Bases de Données : Modèle Physique (SQL)

Stéphane Devismes

Université Grenoble Alpes

26 août 2020

Plan

Traduction

Traduction du modèle logique (algèbre relationnelle) vers le modèle physique (d'abord SQL, ensuite PL/SQL).

Lorsque le concepteur a construit le modèle logique, il a déterminé les tables à construire et les contraintes qu'elles doivent satisfaire.

Les contraintes

Les contraintes sont réparties en trois classes :

1. **Les contraintes exprimables en SQL :** ce sont, essentiellement, les contraintes qui portent sur une seule table et les contraintes de clé étrangère.
2. **Les contraintes exprimables dans le langage procédural PL/SQL :** ce sont, essentiellement, les contraintes concernant plusieurs tables en-dehors des clés étrangères. L'expression de ces contraintes utilise des déclencheurs (≪ triggers ≫). Par exemple, pour une contrainte du type $R_1[A_1] \cap R_2[A_2] = \emptyset$.
3. **Enfin les contraintes d'atomicité transactionnelle** sont gérées qu'au niveau de l'application.

Exemple : Foot

$Pays(\underline{code}, nom\_pays, continent)$

$Joueurs(\underline{nom, prenom}, nbselection, dateNais, id\_pays)$

$Competitions(\underline{id}, edition, nature, organisateur, nbQualifie)$

$Qualifie(\underline{id\_pays, id\_compet})$

$Participations(\underline{id\_compet, nom, prenom})$

$Joueurs(id\_pays) \subseteq Pays(code)$

$Competitions(organisateur) \subseteq Pays(code)$

$Qualifie(id\_pays) \subseteq Pays(code)$

$Qualifie(id\_compet) \subseteq Competitions(id)$

$Participations(nom, prenom) \subseteq Joueurs(nom, prenom)$

$Participations(id\_compet) \subseteq Competitions(id)$

Destruction d'une table

(toujours détruire les tables dans l'ordre inverse de leur création)

- `DROP TABLE` nom-de-table ;

Création de table

- `CREATE TABLE` nom-de-table (définition-de-la-table) ;

on précise :

- Le nom de la table.
- Le nom et le type des attributs.
- Les contraintes (avec leurs noms).

Modification d'une table

Ajouter, supprimer ou renommer une colonne :

- `ALTER TABLE` nom-de-table `ADD` nom-de-colonne nom-de-type (contrainte)
- `ALTER TABLE` nom-de-table `DROP COLUMN` nom-de-colonne
- `ALTER TABLE` nom-de-table `RENAME COLUMN` nom-de-colonne `TO` nouveau-nom

Modifier la définition d'une table :
`ALTER TABLE` nom-de-table `MODIFY` (parties-à-modifier) ;

Rappel

Oracle effectue un `COMMIT` après chaque instruction de type `CREATE, DROP, ALTER`.

Donc les modifications utilisant ces instructions ne peuvent pas être annulées par un `ROLLBACK`.

Types atomiques

Les types des attributs doivent être des types atomiques : la place nécessaire pour stocker les éléments en mémoire est limitée.

Le type d'un attribut ne peut pas être

- un couple de . . . ,
- une liste de . . . ,
- un ensemble de . . . ,
- . . .

Types courants

En algèbre relationnelle, on a défini seulement trois domaines : `Nombre`, `Chaine` et `Date`.

En `SQL`, **la plupart des types sont paramétrés** pour préciser **leur format et leur taille**.
Parmi les types les plus courants :

- `CHAR(n)` : chaînes de caractères ASCII de longueur *n*.
- `VARCHAR2(n)` : chaînes de caractères UNICODE de longueur variable contenant **au maximum** *n*. (max : 4000)
- `NUMBER(p)` : nombre entier comportant **au plus** *p* chiffres significatifs
- `NUMBER(p,s)` : nombre décimal comportant **au plus** *p* chiffres significatifs **dont** *s* décimales.
- `DATE` : date.

