Bases de données
SQL

E.Coquery

emmanuel.coquery@liris.cnrs.fr

SQL

- Un langage concret interagir avec le modèle relationnel :
  - Un langage de manipulation de données.
  - Un langage de description de données.
  - Un langage pour administrer la base, gérer les contrôles d'accès.
- Origine : IBM, dans les années 70.
- Standards :
  - SQL-87 : 1987 (ISO)
  - SQL-2 : 1992 (ANSI)
  - SQL-3 : 1999
  - SQL-2003
  - SQL-2006
- Différences avec la théorie :
  - possibilités de doublons ;
  - possibilité d'ordonner le résultat des requêtes ;
  - notion de valeur non définie.

Plan

Interrogation simple

SELECT att_1, att_2, ...
FROM *nom_table* ;

- Récupérer les valeurs contenus dans la table `nom_table`, en ne gardant que les attributs `att1, att2, ...`
- En algèbre relationnelle :
  $\pi_{att_1,att_2,...}(nom\_table)$

On peut remplacer att_1, att_2, ... par $*$ pour utiliser tous les attributs.

Exemple

Schéma :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner le nom et la fonction de chaque employé :

- `SELECT Nom,Fonction FROM Employe ;`
- $\pi_{Nom,Fonction}(Employe)$

Exemple 2

Schéma :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner les informations sur chaque employé :

- `SELECT * FROM Employe ;`

- *Employe*

mot clé DISTINCT

Le mot clé DISTINCT permet d'éliminer les doublons dans le résultat.

Exemple :

Donner les différentes fonctions occupées dans l'entreprise :

- `SELECT DISTINCT Fonction FROM Employe ;`

Sélections (de lignes)

SELECT att_1, att_2, ...
FROM *nom_table*
WHERE *condition*

- La clause WHERE spécifie les lignes à sélectionner grâce à la *condition*.
- En algèbre relationnelle :
  $\pi_{att_1,att_2,...}(\sigma_{condition}(nom\_table))$

Conditions du WHERE

Expressions simples :

- Comparaisons (=, !=, <, <=, >, >=)
- entre un attribut et une constante ou un autre attribut
- différents types de données utilisés pour les constantes :
  - nombres : 1, 1980, 1.5
  - chaînes de caractères : 'Martin', 'directeur'
  - dates : '1980-06-18'
    - le formattage des dates peut varier

Combinaison d'expressions via :

- le 'et', $\wedge$: AND
- le 'ou', $\vee$: OR

Exemple

Schéma :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés dont la date d'embauche est antérieure au 1^{er} janvier 1999 :

- ```
  SELECT Nom
  FROM Employe
  WHERE Embauche < '1999-01-01' ;
  ```
- $\pi_{Nom}(\sigma_{Embauche<'1999-01-01'}(Employe))$
  ```

Exemple 2

Schéma :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés dont la date d'embauche est antérieure au 1^{er} janvier 1999 et touchant au moins 30000 euros de salaire :

- ```
  SELECT Nom
  FROM Employe
  WHERE Embauche < '1999-01-01'
  AND Salaire >= 30000 ;
  ```
- $\pi_{Nom}(\sigma_{Embauche<'1999-01-01' \wedge Salaire \geq 30000}(Employe))$
  ```

Autres conditions

- L'opérateur IN permet de spécifier un ensemble de valeur possibles :
  - Quels sont les employés qui sont directeur ou ingénieur ?

```
SELECT Nom, Fonction
FROM Employe
WHERE Fonction IN ('ingenieur','directeur') ;
```

- L'opérateur BETWEEN ... AND permet de spécifier un intervalle de valeurs :
  - Quels employés gagnent entre 25000 et 30000 euros ?

```
SELECT Nom, Salaire
FROM Employe
WHERE Salaire BETWEEN 25000 AND 30000 ;
```

  - Attention à ne pas confondre le AND du BETWEEN avec celui qui correspond au $\wedge$.

Autre exemple

Quels sont les employés directeur ou ingénieur, embauchés entre le 1^{er} janvier 1990 et le 31 décembre 1999 gagnant moins de 32000 euros ?

```
SELECT Nom, Embauche, Fonction, Salaire
FROM Employe
WHERE Fonction IN ('ingenieur','directeur')
AND Embauche BETWEEN '1990-01-01' AND '1999-12-31'
AND Salaire < 32000 ;
```

condition, connecteur $\wedge$

Valeurs non définies

En pratique, il est possible d'avoir des valeurs non définies.

- Elles sont représentées par le mot clé NULL.
- On peut tester si une valeur n'est pas définie grâce à la condition IS NULL (ou au contraire IS NOT NULL)

Schéma : Batiment(Num_bat, Nom_bat, Ent_princ, Ent_Sec)

- Les bâtiments qui n'ont pas d'entrée secondaire auront une valeur NULL pour l'attribut Ent_Sec.
- La requête suivante indique les bâtiments n'ayant pas d'entrée secondaire :

```
SELECT *
FROM Batiment
WHERE Ent_sec IS NULL ;
```

Tri du résultat d'une requête

En pratique, il peut être intéressant de trier le résultat d'un requête.

SELECT att_1, att_2, ...
FROM *nom_table*
WHERE *condition*
ORDER BY att_i, att_j, ...

- Le résultat de la requête est trié par ordre croissant sur l'attribut att_i
- En cas d'égalité entre deux lignes au niveau de l'attribut att_i, on utilise l'attribut att_j, etc ...
- Dans un ORDER BY, il est possible de faire suivre le nom d'un attribut par ASC ou DESC pour indiquer un ordre croissant ou décroissant.

Exemple

Schéma :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner le nom des employés du département numéro 20, en triant le résultat par salaire décroissant, puis par nom (croissant) :

```
SELECT Nom
FROM Employe
WHERE Num_dept=20
ORDER BY Salaire DESC, Nom ;
```

Requêtes sur plusieurs tables

SELECT att_1, att_2, ...
FROM nom_table_1, nom_table_2, ...
WHERE *condition*
ORDER BY att_i, att_j, ...

- Il est possible d'utiliser plusieurs tables dans une requête.
- Cela correspond à effectuer un produit cartésien entre les différentes tables.
- Si un attribut est présent dans plusieurs tables utilisées, on doit l'écrire *nom_table.att*

Jointures naturelles

On peut remplacer la virgule par NATURAL JOIN : SELECT att_1, att_2, ...
FROM nom_table_1 NATURAL JOIN nom_table_2, ...
WHERE *condition*
ORDER BY att_i, att_j, ...

