation par un bloc :

SELECT <liste des attributs projetés>
FROM <liste des relations touchées par la question>
[WHERE <liste des critères de restriction>
GROUP BY <liste des attributs d'agrégation>
HAVING <liste des critères de restriction sur les groupes>
ORDER BY <liste des attributs de tri du résultat>**]**

- SQL possède également un *langage de définition de données* (création et modification de schéma) et un *langage de manipulation de données* (insertion, suppression et mise à jour de tuples)

Base de données exemple

- Schéma de la base de données EMPLOYEES (2 relations ou tables regroupant des informations sur des départements et des employés) :
  - DEPT(DEPTNO, DNAME, LOC)
  - EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

indique une clé étrangère : un attribut dont les valeurs sont « importées » d'une autre relation (e.g. #DEPTNO défini dans DEPT) ou de la relation elle-même (e.g. #MGR qui prend ses valeurs depuis EMPNO défini dans EMP)

Un attribut souligné est une clé primaire de la relation : identifie de manière unique un tuple de la relation

Projection

Clause SELECT

- L'ordre minimal pour une interrogation est constitué de deux parties : SELECT et FROM, appelées clauses, toujours écrites dans cet ordre
- Recherche simple
  - SELECT : expression des données requises en résultat (attributs de relations, fonctions, etc)
  - FROM : provenance des données (tables, vues, ...)
- Exemple : Sélection de tous les attributs de la relation des services

```
SELECT *
FROM   DEPT
```

"*" affiche toutes les données contenues dans la relation DEPT, dans l'ordre des attributs défini à la création de la relation. L'utilisation du * est cependant à déconseiller dans les applications.

Clause SELECT (suite)

- Sélection d'attributs : projection
  - Sélectionner des attributs dans une relation correspond à l'opération de PROJECTION : extraction d' une partie des attributs d'une relation
  - L'ordre des noms d'attributs dans le SELECT détermine l'ordre des attributs dans le résultat
  - Sélection d'un attribut de la relation des services

```
SELECT dname
FROM   DEPT
```

  - SQL n'élimine pas les doublons, pour les éliminer on utilise DISTINCT

Clause SELECT DISTINCT

- Permet d'éliminer les doublons

- Exemple sans distinct

```
SELECT job
FROM   emp

JOB
---------
CLERK
SALESMAN
SALESMAN
MANAGER
SALESMAN
MANAGER
MANAGER
ANALYST
PRESIDENT
SALESMAN
CLERK
CLERK
ANALYST
CLERK
```

- Exemple avec distinct

```
SELECT DISTINCT job
FROM   emp

JOB
---------
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN
```

Clause FROM

- Décrit les objets (tables, vues...) utilisés par la requête :
  - Nom de la table

```
SELECT deptno
FROM   dept
```

  - Nom du schéma (≃base de données), nom de la table

```
SELECT deptno
FROM   cours1.dept
```

- Création d'un alias

```
SELECT d.deptno
FROM   dept d
```

Colonnes de SELECT

- Constante : 1
- Nom d'attribut d'une relation : deptno
- Expressions arithmétiques, sur chaînes de caractères, ou sur dates : (sal*1.1) + 20
- Fonction simple appliquée à une expression : power(expr,2)
- Fonction agissant sur les valeurs provenant de plusieurs lignes : max(sal)

Constantes

- Constantes numériques

  ```
  -2
  105.37
  3.5E17
  ```

- Constantes de type chaîne alphanumérique

  ```
  'ROSS154'
  'Jaune plombé'
  'Pluie d''étoiles en juillet 1992'
  ```

- Constante de type date

  ```
  '21/07/91'
  ```

- Constante NULL : valeur inconnue

- Exemple NULL

```
SELECT      1,
            NULL
FROM        dept

--------- ----
        1 NULL
        1 NULL
        1 NULL
        1 NULL
```

En-têtes d'attributs

- Peuvent être modifiés
  - Un nom local peut être donné à une colonne de résultat, en vue d'une présentation particulière des en-têtes de colonnes ou de la compréhension de la requête. Ce nom local de colonne ne peut apparaître que dans la clause SELECT
  - Pour créer un nom de colonne comportant des blancs de séparation, ou des caractères spéciaux (/, *, %, etc), il faut placer ce nom entre guillemets
  - Exemple

```
SELECT deptno Service,
       dname "Nom service"
FROM   dept
```

```
Service     Nom service
----------- --------------
         10 ACCOUNTING
         20 RESEARCH
         30 SALES
         40 OPERATIONS
```

OU autre écriture possible

```
SELECT deptno  AS Service,
       dname AS "Nom service"
FROM   dept
```

Expressions arithmétiques

- Opérateurs
  - \+ Addition
  - \- Soustraction
  - * Multiplication
  - / Division

- Utilisables pour tout attribut numérique
- Priorité selon l'ordre arithmétique ou en fonction des parenthèses
- Exemple :

```
SELECT (sal*1.1) + comm
FROM   emp

(SAL*1.1)+COMM
--------------
          2060
          1875
          NULL
          ...
```

Restriction / sélection

Clause ORDER BY

- Tri le résultat d'une requête,
  - Classement ascendant ASC (par défaut), ou descendant (DESC),
- Une seule clause ORDER BY toujours à la fin du SELECT
- Les éléments cités dans la clause ne figurent pas nécessairement dans le SELECT
- Les valeurs nulles se trouvent en fin de résultat si option ASC, en tête si option DESC
- Les éléments de la clause peuvent être le nom des colonnes, le numéro des colonnes, une expression
- Exemples :

```
SELECT      ename,
            sal,
            deptno
FROM        emp
ORDER BY    deptno, sal
```

```
ENAME             SAL     DEPTNO
---------- ---------- ----------
MILLER            1300         10
CLARK             2450         10
...
```

```
SELECT ename,
       sal,
       deptno
FROM   emp
ORDER BY 3, 2, 1
```

```
ENAME             SAL     DEPTNO
---------- ---------- ----------
MILLER            1300         10
CLARK             2450         10
...
```

Clause WHERE

- La clause WHERE réduit le nombre de lignes concernées par la recherche
  - Exemple : Sélection des employés du service 10

```
SELECT  ename, sal
FROM    emp
WHERE   deptno = 10

ENAME                SAL
---------- ----------
CLARK              2450
KING               5000
MILLER             1300
```

- Les conditions de recherche sont des expressions logiques vraies ou fausses en fonction des lignes passées en revue
- Une condition est aussi appelée prédicat. Elle est toujours composée de trois membres : deux expressions encadrant un opérateur de comparaison

```
Prédicat = Expression   OPERATEUR DE COMPARAISON   Expression
=>    Prédicat = attribut       OPERATEUR DE COMPARAISON     valeur
      Prédicat = attribut       OPERATEUR DE COMPARAISON     attribut
```

