Langage SQL (3)

Sébastien Limet Denys Duchier

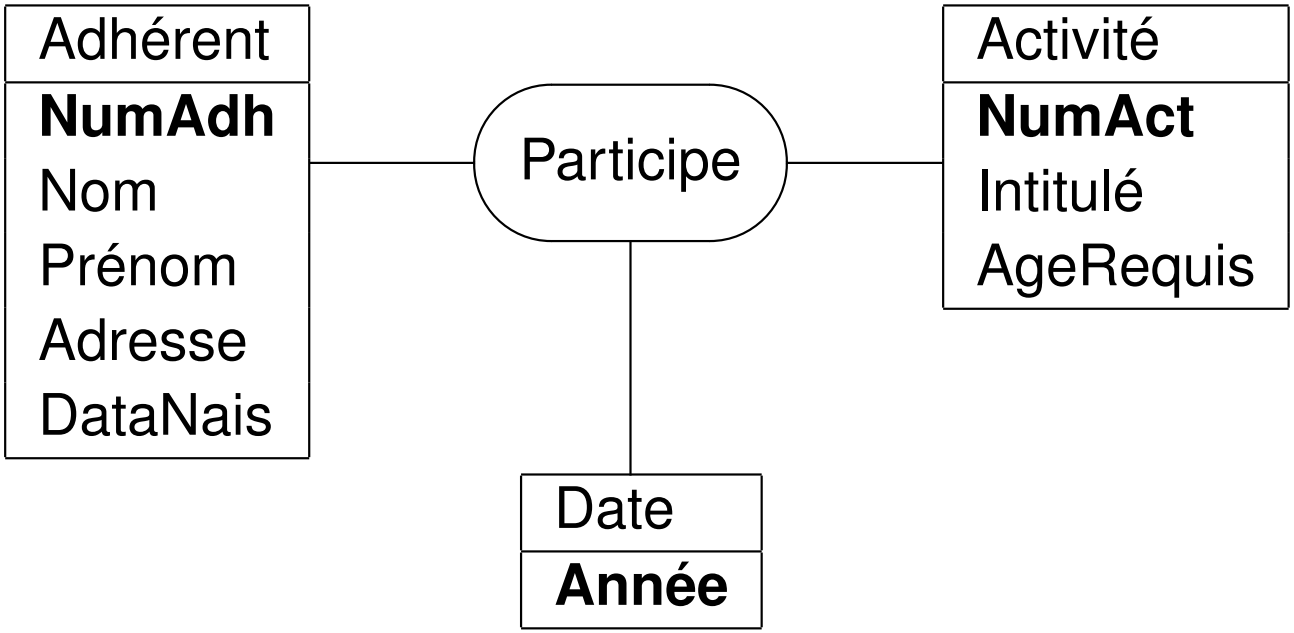
IUT Orléans

7 septembre 2007

Compléments sur le LDD

- notions de clés
- contraintes d'intégrité
- index

Exemple (MCD)


Adhérent
NumAdh
Nom
Prénom
Adresse
DataNais
Participe
Activité
NumAct
Intitulé
AgeRequis
Date
Année

Exemple (MLD)

Adhérent	Activité	Participe
numadh	numact	numadh
nom	intitulé	numact
prénom	agerequis	anneeparticipe
adresse		
datenais		

- `numadh` et `numact` sont les clés respectives de Adhérent et Activité
- ces deux champs sont reportés dans Participe : ce sont dont des clés *étrangères* dans cette table

Création des tables

```
create table Adherent
   (numadh     number(5),
    nom        varchar2(30),
    prenom     varchar2(30),
    adresse    varchar2(30),
    datenais   date,
    primary key (numadh));
```

```
create table Activite
   (numact     number(2),
    intitule   varchar2(20),
    agerequis  number(2),
    primary key (numact));
```

Création des tables

```
create table Participe(
   (numadh           number(5),
    numact           number(2),
    anneeparticipe   number(4),
    primary key (numadh,numact,anneeparticipe),
    foreign key (numadh)
            references Adherent(numadh),
    foreign key (numact)
            references Activite(numact));
```

Syntaxe

⟨création table⟩ ::= **create table** ⟨TABLE⟩ **(**
⟨colonnes⟩
⟨clé primaire⟩
⟨clé étrangère⟩**) ;**

⟨clé primaire⟩ ::=

| **, primary key (**⟨colonnes⟩**)**

⟨clé étrangère⟩ ::=

| **, foreign key (**⟨col⟩**)**
references ⟨table⟩ **(**⟨col⟩**)**
⟨clé étrangère⟩

Autres contraintes d'intégrité

- interdiction des valeurs NULL :

```
NumAdh Number(5) NOT NULL
```

- valeurs uniques :

```
NumAdh Number(5) UNIQUE
```

- restriction des valeurs du champ :

```
Sexe Char(1) check (Sexe in ('M','F')),
Annee Number(4) check Annee > 1990,
Nom Varchar2(30) check Nom = to_upper(Nom)
```

On ne peut se référer qu'à des colonnes de la même table ! ! !

Contraintes d'intégrité

- informations qui permettent au SGBD de maintenir la cohérence de la base de données
- ralentit les traitements du serveur
- mais très important dans un environement client/serveur

Création d'index

- un index permet d'accéder très rapidement à une donnée stockée sur le disque
- créer des index permet :
  - d'accélérer certaines requètes
  - d'améliorer les tris de données

Gestion d'index

- création :

  create [**unique**] **index** ⟨idx⟩ **on** ⟨table⟩ (⟨col⟩ [**desc**], . . .)

```
create index IndNom on Client(Nom);
```

- effacement :

  drop index ⟨idx⟩

Notions de SQL avancé

- différents types de jointures
- traitement des valeurs **`NULL`**

Eviter les requêtes imbriquées

Comment exprimer les requêtes du type :

- les adhérents qui ont eu au moins une activité en 2005
- les adhérents qui n'ont eu aucune activité
- quel est l'adhérent le plus agé ?
- quels sont les adhérents qui ont participé à toutes les activités artistiques ?