Jointure naturelle sur les relations $R(A_1, A_2, B_1, B_2)$ et $S(C_1, C_2, B_1, B_2)$, équivalent à :

SELECT A_1, A_2, $R.B_1$, $S.B_2$, C_1, C_2
FROM R, S
WHERE $R.B_1$=$S.B_1$ AND $R.B_2$=$S.B_2$

Exemple

Schéma :
Batiment(Num_bat, Nom_bat, Ent_princ, Ent_Sec)
Departement(Num_dept, Nom_dept, Num_bat)

Donner les départements avec leur bâtiments :

- *Departement* $\bowtie$ *Batiment*
- ```
  SELECT Num_dept, Nom_dept, Batiment.Num_bat,
      Nom_bat, Ent_princ, Ent_sec
  FROM Departement, Batiment
  WHERE Departement.Num_bat = Batiment.Num_bat ;
  ```
  ```

Renommages

Il est parfois utile de renommer des tables :

SELECT att_1, att_2, ...
FROM nom_table_1 $nouveau\_nom_1$,
nom_table_2 $nouveau\_nom_2$, ...
WHERE *condition*
ORDER BY att_i, att_j, ...

- Indication des renommage dans le FROM.
- Les anciens noms indiqués dans le FROM ne peuvent pas être utilisés dans les autres parties de la requête.
- Utile lorsque l'on veut effectuer des jointures ou des produits cartésiens d'une table avec elle-même.

Exemple

Schema :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner les noms et la fonction des employés avec le nom de leur supérieur hiérarchique.

- $\pi_{Nom,Superieur,Fonction}(\sigma_{Num=Num\_sup}(\pi_{Nom,Num\_sup,Fonction}(Employe) \times \pi_{Superieur,Num}(\rho_{Nom/Superieur}(Employe))))$

- ```
  SELECT Employe.Nom, Employe.Fonction,
         Chef.Nom Superieur
  FROM Employe, Employe Chef
  WHERE Chef.Num = Employe.Num_sup ;
  ```
  ```

Exemple 2

Schema :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés, donnés avec leur salaire, qui gagnent moins que Bellot ?

```
SELECT Employe.Nom, Employe.Salaire
FROM Employe, Employe bel
WHERE Employe.Salaire < bel.Salaire
AND bel.Nom = 'Bellot' ;
```

Sous-requêtes

Il est possible d'utiliser le résultat d'une requête dans une autre requête.

- Augmentation de la puissance d'expression du langage.
- Les sous-requêtes sont utilisables dans les parties
  - WHERE
  - FROM (à condition de renommer le résultat)
  - SELECT (à condition que pour chaque ligne sélectionnée par la requête principale, on ne selectionne qu'une ligne dans la sous-requête).

- En cas de conflit sur les nom, c'est la déclaration la plus proche qui est utilisée.

Exemple

Si la sous-requête renvoie un résultat simple sur une ligne :

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés ayant la même fonction que 'Jones' ?

```
SELECT Nom
FROM Employe
WHERE Fonction =
       (SELECT Fonction
        FROM Employe
        WHERE Nom='Jones') ;
```

Exemple : Sous-requête liée à la requête principale

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés qui ne travaillent pas dans le même département que leur supérieur ?

```
SELECT Nom
FROM Employe Emp
WHERE Num_dept !=
        (SELECT Num_dept
         FROM Employe
         WHERE Emp.Num_sup = Num) ;
```

Sous-requêtes renvoyant plusieurs lignes

Opérateurs permettant d'utiliser de telles sous-requêtes :

- *a* IN (*sous_requete*)
  - vrai si *a* apparaît dans le résultat de *sous_requete*.
- *a* □ ANY (*sous_requete*)
  où □ peut être $\{=, <, >, <=, >=\}$
  - vrai si il existe un *b* parmi les lignes renvoyées par *sous_requete* tel que $a \square b$ soit vrai.
- *a* □ ALL (*sous_requete*)
  où □ peut être $\{=, <, >, <=, >=\}$
  - vrai si pour toutes les lignes *b* renvoyées par *sous_requete*, $a \square b$ est vrai.
- EXISTS (*sous_requete*)
  - vrai si le résultat de *sous_requete* n'est pas vide.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés, donnés avec leur salaire, gagnant plus que tous les employés du département 20 ?

```
SELECT Nom, Salaire
FROM Employe
WHERE Salaire > ALL (SELECT Salaire
                     FROM Employe
                     WHERE Num_dept = 20) ;
```

Exemple 2

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés qui ont un subalterne ?

```
SELECT Nom
FROM Employe Chef
WHERE EXISTS (SELECT Nom
              FROM Employe
              WHERE Employe.Num_sup = Chef.Num) ;
```

Sous-requête avec un résultat à plusieurs colonnes

On peut utiliser la notation $(a, b, \ldots)$ pour former un n-uplet à comparer avec le résultat de la sous-requête :

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés ayant même fonction et même supérieur que 'Bellot' ?

```
SELECT Nom
FROM Employe
WHERE (Fonction, Num_sup) = (SELECT Fonction, Num_sup
                             FROM Employe
                             WHERE Nom='Bellot') ;
```

Sous-requêtes imbriquées

Il est possible d'imbriquer les sous-requêtes :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner le nom et la fonction des employés du département 20 ayant même fonction qu'une personne du département de 'Dupont'.

```
SELECT Nom, Fonction
FROM Employe
WHERE Num_dept = 20
AND fonction IN
        (SELECT Fonction
         FROM Employe
         WHERE Num_dept = (SELECT Num_dept
                           FROM Employe
                           WHERE Nom = 'Dupont')) ;
```

Opérations ensemblistes

- Permettent de combiner les résultats de plusieurs SELECT.
- Opérateur :
  - $\cup$: UNION
  - $\cap$: INTERSECTION
  - $-$: MINUS
- Pas de doubles (DISTINCT implicite).
- Les SELECT doivent contenir le même nombre d'attributs.
- Les noms des attributs sont ceux du premier SELECT.
  - C'est l'ordre des attributs qui compte.
- Seul le dernier SELECT peut contenir un ORDER BY.
  - Les colones à utiliser pour le tri sont précisées par leur numéro et pas par leur attribut.

Exemple

Schéma :
Employe1(Nom, Num, Fonction, NumSup, Embauche, Salaire, NumDept)
Employe2(Nom, Num, Fonction, Numsup, Embauche, Salaire, NumDept)

Liste des département ayant des employé dans 2 filiales dont les employés sont donnés par Employe1 et Employe2 :

```
(SELECT NumDept FROM Employe1)
INTERSECT
(SELECT NumDept FROM Employe2) ;
```

Expressions

Il est possible d'utiliser des expressions plus complexes que simples attributs.

