

Les contraintes

Les vues

Année 2023-2024

Contraintes d'attribut

PRIMARY KEY

- désigne un *ensemble d'attributs* comme la *clé primaire* de la table

FOREIGN KEY

- désigne un *ensemble d'attributs* comme la *clé étrangère* dans une contrainte référentielle

NOT NULL

- spécifie qu'un attribut ne peut avoir de valeurs nulles

UNIQUE

- spécifie un *ensemble d'attributs* dont les valeurs doivent être distinctes pour chaque couple de n-uplets.

Clés étrangères

Clé étrangère :

Un ensemble d'attributs X d'une relation R (child table) est appelé une *clé étrangère (foreign key)* de la table R vers une table R' si X correspond à la clé primaire d'une relation R' (parent table).

Exemples :

Emp (Eno, Ename, Title, City) Pay(Title, Salary)
Works(Eno, Pno, Resp, Duree) Projet(Pno, Pname, Budget)

- Title est une clé étrangère de vers
- Eno est une clé étrangère de vers
- Pno est une clé étrangère de vers

Contraintes référentielles

- **Contrainte d'intégrité référentielle** : Pour chaque n-uplet t de la relation R avec la clé étrangère X , il existe un n-uplet t' dans la relation référencée R' où $t.X = t'.X$ (X est la clé primaire de R').
- Exemple :

Emp (Eno, Ename, Title, City) **Pay**(Title, Salary)

Pour chaque employé e dans la table Emp, il existe un n-uplet p dans la table Pay avec le salaire de l'employé.

Exemple

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Définition des références inter-tables

```
CREATE TABLE Works
(
    Eno      CHAR(3) ,
    Pno      CHAR(3) ,
    Resp     CHAR(15) ,
    Dur      INT,
PRIMARY KEY (Eno, Pno) ,
FOREIGN KEY (Eno) REFERENCES Emp (Eno) ,
FOREIGN KEY (Pno) REFERENCES Project (Pno) ) ;
```

Maintenance automatique de l'intégrité référentielle

La suppression (**ON DELETE**) ou la mise-à-jour (**ON UPDATE**) d'un n-uplet référencé (de clé primaire) nécessite une action sur le n-uplet avec la clé étrangère :

- **RESTRICT** : l'opération est *rejetée* (par défaut)
- **CASCADE** : supprime ou modifie tous les n-uplets avec la clé étrangère si le n-uplet référencé est supprimée ou *sa clé* est modifiée
- **SET [NULL | DEFAULT]** : mettre à NULL ou à la valeur par défaut quand le n-uplet référencé est effacée/sa clé est modifiée.

Exemple

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Définition des références inter-tables


```
CREATE TABLE Works
(
    Eno    CHAR(3),      Pno    CHAR(3),
    Resp   CHAR(15),     Dur    INT,
    PRIMARY KEY (Eno, Pno),
    FOREIGN KEY (Eno) REFERENCES Emp(Eno)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (Pno) REFERENCES Project(Pno)
        ON DELETE RESTRICT;
```

Exemple: cycle

Cycle de références :

```
CREATE TABLE Emp
(   Eno CHAR(3),
    Ename VARCHAR(20),
    Director CHAR(3)
    CONSTRAINT key_emp PRIMARY KEY Eno);
```

*Nom de contrainte
(optionnel)*

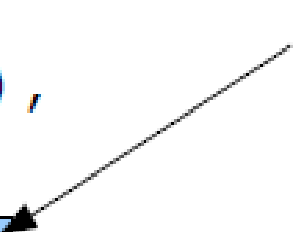


```
ALTER TABLE Emp
(  ADD CONSTRAINT foreign_key_emp
    FOREIGN KEY Director REFERENCES Emp );
```


Exemple: effacer contrainte

```
CREATE TABLE Emp
(   Eno CHAR(3),
    Ename VARCHAR(20),
    Director CHAR(3)
    CONSTRAINT key_emp PRIMARY KEY Eno) ;
ALTER TABLE Emp
(   DISABLE CONSTRAINT key_emp ) ;
ALTER TABLE Emp
(   DROP CONSTRAINT key_emp ) ;
```

optionnel



Contraintes d'attribut et de domaine

Contraintes portant *sur un seul attribut d'une table*

(d'autres attributs/tables peuvent apparaître dans des sous-requêtes)

- Spécifie une condition que chaque n-uplet doit satisfaire:

```
CREATE TABLE Project
(Pno CHAR(3),
 Pname VARCHAR(20),
 Budget DECIMAL(10,2) DEFAULT 0.00
 CONSTRAINT positive_budget CHECK (BUDGET >= 0),
 City CHAR(9))
PRIMARY KEY (Pno);
```

- Peut être utilisé pour définir des **contraintes de domaine** :

```
CREATE DOMAIN Gender AS CHAR(1)
CHECK (VALUE IN ('F', 'M'));
```

Contraintes de n-uplets

Contraintes portant *sur une seule table*.