Opérateurs de comparaison

- Opérateurs permettant de construire les prédicats :
  - Comparaison : =, !=, <>, >, >=, <, <=
  - Intervalles : BETWEEN
  - Concordance de caractères : LIKE avec jokers : _,%
  - Gestion des colonnes "NULL" : IS NULL
  - Enumération : IN
  - Négation : NOT IN, NOT LIKE, NOT BETWEEN, IS NOT NULL

- Exemple : employés possédant une commission et un salaire supérieur à 900

```
SELECT ename,
       job,
       sal,
       comm
FROM   emp
WHERE  sal > 900
   AND comm IS NOT NULL
```

Opérateur de comparaison =

- Exemple : Nom des employés, et numéro de service des employés qui sont vendeurs

```
SELECT ename,
       deptno
FROM   emp
WHERE  job = 'SALESMAN'
```

```
ENAME             DEPTNO
---------- ----------
ALLEN                 30
WARD                  30
MARTIN                30
TURNER                30
```

Opérateurs de comparaison : >,<

- La relation d'ordre utilisée dépend du type de données :
  - numérique,
  - alphabétique,
  - chronologique

- Exemple avec les dates : > signifie postérieur, < signifie antérieur

```
SELECT ename, hiredate
FROM   emp
WHERE  hiredate < '1981-01-01' -- format de la date dépend du SGBD
```

- Pour les comparaisons de chaînes : les lettres majuscules et minuscules sont supérieures aux nombres et les majuscules et minuscules ont parfois une importance (dépend des options du SGBD)
  - Exemple du supérieur avec des chaînes de caractères :

```
SELECT  ename
FROM    emp
WHERE   ename > 'martin'
ORDER BY ename ASC

no rows selected
```

```
SELECT  ename
FROM    emp
WHERE   ename > 'MARTIN'
ORDER BY ename DESC;
ENAME
----------
WARD
TURNER
SMITH
SCOTT
MILLER
```

Opérateur de comparaison LIKE

- Exemple avec jokers _ (remplace une lettre) et % (toute chaîne de caractères)
  - Nom des employés ayant AR dans leur nom à partir de la deuxième lettre

```
SELECT ename
FROM   emp
WHERE  ename LIKE ('_AR%')

ENAME
----------
WARD
MARTIN
```

Opérateur de comparaison IN

- Exemple : Employés travaillant dans les services 10 et 20

```
SELECT ename,
       deptno
FROM   emp
WHERE  deptno IN (10,20)

ENAME          DEPTNO
---------- ----------
SMITH              20
JONES              20
CLARK              10
SCOTT              20
KING               10
ADAMS              20
FORD               20
MILLER             10
```

- Exemple avec IN et des chaînes de caractères

```
SELECT deptno,
       dname
FROM   dept
WHERE  loc IN ('DALLAS', 'CHICAGO')
```

Opérateur de comparaison NOT IN

- Exemple : Nom des employés ne travaillant pas dans les services 10 et 20

```
SELECT ename,
       deptno
FROM   emp
WHERE  deptno NOT IN (10, 20)
```

```
ENAME           DEPTNO
---------- ----------
ALLEN               30
WARD                30
MARTIN              30
BLAKE               30
TURNER              30
JAMES               30
```

Opérateurs de comparaison BETWEEN / NOT BETWEEN

- **Opérateur BETWEEN :** Les bornes sont prises en compte lors de la recherche
- Exemple : employés ayant un salaire compris entre 2450 et 3000

```
SELECT ename,
       sal
FROM   emp
WHERE  sal BETWEEN 2450 AND 3000
```

```
ENAME                SAL
---------- ----------
JONES              2975
BLAKE              2850
CLARK              2450
SCOTT              3000
FORD               3000
```

Opérateurs de comparaison
IS NULL

- Une valeur NULL est une valeur inconnue
  - NULL <> 0
  - NULL = non renseigné
  - test par IS NULL et non = NULL
  - Exemple : employés n'étant pas commissionnés

```
SELECT ename,
       deptno
FROM   emp
WHERE  comm IS NULL

ENAME           DEPTNO
---------- ----------
SMITH               20
JONES               20
BLAKE               30
CLARK               10
...
```

Opérateurs de comparaison
IS NULL (suite)

- Remarques :
  - L'utilisation d'une colonne "NULL" dans un calcul nécessite l'utilisation d'une fonction IFNULL (c.f fonctions)
  - Exemple :

```
SELECT   ename,
         sal,
         comm,
         sal + comm,
         sal + IFNULL(comm,0) -- Pour Oracle : NVL(), SQL Server : ISNULL()
FROM emp
```

```
ENAME             SAL       COMM   SAL+COMM SAL+IFNULL(COMM,0)
---------- ---------- ---------- ---------- ---------------
SMITH             800                              800
ALLEN            1600        300       1900            1900
WARD             1250        500       1750            1750
JONES            2975                              2975
MARTIN           1250       1400       2650            2650
BLAKE            2850                              2850
CLARK            2450                              2450
SCOTT            3000                              3000
KING             5000                              5000
TURNER           1500          0       1500            1500
ADAMS            1100                              1100
JAMES             950                               950
...
```

Opérateurs logiques : liaisons de conditions AND

- Les opérateurs logiques permettent de construire des prédicats composés en reliant des prédicats simples
- Plusieurs opérateurs logiques peuvent être utilisés {NOT, AND, OR}, ils sont par défaut évalués dans cet ordre
- Il est conseillé d'utiliser les parenthèses pour imposer l'ordre d'évaluation ou éviter toute ambiguïté
- Liaison par AND
  - relie deux conditions ou prédicats
  - ne renvoie des résultats que lorsque toutes les conditions sont vraies
- Exemple : nom des employés, numéro de service et salaire pour les employés travaillant dans le service 10, n'ayant pas de commission, et ayant un salaire supérieur à 1000

```
SELECT ename,
       deptno,
       sal
FROM   emp
WHERE  deptno = 10
AND    sal > 1000
AND    comm IS NULL
```

Opérateurs logiques : liaisons de conditions OR

- Liaison par OR
  - relie deux conditions
  - renvoie des résultats lorsque l'une des conditions est vraie
- Exemple : nom des employés, et salaire pour les employés n'ayant pas de commission, ou ayant un salaire inférieur à 1000

```
SELECT ename,
       sal
FROM   emp
WHERE  comm IS NULL
       OR sal < 1000
```

Mémento

- De même que pour l'arithmétique, le résultat d'une opération portant sur des nombres est un nombre, le résultat d'un SELECT est une relation

  SELECT ...
  FROM
  WHERE ...

  construit une relation

- On pourra donc utiliser le résultat d'un SELECT dans d'autres requêtes SQL
  - sous-sélections,
  - jointures,
  - UNION ...