De nombreux autres types, *e.g.*, `INTEGER`, `REAL`, `FLOAT`, . . .

Rappel : Type DATE

Les types diffèrent selon les implémentations de `SQL`.

Le standard `SQL2` préconise un type `DATE` pour année, mois et jour, et un type `TIME` pour heure, minute, seconde.

Cependant, en Oracle, tout cela est regroupé dans le type `DATE`.

Exemples

Les contraintes en SQL

Les contraintes SQL peuvent être définies lors de la création des tables, elles peuvent aussi être ajoutées, modifiées ou supprimées par la suite.

Chaque contrainte SQL est **associée à une table**.

Nommer les contraintes

On peut nommer les contraintes, et on a intérêt à le faire, en particulier parce que les messages d'erreur de SQL utilisent ces noms.

- CONSTRAINT nom-de-contrainte definition-de-contrainte

Valeur par défaut

On peut imposer une valeur par défaut à un attribut.

- DEFAULT valeur-par-defaut

Condition sur les attributs

On peut imposer qu'une condition soit vérifiée par toutes les lignes.

Cette condition **porte sur les colonnes de la table**, elle ne peut pas faire intervenir des colonnes d'une autre table.

La condition doit être « simple », au sens où elle ne peut pas utiliser de sous-requête.

- `CHECK` condition

Restriction de domaine

En particulier, une condition portant sur un seul attribut permet de restreindre le domaine de l'attribut, par exemple à **un intervalle** (« range ») ou à **un domaine énuméré**.

- `CHECK` nom-colonne `BETWEEN` *x* `AND` *y*
- `CHECK` nom-colonne `IN` (énumération)

Clé primaire

Chaque table a **au plus une clé primaire**, et en général exactement une.

Une clé est un ensemble d'attributs dont la valeur détermine la valeur de tous les attributs, et qui est minimal pour cette propriété : aucun sous-ensemble d'une clé n'est une clé.

Les valeurs absentes sont interdites dans les attributs faisant partie de la clé primaire.

- `PRIMARY KEY`

Clés secondaires

Une table peut aussi avoir une ou plusieurs clés secondaires.

Pour une table, une clé secondaire est une clé qui n'est pas la clé primaire. Les valeurs absentes sont autorisées.

- `UNIQUE`

Valeur obligatoire

Pour chaque attribut, on peut interdire les valeurs absentes. Si cet attribut fait partie de la clé primaire c'est automatique, sinon il faut le préciser.

- `NOT NULL`

Clé étrangère (1/2)

Une contrainte de clé étrangère relie deux tables différentes.

Elle assure l'intégrité référentielle des données : cela permet d'imposer que certaines valeurs d'une table « enfant » (référençante) R' apparaissent déjà dans une table « parent » (référencée) R.

On dit alors que la table R' fait référence à la table R.

Clé étrangère (2/2)

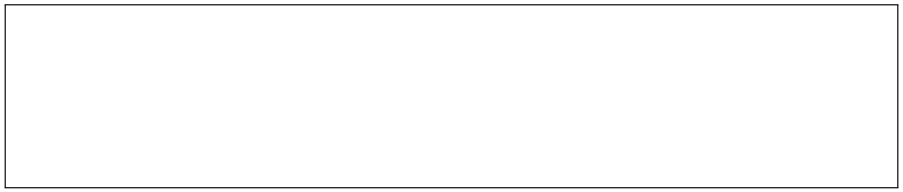
Soient $(A_1, ..., A_n)$ les attributs formant la clé primaire de R et soient $(B_1, ..., B_n)$ des attributs de S, tels que le domaine de A_i est le même que le domaine de B_i pour chaque i **(souvent A_i et B_i ont le même nom, mais ce n'est pas obligatoire)**.

Alors on peut mettre sur la table S la contrainte de clé étrangère disant que $(B_1, ..., B_n)$ font référence à $(A_1, ..., A_n)$ dans R.