Entre autres :

- Fonctions et expressions arithmétiques
- Fonctions sur les chaînes de caractères
- Fonctions sur les dates
- Fonctions de conversion

Il existe également des fonctions de groupes permettant de traiter plusieurs lignes à la fois.

Expressions - 2

Ces expressions sont utilisables :

- Dans le SELECT :
  - le nom dans la relation résultat est en général l'expression elle-même
    $\Rightarrow$ utiliser le renommage.
- Dans le WHERE :
  - permet d'exprimer des conditions plus complexes
- Dans le ORDER BY :
  - il est ainsi possible de trier les lignes selon des valeur plus complexes que de simples attributs

Quelques fonctions numériques

- $+$: unaire et binaire ;
- $-$: unaire et binaire ;
- $*$: multiplication et $/$: division ;
- $ABS(e)$: valeur absolue de e ;
- $COS(e)$: cosinus de e avec e en radians ;
- $SQRT(e)$: racine carrée de e ;
- $MOD(m, n)$: reste de la division entière de m par n, vaut 0 si $n = 0$;
- $ROUND(e, n)$: valeur arrondie de e à n chiffres après la virgule, n optionnel et vaut 0 par défaut ;
- $TRUNC(e, n)$: valeur tronquée de e à n chiffres après la virgule, n optionnel et vaut 0 par défaut.

Pour ROUND et TRUNC, si n est négatif cela indique des chiffres avant la virgule.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner pour chaque commercial son revenu (salaire + commission) :

```
SELECT Nom, (Salaire + Commission) Revenu
FROM Employe
WHERE Fonction = 'commercial' ;
```

Exemple - 2

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner la liste des commerciaux classée par rapport commission/salaire décroissant.

```
SELECT Nom, (Commission/Salaire) Rapport
FROM Employe
WHERE Fonction = 'commercial'
ORDER BY Commission/Salaire ;
```

Exemple - 3

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner, avec leur salaire journalier arrondi au centime près, la liste des employés dont la commission est inférieure à 50% du salaire.

```
SELECT Nom, ROUND(Salaire/(22*12), 2) SJournalier
FROM Employe
WHERE Commission <= Salaire * 0.5 ;
```

Fonctions sur les chaînes de caractères

- $CONCAT(e_1, e_2)$: concaténation de e_1 et e_2
  - Dans certains systèmes, $CONCAT$ peut prendre plus de deux arguments.
  - Dans certains systèmes, $CONCAT$ est représenté par l'opérateur binaire $||$.
- $REPLACE(e, old, new)$: Renvoie e dans laquelle les occurrences de old on été remplacées par new.
- $UPPER(e)$: convertit e en majuscules.
- $LENGTH(e)$: longueur de e.
- $INSTR(e, s)$: donne la position de la première occurrence s dans e.
- $SUBSTR(e, n, l)$ ou $SUBSTRING(e, n, l)$: renvoie la sous-chaîne de e commençant au caractère n et de longueur l
  - si l n'est pas précisé, on prend la sous-chaîne du caractère n jusqu'à la fin de e.

Fonctions sur les dates

Oracle :

- $d + n$ ou $d - n$: d est une date, le résultat est $d \pm n$ jours.
- $ADD\_MONTHS(d, n)$: ajoute n mois à d.
- $d_1 - d_2$: nombre de jours entre d_1 et d_2.
- $SYSDATE$: date courante.

MySQL :

- $ADDDATE(d, INTERVAL\ n\ DAY)$: ajoute n jours à d.
  - DAY peut être remplacé par $SECOND$, $MINUTE$, $HOUR$, $MONTH$, ou $YEAR$.
- $SUBDATE(d, INTERVAL\ n\ DAY)$: similaire à $ADDDATE$, mais effectue une soustraction.
- $DATEDIFF(d_1, d_2)$: nombre de jour entre d_1 et d_2.
- $SYSDATE()$: date courante.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner nombre de jours depuis l'embauche de chaque employé.

```
SELECT Nom, DATEDIFF(SYSDATE(),Embauche)
FROM Employe ;
```

Fonctions de conversion

- *ASCII*(*e*) : renvoie le code ASCII du premier caractère de *e*.

Oracle :

- *CHR*(*e*) : renvoie le caractère dont le code ASCII est *e*.
- *TO_NUMBER*(*e*) convertit la chaîne *e* en nombre.
- *TO_CHAR*(*e*, *format*) convertit *e* en chaîne de caractères.
  - *e* peut être un nombre ou une date ;
  - *format* indique la forme que doit avoir le résultat.
- *TO_DATE*(*e*, *format*) convertit une chaîne de caractères en date.
  - *format* est un chaîne de caractères contenant une indication sur la représentation de la date.
  - ex : TO_DATE('12122003','ddmmyyyy') donne la date '2003-12-12'

Fonction de conversion - 2

MySQL :

- *CAST*(*e AS type*) ou *CONVERT*(*e*, *type*) : convertit *e* en *type*.
  - *type* peut être BINARY, CHAR, DATE, TIME, DATETIME, SIGNED, UNSIGNED

Exécution naïve

SELECT att_1, att_2, ...
FROM $table_1$, $table_2$, ...
WHERE *condition*
ORDER BY att_i, att_j, ...

- Récupération des données dans le FROM
  $\longrightarrow$ on obtient un produit cartésien $table_1 \times table_2 \times \ldots$
- Filtrage des n-uplets obtenus en utilisant la condition du WHERE
- Tri des n-uplets restant suivant l'ordre spécifié par ORDER BY
- Calcul des n-uplets indiqué dans le SELECT à partir des restant n-uplets triés.

Exécution naïve - 2

SELECT att_1, att_2, ...
FROM $table_1$, $table_2$, ...
WHERE *condition*
ORDER BY att_i, att_j, ...

- Les requêtes imbriquées dans le FROM sont exécutées juste avant la création du produit cartésien.
- Les requêtes imbriquées dans le WHERE sont exécutées pour chaque n-uplet à tester.

En réalité, le SGBD optimise l'exécution des requêtes.

- Par exemple, les sous-requêtes dans le WHERE qui ne dépendent pas de la requête principale ne seront exécutées qu'une seule fois.