Jointures

Produit cartésien

- Produit cartésien
  - Opération entre relations qui combine toutes les occurrences (tuples)
  - Le nombre de lignes résultant est égal à la multiplication des nombres de lignes de chacune des relations
  - Exemple :

```
SELECT *
FROM   dept, emp
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7369	SMITH	CLERK	7902	17-DEC-80	800		20	10	ACCOUNTING	NEW YORK
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30	10	ACCOUNTING	NEW YORK
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30	10	ACCOUNTING	NEW YORK
7566	JONES	MANAGER	7839	02-APR-81	2975		20	10	ACCOUNTING	NEW YORK
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30	10	ACCOUNTING	NEW YORK
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30	10	ACCOUNTING	NEW YORK
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10	10	ACCOUNTING	NEW YORK
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20	10	ACCOUNTING	NEW YORK
7839	KING	PRESIDENT		17-NOV-81	5000		10	10	ACCOUNTING	NEW YORK
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30	10	ACCOUNTING	NEW YORK
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20	10	ACCOUNTING	NEW YORK
7900	JAMES	CLERK	7698	03-DEC-81	950		30	10	ACCOUNTING	NEW YORK
7902	FORD	ANALYST	7566	03-DEC-81	3000		20	10	ACCOUNTING	NEW YORK
7934	MILLER	CLERK	7782	23-JAN-82	1300		10	10	ACCOUNTING	NEW YORK
7369	SMITH	CLERK	7902	17-DEC-80	800		20	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30	20	RESEARCH	DALLAS
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30	20	RESEARCH	DALLAS
7566	JONES	MANAGER	7839	02-APR-81	2975		20	20	RESEARCH	DALLAS
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30	20	RESEARCH	DALLAS
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30	20	RESEARCH	DALLAS
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30	20	RESEARCH	DALLAS
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10	20	RESEARCH	DALLAS
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20	20	RESEARCH	DALLAS
7839	KING	PRESIDENT		17-NOV-81	5000		10	20	RESEARCH	DALLAS
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30	20	RESEARCH	DALLAS
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20	20	RESEARCH	DALLAS
7900	JAMES	CLERK	7698	03-DEC-81	950		30	20	RESEARCH	DALLAS

...

56 rows Selected.

Les jointures

EMPNO	ENAME	DEPTNO
7369	SMITH	20
7499	ALLEN	30
7521	WARD	30

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO

Les jointures (suite)

- **Exemple** : Requête ramenant le nom de l'employé, le numéro, le nom et la localité du service

```
SELECT      d.deptno,
            d.dname,
            e.ename,
            d.loc
FROM        dept d, emp e
WHERE       d.deptno = e.deptno
```

```
    DEPTNO DNAME          ENAME      LOC
---------- -------------- ---------- -------------
        10 ACCOUNTING     CLARK      NEW YORK
        10 ACCOUNTING     KING       NEW YORK
        10 ACCOUNTING     MILLER     NEW YORK
        20 RESEARCH       SMITH      DALLAS
        20 RESEARCH       JONES      DALLAS
        20 RESEARCH       SCOTT      DALLAS
        20 RESEARCH       ADAMS      DALLAS
        20 RESEARCH       FORD       DALLAS
        30 SALES          ALLEN      CHICAGO
        30 SALES          WARD       CHICAGO
        30 SALES          MARTIN     CHICAGO
        30 SALES          BLAKE      CHICAGO
        30 SALES          TURNER     CHICAGO
        30 SALES          JAMES      CHICAGO
```

Les jointures (suite)

- La jointure :
  - Produit cartésien suivi d'une restriction et d'une projection,
  - recherche des données provenant d'au moins deux relations,
  - combine les relations en faisant correspondre les valeurs présentes dans chaque relation
- WHERE :
  - au moins une partie de la condition de recherche doit être une clause de jointure valide,
  - les noms de colonnes qui figurent dans la jointure ne sont pas nécessairement les mêmes
- SELECT :
  - les colonnes qui figurent dans la condition ne sont pas obligatoires dans la clause SELECT

Jointure : Alias

- Un alias ou synonyme local :
  - permet de renommer localement une relation
  - permet de faire référence aux relations de façon abrégée
  - permet une lecture et une écriture aisées des requêtes
  - peut être repris n'importe où dans l'ordre SELECT
  - est nécessaire dans le cas des auto-jointures

- Exemple (SQL1):

```
SELECT      a.deptno,
            b.ename,
            b.sal,
            b.comm
FROM        dept a, emp b
WHERE       a.deptno = b.deptno
   AND      b.comm IS NOT NULL
```

- Règle d'écriture à adopter : préférer des noms d'alias
- compréhensibles (1ères lettres,...) !

```
SELECT      d.deptno,
            e.ename,
            e.sal,
            e.comm
FROM        dept d, emp e
WHERE       d.deptno = e.deptno
   AND      e.comm IS NOT NULL
```

Recommandé (car optimisé)

```
(SQL2)
SELECT a.deptno,
       b.ename,
       b.sal,
       b.comm
FROM   dept a JOIN emp b
       ON a.deptno = b.deptno
WHERE
     b.comm IS NOT NULL
```

```
(SQL2)
SELECT d.deptno,
       e.ename,
       e.sal,
       e.comm
FROM   dept d JOIN emp e
       ON d.deptno = e.deptno
WHERE
     e.comm IS NOT NULL
```

Auto-jointure

- L'auto-jointure :
  - opération de jointure d'une relation à elle-même
  - il faut considérer que la provenance est différente, comme si on avait à faire réellement à deux relations, bien que les données se trouvent physiquement en un seul endroit
  - le synonyme (ou alias) de relation est indispensable
  - elle peut être utilisée dès qu'une ligne fait référence à une autre ligne de la même relation
- Exemple : Nom et salaire des employés ayant le même salaire que 'FORD'

```
SELECT      e2.ename,
            e2.sal
FROM        emp e1, emp e2
WHERE       e1.ename = 'FORD'
AND         e1.sal   = e2.sal
```

```
ENAME             SAL
---------- ---------
SCOTT            3000
FORD             3000
```

```
(SQL2)
SELECT e2.ename, e2.sal
FROM   emp e1 JOIN emp e2
       ON e1.sal = e2.sal
WHERE
      e1.ename = 'FORD'
```

Mémento

- On utilisera une jointure si :
  - les colonnes résultats (select liste + clause group by + clause order by) proviennent de plusieurs tables,
  - ou si les lignes résultats sont plus nombreuses que les lignes des tables origines

    Exemple : Pour chaque employé, liste des employés embauchés avant lui

