LifBDW2 –Bases de données avancées
TP2 – Contraintes et dictionnaire

Licence informatique – Automne 2016–2017

Résumé

Ce TP poursuit le précédent avec les contraintes de clef, de valeurs non-nulles, sur les tuples et d'intégrité référentielle, qui sont les contreparties pratiques des dépendances fonctionnelles et d'inclusion vues en cours et en TD.

1 Contraintes de clé

Exercice 1 : re-création de la base avec contraintes

Supprimer les tables créées dans le TP1. Modifier ensuite les déclarations de création de tables afin de prendre en compte les contraintes suivantes.

1. Ajouter les contraintes de clés (`PRIMARY KEY, UNIQUE`) suivantes dans la déclaration des tables
   — `mID` est une clé de `Movie`.
   — Le couple `(title, year)` est également une clé de `Movie`.
   — `rID` est une clé de `Reviewer`.
   — `(rID, mID, ratingDate)` est une clé de `Rating`, mais avec des valeurs nulles autorisées.
2. Exprimer les contraintes des questions précédentes dans le formalisme des dépendances fonctionnelles.
3. Ajouter les contraintes de valeurs non-nulles (`NOT NULL`) suivantes dans la déclaration des tables
   — `Reviewer.name` doit être non nul.
   — `Rating.stars` doit être non nul.
4. Ajouter les contraintes suivantes sur les tuples (`CHECK`) dans la déclaration des tables
   — Les films doivent être réalisés après 1900.
   — `Rating.stars` est défini sur l'ensemble $\{1, 2, 3, 4, 5\}$.
   — La date `Rating.ratingDate` doit être postérieure au 01/01/2000.
   — Les films réalisés par *Steven Spielberg* sont antérieurs à 1990 et ceux de *James Cameron* sont postérieurs à 1990. Pour l'expression de la contrainte booléenne, penser à l'équivalence logique $A \Rightarrow B \equiv \neg A \vee B$.
5. Reprendre quelques-unes des questions précédentes en modifiant la structure des tables sans les supprimer (commande `ALTER TABLE`).

Exercice 2 : vérification des contraintes

1. Importer les données du TP1 dans les tables nouvellement crées. Il ne doit pas y avoir d'erreur car toutes les contraintes sont satisfaites.
2. Vérifier que chacune des commandes suivante génère une erreur. Préciser la contrainte violée en relevant le message d'erreur obtenu.
   1. `insert into Movie values (109, 'Titanic', 1997, 'JC');`
   2. `insert into Reviewer values (201, 'Ted Codd');`
   3. `update Rating set rID = 205, mID=104;`
   4. `insert into Reviewer values (209, null);`
   5. `update Rating set stars = null where rID = 208;`

6. `update Movie set year = year  40;`
7. `update Rating set stars = stars + 1;`
8. `insert into Rating VALUES (201, 101, 1, to_date('01-01-1999','dd-mm-yyyy'));`
9. `insert into Movie values (109, 'Jurassic Park', 1993, 'Steven Spielberg');`
10. `update Movie set year = year-10 where title = 'Titanic';`

3. Vérifier que chacune des commandes suivantes ne génère pas d'erreur.
   1. `update Movie set mID = mID - 1;`
   2. `insert into Movie values (109, 'Titanic', 2001, null);`
   3. `update Rating set mID = 109;`
   4. `update Movie set year = 1901 where director <> 'James Cameron';`
   5. `update Rating set stars = stars  1;`
4. Videz les trois tables puis réimportez les données du TP1 avant de passer à la suite.

2 Contraintes d'intégrité référentielle

Exercice 3 : contraintes d'intégrité référentielle

Oracle permet plusieurs types d'application des contraintes d'intégrité référentielle qu'il faut préciser au moment de la déclaration de la contrainte via `FOREIGN KEY`. Dans les exemples suivants `DEPT` est la *table parent* et `EMP` est la *table enfant*, c'est-à-dire respectivement la table *référencée* et celle sur laquelle *porte* la contrainte.

— Pour empêcher toute mise à jour ou suppression d'une clé de la table parent :

```
CREATE TABLE EMP (
  empno number(4),
  deptno number(2),
  FOREIGN KEY (deptno) REFERENCES DEPT (deptno));
```

— Quand une clé de la table parent est supprimée, tous les tuples de la table enfant qui dépendent de la valeur supprimée sont également supprimés :

```
CREATE TABLE EMP (
  empno number(4),
  deptno number(2),
  FOREIGN KEY (deptno) REFERENCES DEPT (deptno)
  ON DELETE CASCADE);
```

— Ne pas supprimer les tuples de la table enfant quand une clé de la table parent est supprimé. Les valeurs de la table enfant sont alors mises à `NULL` :

```
CREATE TABLE EMP (
  empno number(4),
  deptno number(2),
  FOREIGN KEY (deptno) REFERENCES DEPT (deptno)
  ON DELETE SET NULL);
```

1. Modifiez la définition de la table `Rating` pour prendre en compte les contraintes suivantes :
   1. Intégrité référentielle entre `Rating.rID` et `Reviewer.rID` (suppression sur `Reviewers` : `SET NULL`).
   2. Intégrité référentielle entre `Rating.mID` et `Movie.mID` (suppression sur `Movies` : `CASCADE`).
2. Exprimer les contraintes des questions précédentes dans le formalisme des dépendances d'inclusion.