Exemple

Schéma :
Departement(Num_dept, Nom_dept, Num_bat)
Batiment(Num_bat, Nom_bat, Ent_princ, Ent_Sec)

```
SELECT Nom_dept, Batiment.Nom_bat
FROM Departement, Batiment
WHERE Departement.Num_bat = Batiment.Num_bat
ORDER BY Nom_dept ;
```

Exemple - 2

Departement			Batiment			
Num_dept	Nom_dept	Num_bat	Num_bat	Nom_bat	Ent_princ	Ent_Sec
10	Marketing	1	1	Turing	Nord	Ouest
20	Developpement	2	1	Turing	Nord	Ouest
30	Direction	3	1	Turing	Nord	Ouest
10	Marketing	1	2	Einstein	Ouest	NULL
20	Developpement	2	2	Einstein	Ouest	NULL
30	Direction	3	2	Einstein	Ouest	NULL
10	Marketing	1	3	Newton	Sud	Nord
20	Developpement	2	3	Newton	Sud	Nord
30	Direction	3	3	Newton	Sud	Nord
10	Marketing	1	4	Pointcarre	Est	NULL
20	Developpement	2	4	Pointcarre	Est	NULL
30	Direction	3	4	Pointcarre	Est	NULL

```
FROM Departement, Batiment
```

Exemple - 2

Departement			Batiment			
Num_dept	Nom_dept	Num_bat	Num_bat	Nom_bat	Ent_princ	Ent_Sec
10	Marketing	1	1	Turing	Nord	Ouest
20	Developpement	2	1	Turing	Nord	Ouest
30	Direction	3	1	Turing	Nord	Ouest
10	Marketing	1	2	Einstein	Ouest	NULL
20	Developpement	2	2	Einstein	Ouest	NULL
30	Direction	3	2	Einstein	Ouest	NULL
10	Marketing	1	3	Newton	Sud	Nord
20	Developpement	2	3	Newton	Sud	Nord
30	Direction	3	3	Newton	Sud	Nord
10	Marketing	1	4	Pointcarre	Est	NULL
20	Developpement	2	4	Pointcarre	Est	NULL
30	Direction	3	4	Pointcarre	Est	NULL

```
WHERE Departement.Num_bat = Batiment.Num_bat
```

Exemple - 3

Departement			Batiment			
Num_dept	Nom_dept	Num_bat	Num_bat	Nom_bat	Ent_princ	Ent_Sec
20	Developpement	2	2	Einstein	Ouest	NULL
30	Direction	3	3	Newton	Sud	Nord
10	Marketing	1	1	Turing	Nord	Ouest

```
ORDER BY Nom_dept
```

Nom_dept	Num_bat
Developpement	Einstein
Direction	Newton
Marketing	Turing

```
SELECT Nom_dept, Batiment.Nom_bat
```

Regroupements

SELECT att_1, att_2, ...
FROM $table_1$, $table_2$, ...
WHERE *condition*
GROUP BY att_k, att_l, ...
ORDER BY att_i, att_j, ...

Le GROUP BY, exécuté après le WHERE, indique de procéder à une répartition du résultat en groupes de n-uplets :

- Deux n-uplets sont dans un groupe s'il ont mêmes valeurs sur les attributs att_k, att_l, ...
- Si deux n-uplets sont dans deux groupes, alors il y a au moins un attribut parmi att_k, att_l, ... pour lequel ils ont une valeur différente.

Conséquences du regroupement

- La requête ne renvoie qu'un seul n-uplet par groupe.

- Le SELECT et le ORDER BY ne peuvent utiliser que des attributs présents dans le GROUP BY.
  - Dans un groupe, la valeur pour les attributs du GROUP BY est fixe, on peut donc l'utiliser.
  - En revanche, la valeur pour les autres attributs peut varier, ce qui rend leur utilisation directe impossible.
    (On ne saurait pas quelle valeur utiliser.)

Exemple

Schéma : Employe(Nom, Num, Fonction, Salaire, Num_Dept)

```
SELECT Fonction,Num_Dept
FROM Employe
GROUP BY Fonction, Num_Dept
ORDER BY Num_Dept ;
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Dupuis	14028	commercial	20000	10
LambertJr	15630	stagiaire	6000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Lambert	25012	directeur	30000	20
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10

Exemple - 2

```
SELECT Fonction
FROM Employe
GROUP BY Fonction, Num_Dept
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Dupuis	14028	commercial	20000	10
Fildou	25631	commercial	20000	10
LambertJr	15630	stagiaire	6000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Lambert	25012	directeur	30000	20
Soule	28963	directeur	25000	10

Exemple - 3

```
ORDER BY Num_Dept
```

Nom	Num	Fonction	Salaire	Num_dept
Dupuis	14028	commercial	20000	10
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10
Bellot	13021	ingenieur	25000	20
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
LambertJr	15630	stagiaire	6000	20
Lambert	25012	directeur	30000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30

Exemple - 4

```
SELECT Fonction, Num_Dept
```

Fonction	Num_dept
commercial	10
directeur	10
ingenieur	20
stagiaire	20
directeur	20
directeur	30
gestionnaire	30

Fonctions d'agrégation

- Fonctions agissant sur un ensemble de valeurs atomiques.
- Utilisables en conjonction avec un GROUP BY pour combiner les valeurs des attributs qui ne font pas partie du GROUP BY.
- Utilisées dans le SELECT et dans le ORDER BY.
- On ne peut *pas* les utiliser dans le WHERE.
  (Le where a lieu *avant* regroupement.)
- Par exemple, $AVG(e)$ donne la moyenne de l'expression e pour le groupe considéré.

Exemple

Schéma : Employe(Nom, Num, Fonction, Salaire, Num_Dept)

Donner le salaire moyen pour chaque fonction :

```
SELECT Fonction, AVG(Salaire) SalaireMoyen
FROM Employe
GROUP BY Fonction ;
```

Fonctions d'agrégation - 2

- *COUNT*(*e*) : Le nombre d'occurrences de *e* dans le groupe.
  - Les n-uplets pour lesquels *e* vaut NULL ne sont pas comptés.
  - * peut remplacer *e*. Compte alors le nombre de n-uplets du groupe.
- *MAX*(*e*) : La valeur maximale de *e* pour le groupe.
- *MIN*(*e*) : La valeur minimale de *e* pour le groupe.
- *SUM*(*e*) : La somme des valeurs de *e* pour le groupe.
- *AVG*(*e*) : La moyenne de l'évaluation de *e* sur le groupe.
- *STDDEV*(*e*) : L'écart-type de *e* pour le groupe.
- *VARIANCE*(*e*) : La variance de *e* pour le groupe.

e peut être précédé du mot clé DISTINCT : dans ce cas, on élimine les doublons.