```
SELECT e1.empno,
       e1.hiredate,
       e2.empno,
       e2.hiredate

FROM  emp e1, emp e2

WHERE      e1.hiredate > e2.hiredate
```

```
(SQL2)
SELECT  e1.empno, e1.hiredate,
        e2.empno, e2.hiredate
FROM    emp e1 JOIN emp e2
    ON e1.hiredate > e2.hiredate
```

  - Dans les autres cas, on préfère d'autres modes d'écriture (sous-sélections) afin d'améliorer la lisibilité et la testabilité

  - 2 relations = 1 condition de jointure au minimum
  - n relations = n-1 conditions de jointure au minimum

Vues

Intérêt d’une vue

- Simplification de requêtes pour des non spécialistes : en masquant les jointures fréquemment utilisées
  Création de résultats intermédiaires pour des requêtes complexes (beaucoup de colonnes, beaucoup de lignes, ou des noms complexes)
- Mise en œuvre de la confidentialité : en cachant aux utilisateurs certaines colonnes ou certaines lignes
- "Sauvegarder" des requêtes fréquemment utilisées

Définition d'une vue

- VUE : table virtuelle => aucune implémentation physique de ses données (résultat d' un SELECT)
- Seule la définition de la vue est enregistrée A chaque utilisation de la vue : le SGBD réactive sa construction
- Vue mono-table
  - Créée à partir d'une table
  - Modifications possibles dans la vue
- Vue multi-tables
  - Créée à partir d' une jointure
  - Aucune modification autorisée

Création/Suppression des vues

- **Create [or replace]** [force | no force] **view <nom-de-vue>**
  [(<liste-alias>)]
  As <requête-select>

  [WITH [CASCADED | LOCAL] CHECK OPTION] ;

  # les options qui ne sont pas en gras sont liées à des aspects plus avancés

- **Drop view <nom-de-vue> ;**

Création/Suppression des vues

- OR REPLACE

  permet le remplacement de la description par la nouvelle requête si la vue existe déjà

- WITH CHECK OPTION (vues actualisables)

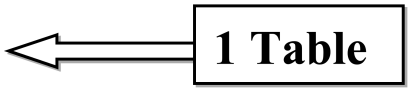
  vérifie, lors de l'insertion ou de la modification de lignes dans la vue, que les lignes insérées ou modifiées sont visualisables dans cette vue

  - LOCAL vérification de l'intégrité au niveau de la vue seule
  - CASCADED vérification au niveau de la vue et des vues dépendantes (option par défaut)

Exemples de création de VUES

- Vue mono-table 1

```
CREATE VIEW enseignant_info AS
SELECT * FROM enseignant
WHERE idDip IN
      (SELECT idDip FROM diplome
       WHERE UPPER(nomDiplome) LIKE '%INFO%');
```



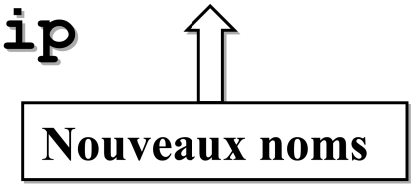
- Vue mono-table 2

```
CREATE VIEW etudiant_scol AS
SELECT idEtu,nomEtu,adrEtu,idDip FROM etudiant;
```

Exemples de création de VUES

- Vue mono-table 3

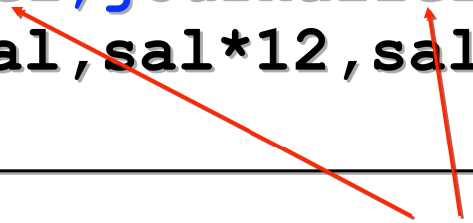
```
CREATE VIEW etudiant_info
(numEtudiant,nomEtudiant,adrEtudiant,dip) AS
SELECT idEtu,nomEtu,adrEtu,idDip
FROM etudiant
WHERE idDip IN
      (SELECT idDip FROM diplome
       WHERE UPPER(nomDiplome) LIKE '%INFO%');
```



Exemples de création de VUES

- Vue mono-table avec colonnes virtuelles

```
CREATE VIEW employe_salaire
(ne,nome,mensuel,annuel,journalier) AS
SELECT idEmp,nomEmp,sal,sal*12,sal/22
FROM employe;
```

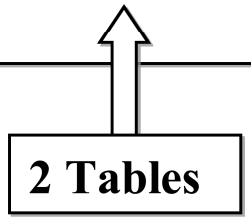


- Pas de modification sur les colonnes virtuelles
- Modifications autorisées sur les colonnes de base → mise à jour instantanée !

Vues multi-tables

- Simplification de requêtes
- Pas de modifications possibles de ce type de vue
- Tables temporaires 'virtuelles' de travail
- Transformation de la présentation des données
  →Schéma externe

```
CREATE VIEW emp_ser(nom_service, nom_employe)
AS
SELECT s.noms,e.nome FROM emp e JOIN service s ON
e.idSer=s.idSer;
```



Exemples de vues multi-tables

- Reconstitution des clients (UNION)

```
CREATE VIEW clients(idCli,nom,….,secteur) AS
SELECT ct.*,'T' FROM clients_toulouse ct
UNION
SELECT cb.*,'B' FROM clients_bordeaux cb
UNION
SELECT cm.*,'M' FROM clients_montpellier cm;
```

- Reconstitution des étudiants (JOINTURE)

```
CREATE VIEW etudiants(idEtu,nom,adresse,
nomstage,entrstage) AS
SELECT e.id,e.nom,e.adr,s.nomS,s.entrS
FROM etudiant e JOIN stage s ON e.id=s.id;
```

Langage de Manipulation de Données (LMD)

- ***INSERT***
- ***UPDATE***
- ***DELETE***

Principe

- Le Langage de Manipulation des Données :
  - INSERT : insertion d'une ou plusieurs lignes dans une table existante
  - DELETE : destruction d'une ou plusieurs lignes d'une table existante
  - UPDATE : mise à jour d'une ou plusieurs lignes d'une table existante

- L'ordre SELECT avec toutes ses fonctionnalités (sauf ORDER BY) peut être utilisé dans les ordres LMD

INSERT

- Pour insérer des n-uplets :

```
INSERT INTO R(A1,A2,...,An) VALUES (v1, v2,... vn)
```

- Donc on donne deux listes : celles des attributs (les A_i) de la table et celle des valeurs respectives de chaque attribut (les v_i)
  - Chaque A_i doit être un attribut de R
  - Les attributs non indiqués restent à **NULL** ou à leur valeur par défaut
  - On doit toujours indiquer une valeur pour un attribut déclaré **NOT NULL**

INSERT (suite)

- Exemple sans nom des colonnes

```
INSERT INTO dept
VALUES (50, null, null)

1 row created.
```

- Exemple avec nom de colonne

```
INSERT INTO dept (deptno)
VALUES (50)

1 row created.
```

- Pour une meilleure évolutivité, il est préférable de nommer les colonnes utilisées

INSERT (suite)

- Exemples insert avec select (plusieurs lignes) :

```
INSERT INTO essai
SELECT *
FROM   emp

14 rows created.