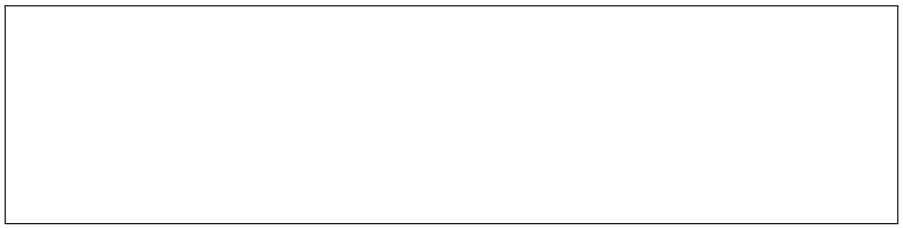
Cela signifie que pour toute ligne l_S de S il y a une ligne l_R de R telle que les valeurs de $(B_1, ..., B_n)$ dans l_S sont les valeurs de $(A_1, ..., A_n)$ dans l_R.

- `FOREIGN KEY` $(B_1, ..., B_n)$ `REFERENCES` $R(A_1, ..., A_n)$

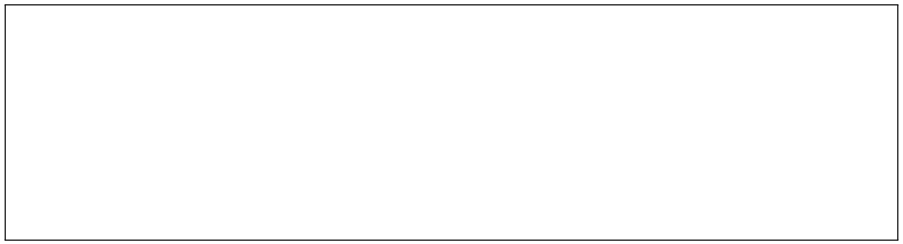
Exemples (1/5)



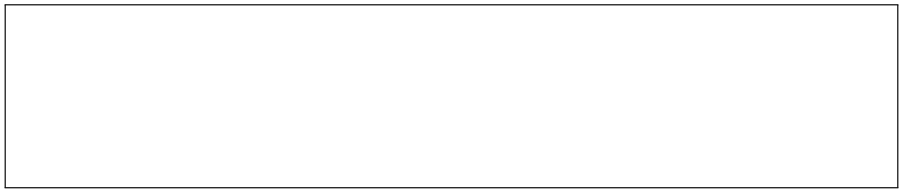
Exemples (2/5)



Exemples (3/5)



Exemples (4/5)



Exemples (5/5)

Modification des contraintes (1/2)

Les contraintes font partie de la table, donc tout ajout, modification ou suppression d'une contrainte est considéré comme une modification de la table.

Au moment où une contrainte est ajoutée ou modifiée, le système vérifie que la nouvelle contrainte est bien satisfaite, sinon il refuse l'ajout ou la modification.

Modification des contraintes (2/2)

- `ALTER TABLE` nom-de-table `ADD CONSTRAINT` nom-de-contrainte definition-de-contrainte
- `ALTER TABLE` nom-de-table `DROP CONSTRAINT` nom-de-contrainte
- `ALTER TABLE` nom-de-table `MODIFY` nom-de-attribut (type) definition-de-contrainte

Exemples

Voir les tables et les contraintes

- `SELECT` table_name `FROM` user_tables ;
- `SELECT` table_name, constraint_name, constraint_type `FROM` user_constraints ;
- `SELECT` constraint_name, constraint_type `FROM` user_constraints `WHERE` table_name = 'PAYS' ;

ATTENTION : Les noms de tables sont en majuscule et le type des contraintes est « codé » sur une lettre :

- `C` pour `CHECK` et pour `NOT NULL`,
- `P` pour `PRIMARY KEY`,
- `U` pour `UNIQUE`,
- `R` pour `REFERENCES`.