- Important pour COUNT, SUM, AVG, STDDEV et VARIANCE.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Salaire, Num_Dept)
Departement(Num_dept, Nom_dept, Num_bat)

Donner pour chaque département le nombre de fonction différentes occupée dans ce département :

```
SELECT Nom_dept, COUNT(DISTINCT Fonction) NbFonctions
FROM Employe, Departement
WHERE Employe.Num_dept = Departement.Num_dept
GROUP BY Departement.Num_dept, Nom_dept ;
```

Exemple - 2

Schéma : Employe(Nom, Num, Fonction, Salaire, Num_Dept)

Donner pour chaque département le ou les employés qui ont le plus haut salaire :

```
SELECT Num_dept, Nom, Salaire
FROM Employe
WHERE (Num_dept, Salaire) IN
   (SELECT Num_dept, MAX(Salaire)
    FROM Employe
    GROUP BY Num_dept) ;
```

Sélection des groupes

SELECT att_1, att_2, ...
FROM $table_1$, $table_2$, ...
WHERE *condition*
GROUP BY att_k, att_l, ...
HAVING *condition_groupe*
ORDER BY att_i, att_j, ...

- Le WHERE ne pour que sur les n-uplets individuels, avant regroupement.
- La condition du HAVING pour sur les groupes et pas sur les n-uplets individuels :
  - Utilisation directe des attributs du GROUP BY possible.
  - Utilisation des autres attributs à travers les fonctions d'agrégation.
  - Exécuté entre le GROUP BY et le ORDER BY.

Exemple

```
SELECT Num_Dept, COUNT(DISTINCT Fonction) NbFonctions
FROM Employe
WHERE Salaire > 15000
GROUP BY Num_Dept
HAVING COUNT(*) > 2 ;
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Dupuis	14028	commercial	20000	10
LambertJr	15630	stagiaire	6000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Lambert	25012	directeur	30000	20
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10

Exemple - 2

```
FROM Employe WHERE Salaire > 15000
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Dupuis	14028	commercial	20000	10
LambertJr	15630	stagiaire	6000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Lambert	25012	directeur	30000	20
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10

Exemple - 3

```
GROUP BY Num_Dept
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Lambert	25012	directeur	30000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Dupuis	14028	commercial	20000	10
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10

Exemple - 4

```
HAVING COUNT(*) > 2
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Lambert	25012	directeur	30000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Dupuis	14028	commercial	20000	10
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10

Exemple - 5

```
SELECT Num_Dept, COUNT(DISTINCT Fonction) NbFonctions
```

Num_dept	NbFonctions
10	2
20	2

Tout regrouper

Utilisation d'une fonction d'agrégation sans GROUP BY :

- Provoque la création d'un groupe englobant tous les n-uplets sélectionnés.
- Le SELECT ne peut alors contenir que des fonctions d'agrégation.
- Utile pour obtenir des informations sur l'ensemble des lignes sélectionnées.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Salaire, Num_Dept)

Donner le total des salaires du département 10 :

```
SELECT SUM(Salaire)
FROM Employe
WHERE Num_dept = 10 ;
```

Double regroupement

Utilisation d'une fonction d'agrégation au résultat d'une fonction d'agrégation dans un SELECT :

- Possible uniquement dans une requête avec un GROUP BY.
- Cette utilisation provoque deux regroupements :
  - Premier regroupement classique par le GROUP BY
  - Deuxième regroupement implicite dû à la fonction d'agrégation dans le SELECT

Remarque : non implémenté dans MySQL, mais possibilité d'imiter ce comportement à l'aide d'une requête imbriquée.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Salaire, Num_Dept)

Donner la taille du plus gros département en termes de nombre d'employés.

```
SELECT MAX(COUNT(*))
FROM Employe
GROUP BY Num_dept ;
```

```
SELECT MAX(NbEmp)
FROM (  SELECT COUNT(*) NbEmp
        FROM Employe
        GROUP BY Num_dept)
                    CountEmp ;
```

Plan

Modification des données stockées dans une base

La modification s'effectue par ajout, suppression ou modification de n-uplets (lignes) dans l'instance de la base.

- SQL sert ici de langage de manipulation de données.
- Trois instructions SQL permettent ces modifications :
  - INSERT
  - DELETE
  - UPDATE
- Ces instructions de mise à jour peuvent utiliser des (morceaux de) requête afin d'effectuer des calculs pour sélectionner et/ou générer des données.

Insertion

Instruction INSERT

- INSERT INTO $nom\_table(att_1, \ldots, att_n)$ VALUES$(val_1, \ldots, val_n)$
- Ajoute le n-uplet $(val_1, \ldots, val_n)$ à la relation nom_table.
- val_i correspond à l'attribut att_i.
- Si un attribut de la relation nom_table n'apparaît pas dans $att_1, \ldots, att_n$, alors la valeur du n-uplet pour cet attribut est NULL.
- La spécification des attributs $att_1, \ldots, att_n$ est *optionnelle*
- Si on précise pas les attributs, il faut donner une valeur à tous les attributs.
  - L'ordre sur des valeurs $(val_1, \ldots, val_n)$ est celui des attributs dans la définition de la relation nom_table.
  - C'est un des cas où cet ordre est important.

Exemple

Schéma : Batiment(Num_bat, Nom_bat, Ent_princ, Ent_Sec)

Num_bat	Nom_bat	Ent_princ	Ent_sec
1	Turing	Nord	Ouest
2	Einstein	Ouest	NULL
3	Newton	Sud	Nord
4	Pointcarre	Est	NULL
5	Curie	Nord	NULL
6	Bohr	Sud	Est

```
INSERT INTO Batiment(Nom_bat,Num_bat,Ent_princ)
VALUES ('Curie',5,'Nord');
```

```
INSERT INTO Batiment VALUES (6,'Bohr','Sud','Est');
```

Insertion utilisant une requête

INSERT INTO $nom\_table(att_1, \ldots, att_n)$
SELECT $e_1, \ldots, e_n$
FROM ...

- Insertion dans *nom_table* des n-uplets calculés par la requête SELECT ... FROM ...
- La requête ne peut pas contenir de ORDER BY
  - De toute façon, c'est le SGBD qui détermine l'ordre dans lequel les n-uplets sont stockés.
- Le nom des colonnes dans le résultat de la requête n'est pas important : c'est l'ordre des expressions qui compte.