INSERT INTO essai2 (deptno)
SELECT DISTINCT deptno
FROM   emp

3 rows created.
```

- Exemple d'insertion avec des chaînes de caractères :

```
INSERT INTO DEPT (deptno, dname)
VALUES (50, 'COURS')
```

UPDATE

- On modifie une table avec la commande UPDATE :

```
UPDATE R SET A1=v1,A2=v2,...,An=vn
WHERE condition
```

- Contrairement à INSERT, UPDATE s'applique à un ensemble de lignes
  - On énumère les attributs que l'on veut modifier
  - On indique à chaque fois la nouvelle valeur
  - La clause **WHERE** *condition* permet de spécifier les lignes auxquelles s'applique la mise à jour. Elle est identique au WHERE du SELECT
- On ne peut pas violer les contraintes sur la table

UPDATE (suite)

- Exemple sans WHERE : augmentation des salaires de 5 %

```
UPDATE emp
SET    sal = sal * 1.05

14 rows updated.
```

- Exemple avec WHERE : augmentation du salaire des vendeurs

```
UPDATE emp
SET    sal = sal * 1.05
WHERE  job = 'SALESMAN'

4 rows updated.
```

UPDATE (suite)

- Exemple : sous-requête synchronisée

```
UPDATE emp e
SET    sal = (SELECT AVG(e2.sal)
              FROM   emp e2
              WHERE  e.deptno = e2.deptno)
```

- Exemple : jointure (sans alias, ne fait pas partie de la norme)

```
UPDATE emp
SET    Sal = Sal + 100
FROM   emp
  JOIN dept ON  emp.deptno = dept.deptno
WHERE  dept.loc = 'NEW YORK'
```

DELETE

- On détruit une ou plusieurs lignes dans une table avec la commande DELETE :

  `DELETE FROM R WHERE` *`condition`*

- C'est la plus simple des commandes de mise-à-jour puisque elle s'applique à des lignes et pas à des attributs
- Comme précédemment, la clause WHERE *condition* est identique au WHERE du SELECT (=restriction des nombres de lignes impliquées)

DELETE (suite)

- Exemple sans WHERE : suppression de tous les services

```
DELETE FROM dept
4 rows deleted
```

- Exemple avec WHERE : suppression des employés ayant une commission à zéro

```
DELETE FROM emp
WHERE  comm = 0
1 row deleted
```

DELETE (suite)

- Exemple avec WHERE et SELECT : suppression des employés travaillant à DALLAS

```
DELETE    FROM emp
WHERE     deptno =
          (SELECT deptno
         FROM   dept
         WHERE  loc = 'DALLAS')
5 rows deleted.
```

comme pour les update, on peut utiliser la notion de jointure

Syntaxe simplifiée

```
INSERT   INTO  [ schema. ]  { table | view }
                [ ( colonne [ , colonne ] ...]
{ VALUES       ( expr [ , expr ] ... ) | Ordre SELECT }


UPDATE                [ schema. ] { table | view } [ alias ]
SET             { ( colonne [ , colonne] ...) = (Ordre SELECT)
                | colonne = {expr | (Ordre SELECT)}}
                [ ,{ (colonne [, colonne] ...) = (Ordre SELECT)
                | colonne = {expr | (Ordre SELECT)}}]...
[WHERE          condition ]


DELETE
FROM            [schema.]{table | view} [alias]
[WHERE          condition]
```

Langage de Définition de Données (LDD)

- *Objectifs LDD*
- *Création de tables*
- *Modification de tables*
- *Suppression de tables*

Objectifs LDD

- Objectifs :
  - `CREATE` (création) :
    - Tables avec des colonnes de types différents et contraintes (`CREATE TABLE`)
  - `ALTER` (modification) :
    - Tables et contraintes (`ALTER TABLE`)
  - `DROP` (suppression) :
    - Tables et contraintes (`DROP TABLE`)

Création de tables

- Syntaxe :
  CREATE TABLE *nom_table* (*nom_champ*1 *type_champ*1,
  ***nom_champ*2 *type_champ*2,**
  ...
  ***nom_champ*n *type_champ*n) ;**
- Exemple : création de la table des services

```
CREATE TABLE DEPT
(
 DEPTNO                NUMERIC(2) NOT NULL,
 DNAME                 VARCHAR(14),
 LOC                   VARCHAR(13) DEFAULT 'PARIS'
);
```

Nom de colonne	Type de donnée	Propriété (défaut = NULL)	Valeur par défaut (facultatif)
DEPTNO DNAME LOC	NUMERIC(2) VARCHAR(14) VARCHAR(13)	NOT NULL	 DEFAULT 'PARIS'

Création de tables : choix des champs

- Avant de créer une table, il faut choisir les champs que l'on va utiliser. Pour chaque champ il faut choisir un *nom* et un *type*.
- Les champs peuvent être des types suivants : *numérique* (entier ou rationnel), *date et heure*, *chaîne de caractères*, *texte*, *blob*, *énuméré*, *ensemble*.
- Types possibles :
  - MySQL : CHAR(M), VARCHAR(M), INT[(M)] ou INTEGER[(M)], FLOAT[(M,D)], DOUBLE[(M,D)] ou REAL [(M,D)], DECIMAL[(M[,D])] ou NUMERIC[(M[,D])], DATE, TIME, DATETIME, TIMESTAMP[(M)], BLOB, ENUM, SET,...
  - Oracle : CHAR(M), VARCHAR2(M), NUMBER[(M[,D])], DATE, TIMESTAMP[(M)], BLOB, CLOB,BFILE,...
  - SQL SERVER : BIT, CHAR(M), VARCHAR(M), TEXT, DECIMAL[(M[,D])], FLOAT, INT, MONEY, REAL, DATETIME, SMALLDATETIME, TIMESTAMP[(M)], IMAGE, BINARY(M), UNIQUEIDENTIFIER,...

Création de tables : choix des champs

- Avant de créer une table, il faut choisir les champs que l'on va utiliser. Pour chaque champ il faut choisir un *nom* et un *type*.
- Les champs peuvent être des types suivants : *numérique* (entier ou rationnel), *date et heure*, *chaîne de caractères*, *texte*, *blob*, *énuméré*, *ensemble*.
- Types possibles :
  - MySQL : CHAR(M), VARCHAR(M), INT[(M)] ou INTEGER[(M)], FLOAT[(M,D)], DOUBLE[(M,D)] ou REAL [(M,D)], DECIMAL[(M[,D])] ou NUMERIC[(M[,D])], DATE, TIME, DATETIME, TIMESTAMP[(M)], BLOB, ENUM, SET,...
  - Oracle : CHAR(M), VARCHAR2(M), NUMBER[(M[,D])], DATE, TIMESTAMP[(M)], BLOB, CLOB,BFILE,...
  - SQL SERVER : BIT, CHAR(M), VARCHAR(M), TEXT, DECIMAL[(M[,D])], FLOAT, INT, MONEY, REAL, DATETIME, SMALLDATETIME, TIMESTAMP[(M)], IMAGE, BINARY(M), UNIQUEIDENTIFIER,...

Création de tables (suite)

- CREATE à l'aide de SELECT
  - Exemple : création de la table dept_20. La structure de cette table est identique à celle de la table DEPT