Les différentes jointures SQL

- mot clé pour les jointures : **`join`**
- instructions de la clause **`from`** :
  - **`cross join`** : produit cartésien
  - **`inner join`** : jointure interne
  - **`outer join`** : jointure externe

Produit cartésien

```
select *
  from adherent cross join participe
 where adherent.numadh = participe.numadh;
```

est équivalent à :

```
select *
  from adherent, participe
 where adherent.numadh = participe.numadh;
```

Jointure interne

C'est la jointure telle qu'on la connait ! Les 4 requêtes suivantes sont équivalentes :

```
select * from adherent a inner join participe p
    on a.numadh = p.numadh;
```

```
select * from adherent a inner join participe p
 using numadh;
```

```
select * from adherent a natural inner join
              participe p;
```

```
select * from adherent a, participe p
 where a.numadh = p.numadh;
```

Différentes options de la jointure

- **on** indique les conditions de jointures : ± ce qu'on met dans le **where** habituellement
- **using** ⟨cols⟩ : indique que le test de jointure se fait sur l'égalité des colonnes de la liste. équivalent à :

```
on t1.c1=t2.c1 and t1.c2=t2.c2 and ...
```

- **natural** : indique que la jointure se fait sur l'égalité des colonnes qui portent le même nom dans les deux tables

Jointure externe

- mot clé : **`outer join`**
- permet de garder toutes les lignes d'une des deux tables de jointure ; même celles qui ne se joignent à aucune dans l'autre table
- 3 options : **`left`**, **`right`**, **`full`** pour garder resp. les lignes de la table de gauche, de droite, ou des deux
- le mot clé **`outer`** peut être omis

Exemple `outer join`

```
select * from adherent natural outer join
              participe;
```

Adherent

Numadh	Nom
1	Dupont
2	Duval

Participe

Numadh	AnneeParticipe	NumAct
1	2000	1

Résultat

Numadh	Nom	Numadh	AnneeParticipe	NumAct
1	Dupont	1	2000	1
2	Duval	**NULL**	**NULL**	**NULL**

Exemples d'utilisation

- cadre des bases de données sans vues no requêtes imbriquées
- faciliter le travail d'optimisation des requêtes
- gérer les comptages avec des 0

Le prédicat IN

Les adhérents qui ont eu au moins une activité en 2000 :

- avec **IN** :

```
select * from adherent
 where numadh in
   (select numadh from participe
     where anneeparticipe = 2000)
```

- sans **IN** :

```
select distinct a.*
  from adherent a natural inner join
       participe p
 where anneeparticipe = 2000;
```

Le prédicat NOT IN

Les adhérents qui n'ont eu aucune activité :

```
select * from adherent
 where numadh not in
   (select numadh from participe);
```

Sans requêtes imbriquées ni vues cela devient plus compliqué : utilisation de la *jointure externe*.

Expression du NOT IN

Les adhérents qui n'ont participé à aucune activité sont ceux pour lesquels la colonne numact vaut **NULL** après la jointure externe :

```
select * from adherent natural left outer join
              participe
 where numact is NULL;
```

Remarque : on ne peut pas exprimer sans notion de vue la requête suivante *"les adhérents qui n'ont participé à aucune activité en 2000"* car il faudrait faire une sélection sur la table de droite avant la jointure

Exemple

Quel est l'adhérent le plus agé ?

```
select * from adherent a1 left join
              adherent a2
    on a1.datenais > a2.datenais
 where a2.datenais is NULL;
```

La personne la plus agée est celle telle qu'aucune autre personne de la base n'a une date de naissance inférieure à la sienne

Remarque : cette astuce ne marche que si on n'a pas besoin de sélection dans la table de droite

Attention !

- combinaisons de jointures assez périlleuses : il faut bien maîtriser !
- les sous-requêtes sont souvent très utiles. Ex : quels sont les adhérents qui ont participé à toutes les activités artistiques ?

Problème de comptage

Pour chaque adhérent, on veut le nombre dàctivités suivies en 2004 :

```
select numadh, nom, count(numact)
  from adherent natural inner join participe
 group by numadh, nom;
```

Problème : on compte ici le nombre d'activités des adhérents qui ont suivi *au moins une activité*. . . donc pas de ligne avec 0 !

Solution : jointure externe

Pour chaque adhérent, on veut le combre d'activités qu'il a suivies :

```
select numadh, nom, count(numact)
  from adherent natural left join participe
 group by numadh, nom;
```

count compte le nombre de valeurs connues, c'est à dire le nombre de valeurs différentes de **NULL**

Traitement des valeurs NULL

```
create table location (
    NumDVD     number(8),
    NumCli     number(8),
    DateDebut  date,
    DateFin    date);
```

- on veut le nombre de jours de location des DVD non rendus
- un DVD est non-rendu s'il a une DateFin égale à **NULL**

Solution : fonction **NVL**

```
def NVL(X, Y) :
    if (X is NULL) then
        return Y
    else
        return X
```

```
select NumDVD, nvl(DateFin,Sysdate) - DateDebut
  from location
 where DateFin is NULL;
```

Exemple

On veut la durée moyenne des locations pour chaque DVD en tenant compte des locations en cours

```
select NumDVD,
       avg(nvl(DateFin,Sysdate)-DateDebut)
  from location
 group by NumDVD;
```