Exemple

Schéma :
Departement(Num_dept, Nom_dept, Num_bat)
Batiment(Num_bat, Nom_bat, Ent_princ, Ent_Sec)
Dept_important(Nom,Bat)

Ajouter à la table Dept_important les départements qui sont dans des batiments ayant une entrée secondaire :

```
INSERT INTO Dept_important(Bat,Nom)
SELECT Nom_bat, Nom_dept
FROM Batiment, Departement
WHERE Departement.Num_bat = Batiment.Num_bat
AND Ent_sec IS NOT NULL ;
```

Supression

DELETE FROM *nom_table*
WHERE *condition*

- Supprime les n-uplets de la relation *nom_table* qui vérifient *condition*.
- *condition* peut être aussi complexe qu'une condition exprimée dans le WHERE d'un SELECT.
  - En particulier, *condition* peut contenir des requêtes imbriquées.
  - Les requêtes imbriquées ne peuvent pas faire référence à *nom_table*, car elle est en cours de modification.
- WHERE *condition* est optionnel.
  - Si le WHERE est omis, tous les n-uplets sont supprimés (cela revient à utiliser a condition TRUE).

Exemple

Num_bat	Nom_bat	Ent_princ	Ent_sec
1	Turing	Nord	Ouest
2	Einstein	Ouest	NULL
3	Newton	Sud	Nord
4	Pointcarre	Est	NULL
5	Curie	Nord	NULL
6	Bohr	Sud	Est

Supprimer le bâtiment numéro 5 :

```
DELETE FROM Batiment
WHERE Num_bat = 5;
```

Exemple - 2

Schéma :
Batiment(Num_bat, Nom_bat, Ent_princ, Ent_Sec)
Departement(Num_dept, Nom_dept, Num_bat)

Supprimer les bâtiments qui ne correspondent à aucun département :

```
DELETE FROM Batiment
WHERE Num_bat NOT IN
    (SELECT Departement.Num_bat
     FROM Departement) ;
```

Modification de n-uplets

UPDATE *nom_table*
SET $att_1 = e_1$,
$att_2 = e_2$,
...
WHERE *condition*

- *condition* indique les lignes à modifier.
- att_i prend la valeur calculée par l'expression e_i.
- e_i peut utiliser $att_1, att_2, \ldots$, y compris att_i.
  - Ce sont les anciennes valeurs de $att_1, att_2, \ldots$ qui seront utilisées pour le calcul.
- Les e_i peuvent être des requêtes à condition qu'elles renvoient un unique résultat et que *nom_table* n'apparaisse pas dans un FROM.
- Similairement au DELETE, le WHERE est optionnel.
  - Si le WHERE est omis, tous les n-uplets sont modifiés.

Exemple

Schéma : Batiment(Num_bat, Nom_bat, Ent_princ, Ent_Sec)

Changer le nom du bâtiment numéro 3 en 'Copernic' :

UPDATE Batiment SET Nom_bat = 'Copernic' ;

Num_bat	Nom_bat	Ent_princ	Ent_sec
1	Turing	Nord	Ouest
2	Einstein	Ouest	NULL
3	Copernic	Sud	Nord

Exemple - 2

Schéma : Employe(Nom, Num, Fonction, Salaire, Num_Dept)

Augmenter de 10% le salaire des ingénieurs :

```
UPDATE Employe
SET Salaire = Salaire * 1.1
WHERE Fonction = 'ingenieur' ;
```

Exemple - 3

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)
Departement(Num_dept, Nom_dept, Num_bat)

Pour chaque département dont le chef n'est pas connu, spécifier que son chef est le plus ancien employé de ce département occupant la fonction de directeur.

Exemple - 3 - suite

```
UPDATE Departement
SET Num_chef =
    (SELECT Num
     FROM Employe
     WHERE Fonction = 'directeur'
     AND Employe.Num_dept = Departement.Num_dept
     AND Embauche <= ALL
           (SELECT Embauche
            FROM Employe E
            WHERE Fonction = 'directeur'
            AND E.Num_dept = Departement.Num_dept))
WHERE Num_chef IS NULL ;
```

Transactions

- Une transaction est un ensemble de modifications de la base qui forme un tout indivisible.
- Ces modifications doivent être effectuée entièrement ou pas du tout, sous peine de laisser la base dans un état incohérent.
- Au cours d'une transaction, seul l'utilisateur ayant démarré cette transaction voit les modifications effectuées.

Transactions en SQL

Gestion des transactions en SQL :

- COMMIT ;
  - Valide les modifications effectuées.
  - Les modifications sont alors définitives et visibles par tous.
- ROLLBACK ;
  - Annule les modifications effectuées depuis le début de la transaction.

Différences de traitement des transactions entre SGBDs

Dans Oracle :

- Une nouvelle transaction est implicitement démarrée au début de la connection et après chaque COMMIT.
- Le système assure la cohérence des données en cas de mise à jour simultanée par deux utilisateurs, en utilisant un système de verrouillage automatique.

Dans MySQL :

- Les tables doivent être stockées en utilisant le moteur de stockage InnoDB ou BDB pour que les transactions soient gérées.
- Par défaut, chaque mise à jour est immédiatement validée (COMMIT automatique).
- Pour démarrer explicitement une transaction, on utilise l'instruction BEGIN ;
  - Dans ce cas le COMMIT automatique est désactivé.

Plan

Gestion du schéma d'une base

SQL est également un langage de définition de données :

- Permet de créer ou supprimer des tables.
- Permet de modifier la structure d'une table.
- Permet de spécifier certaines contraintes d'intégrité sur le schéma.

DESC *nom_table* ;

Permet d'obtenir des informations sur le schéma d'une table.

- Les attributs et leur type.
- Des informations sur certaines contraintes d'intégrité.

Création de table

Lors de la création d'une table on indique :

- Le nom des attributs.
- Le type de chaque attribut.

De manière optionnelle :

- Certaines contraintes d'intégrité.
- Des caractéristiques de stockage.
- Des données provenant d'une requête.

Création simple

CREATE TABLE $nom\_table(att_1\ type_1, att_2\ type_2, \ldots)$;

- Crée une table nom_table ;
- ayant pour attributs $att_1, att_2, \ldots$;
- att_i ayant le type $type_i$.

Exemple :

```
CREATE TABLE Departement
        (Num_dept integer, Nom_dept varchar(30),
         Num_bat integer, Num_chef integer) ;
```

Création avec insertion de données

CREATE TABLE *nom_table*(att_1 $type_1$, att_2 $type_2$, ...)
AS SELECT ...;

- Crée la table comme précédemment
- Ajoute les données à la table comme si on avait exécuté : INSERT INTO *nom_table* SELECT ...;
- La spécification des attributs est optionnelle. Si les attributs sont omis :
  - Le nom des attributs est donné par le SELECT.
    - Implique un renommage obligatoire des expressions du SELECT.
  - Le type des attributs est déduit à partir du SELECT.
    - On peut utiliser les fonctions de conversion de type dans le SELECT.
- Pas de ORDER BY dans le SELECT.