```
CREATE TABLE dept_20 AS
  SELECT * FROM dept
  WHERE deptno = 20
```

Création de contraintes d'intégrité (1)

- Moyen permettant de garantir que les modifications apportées à la base ne pourront en aucun cas la rendre incohérente
- Différents types de contraintes d'intégrité :
  - Définie lors de la création des tables
    - les contraintes de domaine
      - type de colonne, valeur par défaut, ensemble de valeurs, conditions qu'une valeur doit remplir ; caractère obligatoire ou non des colonnes (NULL ou NOT NULL).
    - les contraintes de clé unique
      - garantissent qu'une même valeur ne peut se trouver sur plus d'une ligne.
    - les contraintes de clé primaire
      - colonne ou groupe de colonnes choisis pour identifier de façon unique chacune des occurrences de tables, et référençable par une clé étrangère.
    - les contraintes référentielles (clé étrangère)
      - garantissent qu'une colonne ou un groupe de colonnes existe dans une autre entité.
  - Définies par trigger (déclencheur)
    - Contraintes temporelles : salaire ne peut pas baisser…
    - Contraintes avec agrégats : ne porte pas sur un attribut ou un tuple, mais plusieurs tuples ou même toute la table :
      - Exemple : « Il doit y avoir autant de départements localisés à Paris que de départements à Londres »
      - Cette contrainte ne peut être vérifiée que lorsque « tous les départements sont insérés »

Création de contraintes d'intégrité (2)

- Respect des contraintes d'intégrité :
  - Lors de chaque accès en mise à jour (ajout, modification, suppression), le SGBD doit vérifier les contraintes d'intégrité
  - Elles sont en général définies lors de la création des tables, en donnant des précisions sur les attributs concernés
  - Certaines font l'objet de procédures particulières appelées triggers (déclencheurs) qui sont exécutées lors de l'accès en MAJ aux données

Création de contraintes : Contraintes de domaine

- Les types de données :
  - `CHAR, VARCHAR, NUMERIC, ...`
- Le caractère obligatoire ou facultatif d'une colonne :
  - `NULL` ou `NOT NULL`
- La clause `DEFAULT` :
  - valeur par défaut pour une colonne, cette valeur est utilisée en création (`insert`) ou modification (`update`) si l'utilisateur ne renseigne pas cette colonne,
  - peut être une fonction (Ex. : `CURRENT_DATE`).
- La clause `CHECK` (ou contraintes de validation) :
  - Contrôle de valeur effectué pour toute exécution d'une commande `update`, `insert` ou `delete` sur chaque ligne de la table
  - Si contrôle est négatif, ordre SQL annulé
  - Contrainte sur colonne :
    - `CONSTRAINT Ck_LIG_CDE check (qte_cdee > 0)`
  - Contrainte sur table (plusieurs colonnes impliquées) :
    - `CONSTRAINT CK_LIG_CDE check (qte_cdee >= qte_livree)`

Création de contraintes : Contraintes de domaine (suite)

- Exemple : contraintes déclaratives

```
CREATE TABLE emp1
(
     empno    NUMERIC(4) NOT NULL,
     ename    VARCHAR(10),
     job      VARCHAR(9) CHECK (job in ('SALESMAN','CLERK','MANAGER')),
     mgr      NUMERIC(4),
     hiredate DATE DEFAULT CURRENT_DATE,
     sal      NUMERIC(7,2) CHECK (sal > 300 AND sal < 9000),
     comm     NUMERIC(7,2) CHECK (comm IS NULL OR comm<=sal/2),
     deptno   NUMERIC(2) NOT NULL
);
```

- Remarque : il est conseillé de nommer les contraintes afin de simplifier le décodage des messages d'erreurs, la gestion des activations ou désactivations des contraintes. Dans ce cas, il est d'usage de les positionner à la fin du `CREATE TABLE`

Création de contraintes : Contraintes de domaine (suite)

- Exemple : contraintes déclaratives

```
CREATE TABLE emp1
(
    empno    NUMERIC(4) NOT NULL,
    ename    VARCHAR(10),
    job      VARCHAR(9),
    mgr      NUMERIC(4),
    hiredate DATE DEFAULT CURRENT_DATE,
    sal      NUMERIC(7,2),
    comm     NUMERIC(7,2),
    deptno   NUMERIC(2) NOT NULL,
    CONSTRAINT ck_emp1_job CHECK (job in
      ('SALESMAN','CLERK','MANAGER')),
    CONSTRAINT ck_emp1_sal CHECK (sal > 300 AND sal < 9000),
    CONSTRAINT ck_emp1_comm CHECK (comm IS NULL OR comm<=sal/2)
);
```

Création de contraintes : Contraintes de domaine (suite)

- **Exemple** : Insertion d'une ligne ne respectant pas la contrainte sur la colonne JOB.

La contrainte portant sur la colonne job n'est pas nommée, le SGBD génère automatiquement un nom unique.

```
INSERT INTO emp1
VALUES (7600, 'TOTO', 'INCONNU', 10, null, 0, 0, 10)
ERROR at line 1:
check constraint (SYS_C0011195) violated
```

- **Exemple** : Insertion d'une ligne ne respectant pas la contrainte sur la colonne salaire (contrainte nommée)

```
INSERT INTO emp1 VALUES
(10, 'toto', 'CLERCK', null, null, 0, 0, 10)
ERROR at line 1:
check constraint (COURS1.CHK_EMP1_SAL) violated
```

- **Remarque** : aucune erreur ne sera soulevée dans MySQL, car ce SGBD ne gère pas les contraintes CHECK !!!

Création de contraintes : Contraintes de clé unique

- Les contraintes de clé unique (`UNIQUE`) :
  - permettent d'assurer l'unicité d'une colonne ou d'un groupe de colonnes,
  - les valeurs NULL sont autorisées,
  - une clé unique ne peut pas être référencée par une clé étrangère

Création de contraintes : Contraintes de clé unique (suite)

- Exemple :

```
CREATE TABLE emp1
(
   empno      NUMERIC(4)   NOT NULL,
   ename      VARCHAR(10)  UNIQUE,
   job        VARCHAR(9),
   mgr        NUMERIC(4),
   Hiredate   DATE,
   sal        NUMERIC(7,2),
   comm       NUMERIC(7,2),
   deptno     NUMERIC(2) NOT NULL
);
```

Création de contraintes : Contraintes de clé unique (suite)

- Exemple : **OU MIEUX**

```
CREATE TABLE emp1
(
   empno       NUMERIC(4)   NOT NULL,
   ename       VARCHAR(10),
   job         VARCHAR(9),
   mgr         NUMERIC(4),
   Hiredate    DATE,
   sal         NUMERIC(7,2),
   comm        NUMERIC(7,2),
   deptno      NUMERIC(2) NOT NULL,
   CONSTRAINT uq_emp1_ename UNIQUE(ename)
);
```

Création de contraintes : Contraintes de clé primaire

- Les contraintes de clé primaire (`PRIMARY KEY`) :
  - toute table doit disposer d'une clé primaire (unique, mais pouvant être composée de plusieurs champs),
  - la norme SQL impose que toutes les colonnes d'une clé primaire soient obligatoires (la ou les colonne(s) est/sont forcée(s) à NOT NULL),
  - unicité des valeurs de la clé
  - clé généralement référencée par des clés étrangères.

Création de contraintes : Contraintes de clé primaire (suite)

- Exemple :