Modification des valeurs (1/2)

Une modification des valeurs d'une base n'est autorisée que si elle respecte les contraintes de la base.

Cela peut poser des problèmes délicats pour les contraintes d'intégrité référentielle (clés étrangères), en particulier en cas d'effacement de données.

Par défaut, il est interdit d'effacer une ligne d'une table parent si elle est référencée par une ligne d'une table enfant.

Cependant il existe plusieurs options, à utiliser avec précaution !

Modification des valeurs (2/2)

- Avec l'option « effacer en cascade », si on efface une ligne d'une table parent alors toutes les lignes des tables enfants qui y font référence sont elles aussi effacées.
  - `FOREIGN KEY... REFERENCES... ON DELETE CASCADE`
- Avec l'option « mettre des valeurs `NULL` », si on efface une ligne d'une table parent alors dans toutes les lignes des tables enfants qui y font référence on met la valeur `NULL`, mais on conserve ces lignes.

  Evidemment, c'est impossible si dans la table enfant les attributs concernés font partie de la clé primaire ou ont la contrainte « non-`NULL` ».
  - `FOREIGN KEY... REFERENCES... ON DELETE SET NULL`

Exemples

Désactivation et réactivation de contrainte

Il est possible, mais dangereux, de désactiver une contrainte. Il faut penser à la réactiver avant de clore la transaction. Alors le système vérifie que la contrainte est bien satisfaite, sinon il refuse la réactivation.

- `ALTER TABLE` nom-de-table `DISABLE CONSTRAINT` nom-de-contrainte
- `ALTER TABLE` nom-de-table `ENABLE CONSTRAINT` nom-de-contrainte

Différer des contraintes

Quand on peut prévoir qu'une contrainte devra être ≪ souvent ≫ désactivée, la meilleure solution est de prévoir de différer sa vérification dès la déclaration de la contrainte.

Ainsi on évite d'oublier de la réactiver !

C'est particulièrement utile pour les contraintes de clé étrangère ≪ fortes ≫.

Pour différer systématiquement la vérification d'une contrainte jusqu'à la fin de la transaction :

- `CONSTRAINT` nom-de-contrainte definition-de-contrainte `DEFERRABLE INITIALLY DEFERRED`

Questions d'ordre

Pour les contraintes de clé étrangère, encore mieux que désactiver ou différer, il suffit souvent de faire les choses dans le bon ordre pour éviter les problèmes !

- Créer la table parent avant la table enfant, et détruire la table enfant avant la table parent.
- Ajouter les données dans la table parent avant la table enfant, et supprimer les données dans la table enfant avant la table parent.

Exemples

Exemples

Séquence

Une séquence est un **compteur** qui est en général utilisé pour fournir des clés primaires.

Pour créer une séquence (par défaut, la séquence part de 1 et s'incrémente de 1 à chaque appel) :

```
CREATE SEQUENCE MaSequence
   [START WITH ... ]
   [INCREMENT BY ...];
```

Pour utiliser une séquence : la valeur courante de la séquence MaSequence est donnée par MaSequence.CURRVAL et la valeur suivante par MaSequence.NEXTVAL.

ATTENTION : dans une session, MaSequence.CURRVAL n'est défini qu'après le premier appel à MaSequence.NEXTVAL.

Exemples

```
create sequence masequence start with 10 increment by 3;

create table compteur(
id integer,
nom varchar2(50),
prenom varchar2(50),
constraint id_cle primary key (id),
constraint np_unique unique (nom,prenom));

insert into compteur(id,nom,prenom)
    values (masequence.nextval,'toto','Devismes');

insert into compteur(id,nom,prenom)
    values (masequence.nextval,'Stephane','Devismes');

alter table compteur disable constraint id_cle;

insert into compteur(id,nom,prenom)
    values (masequence.CURRVAL,'Raymond','Pascal');
```