Exemple

Créer une table dans laquelle on indique pour chaque département son nom et le numéro de son chef, ce dernier étant l'employé du département ayant le salaire le plus élevé.

```
CREATE TABLE Chef_dept
AS
SELECT Nom_dept, Num Chef
FROM Employe, Departement
WHERE Employe.Num_dept = Departement.Num_dept
AND Employe.Salaire >=
    (SELECT MAX(Salaire)
     FROM Employe E
     WHERE E.Num_dept = Departement.Num_dept) ;
```

Vues

Une vue est une requête à laquelle on donne un nom.

- Utilisable comme une table dans un SELECT.
- La vue est recalculée à chaque utilisation.
- Pas d'opération de mise à jour directement sur une vue.

Création :

CREATE VIEW *nom_vue*
AS
SELECT ...

Quelques types SQL Numériques

Type DECIMAL(*precision*,*echelle*)

- Représente un nombre codé sur *precision* chiffres, avec *echelle* chiffres après la virgule.
- *echelle* est optionnel et vaut 0 par défaut.
- *precision* est optionnel si *echelle* n'est pas indiqué.
  - Oracle → valeur par défaut : 38
  - MySQL → valeur par défaut : 10

Type FLOAT(*precision*)

- Représente un nombre à virgule flottante.
- *precision* est optionnel.
  - Oracle → *precision* en binaire, par défaut : 126 (soit 38 en décimal)
  - MySQL → *precision* en décimal, par défaut : 10

Les types INTEGER, INT, DOUBLE, ... sont des raccourcis pour des formes particulières de DECIMAL ou FLOAT

Quelques types SQL sur les caractères

Type CHAR(*longueur*)

- Chaîne de caractères de taille **fixe** *longueur*.

Type VARCHAR(*longueur*)

- Chaîne de caractère de taille **variable** *longueur*.

Objets de grande taille

Oracle

- Types BLOB et CLOB : jusqu'à 8 To de données binaires (BLOB) ou de caractères (CLOB).

MySQL

- Types BLOB et TEXT : jusqu'à 64 Ko de données binaires ou de caractères.
- Types MEDIUMBLOB et MEDIUMTEXT : jusqu'à 16 Mo.
- Types LONGBLOB et LONGTEXT : jusqu'à 4 Go.

Dates

Oracle

- Type DATE : date, y compris l'heure à la seconde près.
- Type TIMESTAMP : plus précis.

MySQL

- Type DATE : date au jour près.
- Type DATETIME : date + heure à la seconde près
- Type TIMESTAMP : nombre de secondes écoulées depuis le 1er janvier 1970, affichage similaire à DATETIME
- Type TIME : un nombre d'heures:minutes:secondes

Types énumérés

MySQL :

- Type ENUM('val_1', 'val_2', ...) :
  - Les valeurs autorisées pour l'attribut sont val_1, val_2, ...

Oracle :

- Type VARCHAR(n) CHECK (att IN ('val_1', 'val_2', ...)) :
  - att est l'attribut dont on définit le type.
  - Les valeurs autorisées pour l'attribut sont val_1, val_2, ...
  - n doit être supérieur à la plus grande longueur de valeur val_i.

Références

Oracle :
`http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14220/datatype.htm`

MySQL :
`http://dev.mysql.com/doc/refman/5.0/fr/column-types.html`

Contrainte NOT NULL

- Il est possible d'ajouter NULL ou NOT NULL après un type dans une définition de table pour indiquer si la valeur NULL est acceptée pour l'attribut.
- Par défaut, la valeur NULL est acceptée.

Exemple :

```
CREATE TABLE Bureau(Num_emp INTEGER,
                    Num_bat INTEGER NOT NULL,
                    Emplacement VARCHAR(20) NOT NULL);
```

Crée une table Bureau avec un attribut Nom_emp entier pouvant être NULL, Num_bat contenant un entier qui ne peut pas être NULL et enfin Emplacement contenant une chaîne de caractère et qui ne peut pas être NULL.

Contraintes d'intégrité en général

CREATE TABLE *nom_table* (
    att_1 $type_1$, att_2 $type_2$, ...,
    CONSTRAINT nom_1 $contrainte_1$,
    CONSTRAINT nom_2 $contrainte_2$, ...
);

- CONSTRAINT nom_i permet de nommer une contrainte.
- Le nom est optionnel.

Contraintes UNIQUE et PRIMARY KEY

..., CONSTRAINT nom_c UNIQUE (att_i, att_j, ...), ...

- Impose que chaque n-uplet aie une combinaison de valeurs différente pour les attributs att_i, att_j, ...
  - Il est par contre possible d'avoir deux fois la même valeur pour un attribut att_i
  - Si une des valeurs pour att_i, att_j, ... est NULL, la contrainte ne s'applique pas sur le n-uplet concerné.

..., CONSTRAINT nom_c PRIMARY KEY (att_i, att_j, ...), ...

- Indique que l'ensemble d'attribut (att_i, att_j, ...) sert d'identifiant principal (également appelé clé primaire) pour les n-uplets de la table.
- Implique NOT NULL sur chacun des (att_i, att_j, ...) et UNIQUE(att_i, att_j, ...)
- Il y a au maximum une contrainte PRIMARY KEY par table.

Clés étrangères

..., CONSTRAINT nom_c FOREIGN KEY (att_1, ..., att_k)
REFERENCES $table\_cible(att'_1,...,att'_k)$, ...

Une clé étrangère est une *référence* vers la clé primaire d'une table.

- Tout comme les clé primaires, elles peuvent être constituées de plusieurs attributs.
- Les valeurs pour (att_1, ..., att_k) doivent correspondre aux valeurs d'un des n-uplets de *table_cible* pour les attributs ($att'_1,...,att'_k$) ;

Rmq : dans MySQL, seul le moteur de stockage InnoDB gère correctement les clés étrangères.

Contrainte CHECK

. . ., CONSTRAINT nom_c CHECK (*condition*), . . .

- *condition* doit être vérifiée par chaque n-uplet stocké dans la table.
- La forme (*att* IN ('val_1', 'val_2', . . .)) utilisée pour les types énumérés est un cas particulier de cette contrainte.