```
CREATE TABLE emp1
(
   empno       NUMERIC(4) NOT NULL PRIMARY
    KEY,
   ename       VARCHAR(10),
   job         VARCHAR(9),
   mgr         NUMERIC(4),
   hiredate    DATE,
   sal         NUMERIC(7,2),
   comm        NUMERIC(7,2),
   deptno      NUMERIC(2)        NOT NULL
)
```

Création de contraintes : Contraintes de clé primaire (suite)

- Exemple : **OU MIEUX**

```
CREATE TABLE emp1
(
   empno        NUMERIC(4) NOT NULL,
   ename        VARCHAR(10),
   job          VARCHAR(9),
   mgr          NUMERIC(4),
   hiredate     DATE,
   sal          NUMERIC(7,2),
   comm         NUMERIC(7,2),
   deptno       NUMERIC(2)  NOT NULL,
   CONSTRAINT   pk_emp1 PRIMARY KEY (empno)
)
```

Création de contraintes : Contraintes de clé étrangère

- Les contraintes de clé étrangère (`FOREIGN KEY`) :
  - une colonne ou un groupe de colonnes qui référencent la clé primaire d'une autre table,
  - la valeur de la clé étrangère doit :
    - exister dans la table référencée (valeur de la clé étrangère = une des valeurs de la clé primaire),
    - ou bien être NULL (sauf si elle est clé étrangère et clé primaire à la fois !).
  - une clé étrangère ne peut référencer une table d'une base distante.
  - **Le type de la clé étrangère doit correspondre à celui de la clé primaire**

Création de contraintes : Contraintes de clé étrangère (suite)

- Exemple :

Implique que DEPTNO soit clé primaire dans DEPT et que la table DEPT soit déjà créée !

```
CREATE TABLE emp1
(
    empno           NUMERIC(4)      NOT NULL,
    ename           VARCHAR(10),
    job             VARCHAR(9),
    mgr             NUMERIC(4),
    hiredate        DATE,
    sal             NUMERIC(7,2),
    comm            NUMERIC(7,2),
    deptno          NUMERIC(2)      NOT NULL,
    CONSTRAINT      pk_emp1 PRIMARY KEY (empno),
    CONSTRAINT      fk_emp1_deptno FOREIGN KEY(deptno)
      REFERENCES dept(deptno),
    CONSTRAINT      fk_emp1_mgr FOREIGN KEY(mgr) REFERENCES
      emp1(empno)
);
```

Création de contraintes : Contraintes de clé étrangère (suite)

- **Exemple** : Insertion d'une ligne ne respectant pas la contrainte de clé étrangère `fk_emp1_deptno`

```
INSERT INTO emp1
VALUES (10, 'EMPLOYE 1','MANAGER', null, now(), 8000, null, 70);
```

ERROR: insert or update on table "emp1" violates foreign key constraint "fk_emp1_deptno"
DETAIL: Key (deptno)=(70) is not present in table "dept".

- **Exemple** : Insertion d'une ligne ne respectant pas la contrainte de clé étrangère `fk_emp1_mgr`

```
INSERT INTO emp1
VALUES (20, 'EMPLOYE 2', 'SALESMAN', 30, current_date,
   4000,null,10);
```

ERROR: insert or update on table "emp1" violates foreign key constraint "fk_emp1_mgr"
DETAIL: Key (mgr)=(30) is not present in table "emp1".

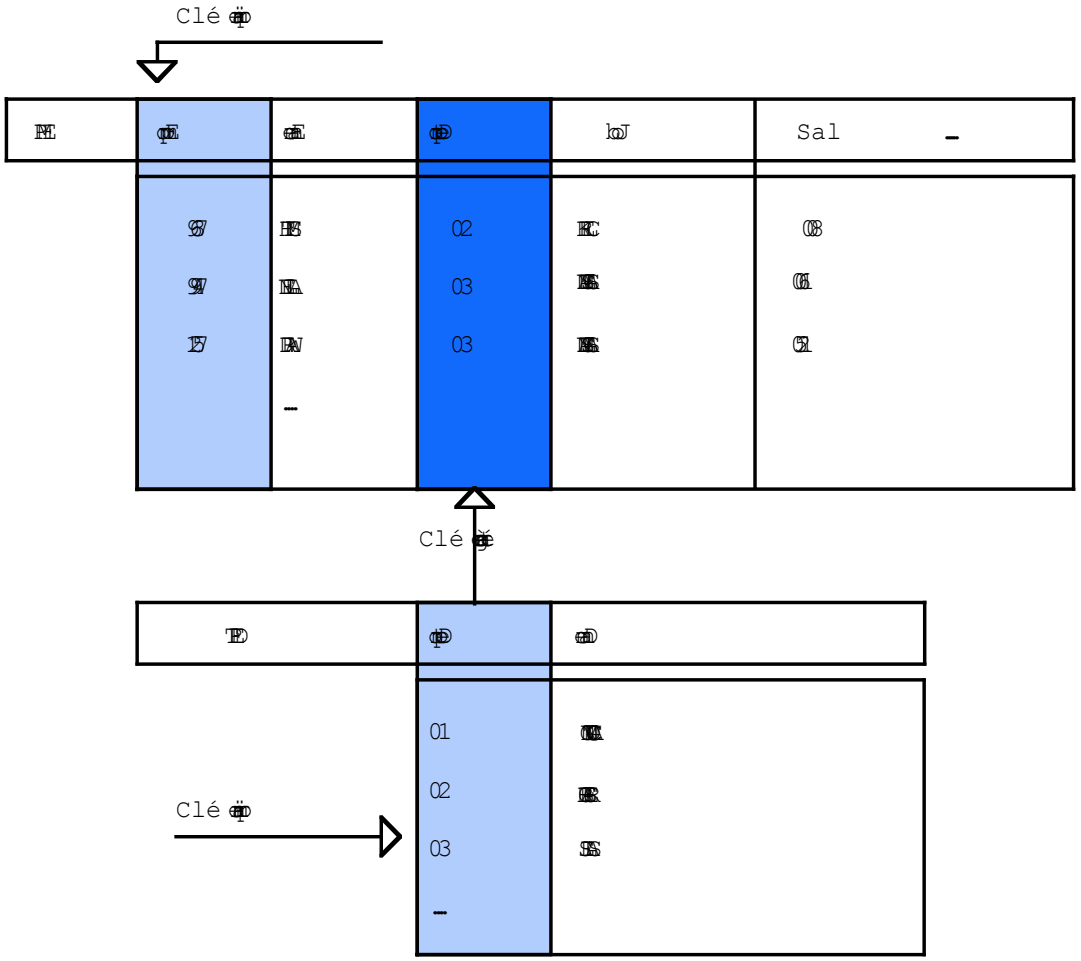
Contrainte de clé étrangère : ON DELETE CASCADE, ON UPDATE CASCADE

- Exemple :