(d'autres tables peuvent apparaître dans des sous-requêtes)

La condition est vérifiée *chaque fois* qu'un **n-uplet** est inséré ou modifié dans la table; la mise-à jour (transaction) est rejetée si la condition est fausse.

```
CREATE TABLE Works
  (Eno      CHAR(3) ,
   Pno      CHAR(3) ,
   Resp     CHAR(15) ,
   Dur      INT,
  PRIMARY KEY (Eno, Pno) ,
  FOREIGN KEY (Eno) REFERENCES Emp(Eno) ,
  FOREIGN KEY (Pno) REFERENCES Project(Pno) ,
  CHECK (NOT (PNO < 'P5') OR Dur > 18) );
```

Contraintes de n-uplets complexes

Un projet ne peut avoir plus de 2 employés avec une affectation de plus de 48 mois :

```
CREATE TABLE Works
(Eno CHAR(3),
 Pno CHAR(3),
 Resp CHAR(15),
 Dur INT,
PRIMARY KEY (Eno, Pno),
FOREIGN KEY (Eno) REFERENCES Emp(Eno)
ON DELETE SET NULL
ON UPDATE CASCADE;
FOREIGN KEY (Pno) REFERENCES Project(Pno),
CHECK(3 > ALL
(SELECT COUNT(Eno) FROM Works
WHERE Dur > 48 GROUP BY Pno));
```

Vues

Une BD peut contenir des *centaines de tables avec des milliers d'attributs* :

1. Les requêtes sont complexes :
 - difficiles à formuler
 - source d'erreurs
2. Une modification du schéma nécessite la modification de beaucoup de programmes.

Solution : Adapter le schéma et les données à des applications spécifiques → **vues**

Définition d'une vue

Définition : Une **vue** $V(a_1, a_2, \dots, a_n)$ est une **relation** avec n attributs qui contient le résultat d'une requête $Q(a_1, a_2, \dots, a_n)$ évaluée sur une base de données BD :

$$V(a_1, a_2, \dots, a_n) :- Q(x_1, x_2, \dots, x_n, BD)$$

Remarques :

- V possède un schéma relationnel avec des attributs a_1, \dots, a_n .
- V reflète l'état actuel d'une base de données BD
- V peut être interrogée et il est possible de définir des vues à partir d'autres vues.
- On distingue les relations « **matérialisées** » (tables) et les relations « **virtuelles** » (vues)

Définition d'une vue dans SQL

```
CREATE VIEW nom_vue [(att1, att2...)]  
AS requête_SQL [ WITH CHECK OPTION ]
```

- *nom_vue* désigne le nom de la relation
- *att1*, ... (optionnel) permet de nommer les attributs de la vue (attributs de la requête par défaut)
- *requête_SQL* désigne une requête SQL standard qui définit le « contenu » (instance) de la vue
- **WITH CHECK OPTION** (voir mises-à-jour de vues)

Exemple

Emp (Eno, Ename, Title, City)
Pay(Title, Salary)

Project(Pno, Pname, Budget, City)
Works(Eno, Pno, Resp, Dur)

Définition de la vue **EmpProjetsParis** des employés travaillant dans des projets à Paris :

```
CREATE VIEW EmpProjetsParis(NumE, NomE, NumP, NomP, Dur)
AS SELECT Emp.Eno, Ename, Works.Pno, Pname, Dur
   FROM Emp, Works, Project
  WHERE Emp.Eno=Works.Eno
     AND Works.Pno = Project.Pno
     AND Project.City = 'Paris'
```


Interrogation de vues

Emp (Eno, Ename, Title, City)
Pay (Title, Salary)

Project (Pno, Pname, Budget, City)
Works (Eno, Pno, Resp, Dur)

Les noms des employés de projets Parisiens :

Requête sans vue:

```
SELECT Ename
FROM Emp, Works, Project
WHERE Emp.Eno=Works.Eno
AND Works.Pno = Project.Pno
AND Project.City = 'Paris'
```

Requête avec vue:

```
SELECT NomE
FROM EmpProjetsParis
```

On obtient le même
résultat

Vues modifiables

Une vue *n'est pas modifiable* :

- quand elle ne contient pas tous les attributs définis comme NON NULL dans la table interrogée
- quand elle contient une jointure
- quand elle contient une fonction agrégat

Règle : Une vue *est modifiable* quand elle est définie comme une *sélection/projection sur une relation R* (qui peut aussi être une vue modifiable) sans utilisation de SELECT DISTINCT.

Mises-à-jour

Emp (Eno, Ename, Title, City)
Pay(Title, Salary)

Project(Pno, Pname, Budget, City)
Works(Eno, Pno, Resp, Dur)

```
CREATE VIEW ProjetParis
AS SELECT Pno, Pname, Budget
      FROM Project
      WHERE City='Paris';
```

```
UPDATE ProjetParis
SET Budget = Budget*1.2;
```

WITH CHECK OPTION

WITH CHECK OPTION protège contre les « disparitions de n-uplets » causées par des mise-à-jour :

```
CREATE VIEW ProjetParis  
  WITH CHECK OPTION  
AS SELECT Pno, Pname, Budget, City  
   FROM Project  
  WHERE City = 'Paris';
```

```
UPDATE ProjetParis  
SET City = 'Lyon'  
WHERE Pno=142;
```

← Mise-à-jour rejetée

Vues et tables

Similitudes :

- Interrogation SQL
- UPDATE, INSERT et DELETE sur vues modifiables
- Autorisations d'accès

Différences:

- On ne peut pas créer des index sur les vues
- On ne peut pas définir des contraintes (clés)
- Une vue est recalculée à chaque fois qu'on l'interroge
 - *Vue matérialisée* : stocker temporairement la *vue* pour améliorer les performances.