!! Contrainte non vérifiée dans MySQL

MySQL : moteurs de tables

Dans MySQL, il existe plusieurs moteurs de stockage pour les tables, parmi lesquels :

- MyISAM : le moteur de stockage par défaut.
- InnoDB : permet de gérer les clés étrangères et les transactions.

CREATE TABLE *nom_table* (...) ENGINE = InnoDB ;

- Permet de créer une table utilisant le moteur InnoDB.

ALTER TABLE *nom_table* ENGINE = InnoDB ;

- Permet de changer le moteur de stockage d'une table en InnoDB.

Suppression et renommage de tables

DROP TABLE *nom_table* ;

- Supprime la table *nom_table*.
- Il ne faut pas qu'une clé étrangère d'une autre table référence la table à supprimer.
- En Oracle, on peut ajouter à la fin le mot clé CASCADE pour déclencher la suppression des clés étrangères qui référencent la table à supprimer.

RENAME *ancien_nom* TO *nouveau_nom* ;

- Renomme la table *ancien_nom* en *nouveau_nom*.

Ajout, modification ou suppression d'attribut

ALTER TABLE *nom_table* ADD *att type* NOT NULL ;

- Ajoute à la table *nom_table* un attribut *att* contenant des données correspondant à *type*.
- On peut optionnellement spécifier NOT NULL lorsque l'on souhaite interdire la valeur NULL.

ALTER TABLE *nom_table* MODIFY *att nouveau_type* NOT NULL ;

- Change le type de l'attribut *att*, en spécifiant optionnellement NOT NULL.

ALTER TABLE *nom_table* RENAME COLUMN *att* TO *nouvel_att* ;

- Change le nom de *att* en *nouvel_att*.

ALTER TABLE *nom_table* DROP COLUMN *att* ;

- Supprime l'attribut *att* de la table *nom_table*.

Ajouter ou supprimer une contrainte d'intégrité

ALTER TABLE *nom_table* ADD CONSTRAINT nom_c *contrainte* ;

- Ajoute la contrainte *contrainte* sur la table *nom_table*.
- CONSTRAINT nom_c spécifie le nom optionnel de la contrainte.

ALTER TABLE *nom_table* DROP PRIMARY KEY ;

- Supprime la clé primaire.

ALTER TABLE *nom_table* DROP FOREIGN KEY *nom_cle* ;

- Supprime la clé étrangère nommée *nom_cle*.

Plan

Exemple : Création du schéma Entreprise - 1

```
CREATE TABLE Employe (
        Nom VARCHAR(30) NOT NULL,
        Num INTEGER,
        Fonction VARCHAR(30) NOT NULL,
        Num_sup INTEGER,
        Embauche DATE NOT NULL,
        Salaire FLOAT NOT NULL,
        Num_dept INTEGER NOT NULL,
        Commission FLOAT,
        PRIMARY KEY (Num)
) ENGINE=InnoDB;
```

Exemple : Schéma Entreprise - 2

```
CREATE TABLE Batiment (
        Num_bat INTEGER NOT NULL,
        Nom_bat VARCHAR(30) NOT NULL,
        Ent_princ VARCHAR(10) NOT NULL,
        Ent_sec VARCHAR(10),
        PRIMARY KEY (Num_bat)
) ENGINE=InnoDB;
```

Exemple : Schéma Entreprise - 3

```
CREATE TABLE Departement (
        Num_dept INTEGER NOT NULL,
        Nom_dept VARCHAR(30) NOT NULL,
        Num_bat INTEGER,
        Num_chef INTEGER,
        PRIMARY KEY (Num_dept)
) ENGINE=InnoDB;
```

Exemple : Schéma Entreprise - 4

```
ALTER TABLE Employe
ADD CONSTRAINT fk_emp_dept
FOREIGN KEY (Num_dept)
REFERENCES Departement(Num_dept);
```

```
ALTER TABLE Employe
ADD CONSTRAINT fk_emp_sup
FOREIGN KEY (Num_sup)
REFERENCES Employe(Num);
```

Exemple : Schéma Entreprise - 5

```
ALTER TABLE Departement
ADD CONSTRAINT fk_dept_bat
FOREIGN KEY (Num_bat)
REFERENCES Batiment(Num_bat);
```

```
ALTER TABLE Departement
ADD CONSTRAINT fk_dept_chef
FOREIGN KEY (Num_chef)
REFERENCES Employe(Num);
```

Exemple : Remplissage des tables - 1

Des clés étrangères ont été définies :

- Les insertions ne peuvent pas se faire dans n'importe quel ordre.

Comme Batiment ne possède pas de clé étrangère on peut la remplir sans problème :

```
INSERT INTO Batiment VALUES (1,'Turing','Nord','Ouest');
INSERT INTO Batiment VALUES (2,'Einstein','Ouest',NULL);
...
```

Exemple : Remplissage des tables - 2

Si on ne spécifie pas les chefs (*i.e.* valeur NULL), on peut à présent remplir la table Departement :

```
INSERT INTO departement VALUES (10,'Marketing',1,NULL);
INSERT INTO departement VALUES (20,'Developpement',2,NULL);
INSERT INTO departement VALUES (30,'Direction',3,NULL);
```

Bien sûr, il faudra mettre à jour la table une fois les employés saisis.

Exemple : Remplissage des tables - 3

Il faut à présent saisir les employés en respectant l'ordre hiérarchique afin de ne pas violer la clé étrangère fk_emp_sup.

```
INSERT INTO Employe VALUES
('Martin',16712,'directeur',NULL,'1990-05-23',40000,30,NULL);
INSERT INTO Employe VALUES
('Julius',12569,'directeur',16712,'2001-02-25',32000,20,NULL);
INSERT INTO Employe VALUES
('Lambert',25012,'directeur',16712,'1998-09-20',30000,20,NULL);
INSERT INTO Employe VALUES
('Bellot',13021,'ingenieur',25012,'1996-05-18',25000,20,NULL);
INSERT INTO Employe VALUES
('Soule',28963,'directeur',16712,'1996-10-21',25000,10,10000);
...
```

Exemple : Remplissage des tables - 4

Enfin, on désigne le chef de chaque département comme étant le directeur gagnant le plus gros salaire :

```
UPDATE Departement
SET Num_chef =
    (SELECT Num
     FROM Employe
     WHERE Fonction = 'directeur'
         AND Employe.Num_dept = Departement.Num_dept
         AND Salaire >=
             (SELECT MAX(Salaire)
              FROM Employe e
              WHERE e.Num_dept = Departement.Num_dept
                  AND Fonction = 'directeur')
);
```