```
CREATE TABLE emp1
(
    empno         NUMERIC(4)       NOT NULL,
    ename         VARCHAR(10),
    job           VARCHAR(9),
    mgr           NUMERIC(4),
    hiredate      DATE,
    sal           NUMERIC(7,2),
    comm          NUMERIC(7,2),
    deptno        NUMERIC(2)   NOT NULL,
    CONSTRAINT    pk_emp1 PRIMARY KEY (empno),
    CONSTRAINT    fk_emp1_deptno FOREIGN KEY(deptno)
      REFERENCES dept(deptno) ON DELETE CASCADE ON UPDATE
      CASCADE,
    CONSTRAINT    fk_emp1_mgr FOREIGN KEY(mgr) REFERENCES
      emp1(empno)
);
```

=> Si on supprime un service, de la table DEPT, référencé dans le table EMP, les lignes de table EMP référençant ce service sont aussi supprimées.

Création de contraintes :
Clé primaire et clé étrangère



Création de contraintes : Clé primaire et clé étrangère (suite)

- Lors d'une insertion :
  - Unicité de la clé primaire et des clés uniques ;
  - Pour chacune des clés étrangères, existence d'une occurrence correspondante dans la table référencée ;
  - Cohérence des valeurs avec leur type (`NUMERIC`, `VARCHAR`,...), le caractère obligatoire ou non (`NULL` ou `NOT NULL`), les conditions éventuelles.
- Lors d'une suppression :
  - S'il n'existe pas de clé étrangère dans d'autres tables dont la valeur correspond à l'une des clés primaires supprimées, la suppression est effectuée ;
  - Sinon,
    - soit la suppression est rejetée (`ON DELETE RESTRICT`)
    - Soit toutes les lignes référençant l'une des clés primaires supprimées sont aussi supprimées (suppression en CASCADE => `ON DELETE CASCADE`).
    - Soit les clés étrangères sont mises à NULL (`ON DELETE SET NULL`)

Création de contraintes : Clé primaire et clé étrangère (suite)

- Lors d'une mise à jour :
  - Cohérence des valeurs avec leur type (`NUMERIC`, `VARCHAR`,...), le caractère obligatoire ou non (`NULL` ou `NOT NULL`), les conditions éventuelles.
  - Pour les mises à jour de clés étrangères, on devra réaliser le même contrôle que lors d'une insertion ;
  - Pour les mises à jour d'une clé primaire,
    - S'il n'existe pas de clé étrangère dans d'autres tables dont la valeur correspond à l'une des clés primaires mises à jour, la mise à jour est effectuée ;
    - Sinon :
      - Soit la mise à jour est rejetée (`ON UPDATE RESTRICT` : interdiction si clé utilisée)
      - Soit la mise à jour est répercutée sur toutes les lignes référençant la clé primaire modifiée (`ON UPDATE CASCADE`).
      - Soit les valeurs de la clé étrangère sont remplacées par NULL (`ON UPDATE SET NULL`)

NULL et clé étrangère

- La contrainte de clé unique ne prend pas en compte les valeurs NULL (NULL n'est pas considérée comme une valeur !)
- Clés étrangères : la contrainte `FOREIGN KEY` n'est pas contrôlée pour une clé étrangère comportant la valeur NULL, y compris dans le cas d'une clé concaténée dont l'une des valeurs est NULL.

Clé étrangère (Col1,Col2)

Col 1	Col 2	Col 3
X1	Y1	
X2	Y2	
X3	NULL	
NULL	NULL	
NULL	Y3	
X3	***Y3***	

Invalide

Clé primaire (Col1,Col2)

Col 1	Col 2
X1	Y1
X2	Y2

Modification de table : ALTER TABLE

- `ALTER TABLE` : permet de modifier la structure initiale d'une table
  - Ajout de colonnes,
  - Modification de la valeur par défaut,
  - Ajout de contraintes,
  - Activation, ou suppression de contraintes,
- Exemple : Changement de la définition de la colonne DNAME de DEPT

```
ALTER TABLE dept ALTER dname TYPE VARCHAR(20);
ALTER TABLE dept ALTER dname SET NOT NULL;
ALTER TABLE dept ALTER dname DROP NOT NULL;
-- Commande ALTER non normalisée (dépend du SGBD).
   Ex. avec MySQL :
     ALTER TABLE dept CHANGE dname
     dname VARCHAR(20) NOT NULL;
```

- Exemple : ajout d'une colonne à la table DEPT

```
ALTER TABLE dept
ADD COLUMN date_creation  DATE NULL
```

Modification de table : `ALTER TABLE` (suite)

- Remarques :
  - Toutes les modifications de structure ne sont pas possibles : il faut respecter le contenu des tables et les contraintes existantes sur les tables (clés étrangères,…).
  - Sur l'exemple de rajout d'une contrainte NOT NULL sur la colonne DNAME : s'il y avait eu des noms de service (DNAME) non renseignés, la contrainte NOT NULL n'aurait pas été valide.

Modification de table : ALTER TABLE (suite)

- Exemple : ajout d'une contrainte de clé primaire à la table DEPT

```
ALTER TABLE dept
ADD CONSTRAINT pk_dept PRIMARY KEY (deptno);
```

- Exemple : ajout d'une contrainte de clé étrangère à la table EMP

```
ALTER TABLE emp
ADD CONSTRAINT fk_emp_deptno FOREIGN KEY
  (deptno) REFERENCES dept(deptno);
```

- Exemple : ajout d'une contrainte CHECK salaire > commission

```
ALTER TABLE emp
ADD CONSTRAINT ck_emp_sal CHECK (SAL > COMM);
```

Modification de table : ALTER TABLE (suite)

- **Remarques** :
  - Il est préférable de créer les contraintes de clé étrangère avec la commande `ALTER TABLE`, i.e. après la création des tables, plutôt que d'utiliser les contraintes déclaratives dans les `CREATE TABLE`. Si les clés étrangères sont créées en fin de script, il ne sera pas nécessaire de respecter l'ordre de création des tables.
  - Si un enregistrement ne satisfait pas la contrainte de type CHECK, celle-ci est rejetée.

```
ALTER TABLE emp
    ADD CONSTRAINT ck_emp_sal
    CHECK (SAL > COMM);
ERROR : CHECK (SAL > COMM)
                *
```

  - Dans certains SGBD, il est possible de rejeter les lignes ne satisfaisant pas les contraintes dans une table d'exception (Oracle, SQL Server)

Suppression de table : DROP TABLE

- Exemple : suppression de la table DEPT

```
DROP TABLE dept;
NOTICE:  constraint fk_emp_deptno on table emp depends on table dept
ERROR:  cannot drop table dept because other objects depend on it
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

=> **Nécessité de supprimer les tables dans l'ordre adéquat (inverse de leur création) afin de respecter les contraintes clé primaire / clé étrangère (de même pour les enregistrements !)**

Ou alors utiliser l'option CASCADE qui supprime les clés étrangères dépendantes : `DROP TABLE dept CASCADE;`

- Impossibilité de supprimer une contrainte avec un DROP. Il faut utiliser ALTER TABLE … DROP …
  - Exemple : `ALTER TABLE emp DROP CONSTRAINT fk_emp_deptno;`