Exercice 4 : vérification des contraintes (suites)

1. Est-ce que les instructions suivantes génèrent des erreurs ou non ? Si oui, relever les messages.
   1. `INSERT INTO Rating VALUES(333, 104, 3, to_date('02-01-2011', 'dd-mm-yyyy'));`
   2. `INSERT INTO Rating VALUES(208, 111, 3, to_date('02-01-2011', 'dd-mm-yyyy'));`
   3. `update Rating set rID = 209 where rID = 208;`
   4. `update Rating set mID = mID + 1;`
   5. `update Movie set mID = 109 where mID = 108;`
   6. `update Reviewer set rID = rID + 10;`
2. Exécutez les requêtes suivantes en vérifiant que la table `Rating` a été modifiée en conséquence.
   1. `DELETE FROM Reviewer WHERE rID = 208;`
   2. `DELETE FROM MOVIE WHERE mID = 101;`
3. Expliquer ce que fait la requête suivante (recherche de la documentation)

```
SELECT table_name,
       constraint_name,
       column_name
FROM USER_CONSTRAINTS NATURAL JOIN USER_CONS_COLUMNS
WHERE table_name in ('MOVIE', 'RATING', 'REVIEWER');
```

Exercice 5 : vérification des dépendances en SQL (cf. TD2)

1. Écrire une requête SQL qui vérifie si une dépendance $X \rightarrow Y$ est vérifiée par une instance r en utilisant la sémantique des dépendances fonctionnelle $\forall t_0, t_1, t_0[X] = t_1[X] \rightarrow t_0[Y] = t_1[Y]$.
2. Proposer une méthode qui permet de tester la satisfaction d'une dépendance fonctionnelle avec SQL en s'appuyant sur le faite que $r \models X \rightarrow Y$ *si et seulement si* $|\pi_X(r)| = |\pi_{XY}(r)|$.
3. Essayer de comparer l'efficacité des approches sur le jeu de données fourni [1] en vérifiant si les dépendances $AA, AB, AC, AD, AE, AF, AG, AH \rightarrow AI$ et $AA, AB, AC, AD, AE, AF, AG, AH, AI \rightarrow AJ$ sont satisfaites.
4. (Subsidiaire) Comment a été généré ce jeu de données ?

1. http://liris.cnrs.fr/marc.plantevit/ENS/LIFBDW2/TP/TP2_dataset.sql

Corrections

Solution de l'exercice 1

On donne les déclarations finales des tables avec toutes les contraintes intégrées. Pour avoir `(rID, mID, ratingDate)` avec des valeurs nulles autorisées, il faut mettre une contrainte `UNIQUE` et pas une `PRIMARY KEY`.

```
DROP TABLE Movie;
create table Movie(
   mID integer NOT NULL PRIMARY KEY,
   title varchar(50) NOT NULL,
   year integer NOT NULL CHECK (year > 1900),
   director varchar(30),
   CONSTRAINT key_title_year UNIQUE (title,year),
   CONSTRAINT chk_spielberg CHECK
    (director NOT LIKE 'Steven␣Spielberg'
      OR (director LIKE 'Steven␣Spielberg' AND year < 1990)),
   CONSTRAINT chk_cameron CHECK
    (director NOT LIKE 'James␣Cameron'
      OR (director LIKE 'James␣Cameron' AND year > 1990))
);

DROP TABLE Reviewer;
create table Reviewer(
   rID integer NOT NULL PRIMARY KEY,
   name varchar(30) NOT NULL
);

DROP TABLE Rating;
create table Rating(
   rID integer,
   mID integer,
   stars integer NOT NULL CHECK (stars IN (1,2,3,4,5)),
   ratingDate date CHECK (ratingDate >= to_date('01-01-2000', 'dd-mm-yyyy')),
   CONSTRAINT key_rid_mid_date UNIQUE (rID,mID,ratingDate)
);
```

Les dépendances associées aux contraintes sont les suivantes :

- *Movie* : *mID* $\rightarrow$ *title*, *year*, *director*
- *Movie* : *title*, *year* $\rightarrow$ *mID*, *director*
- *Reviewer* : *rID* $\rightarrow$ *name*
- *Rating* : *rID*, *mID*, *ratingDate* $\rightarrow$ *stars*

Solution de l'exercice 2

1. C'est bien le cas.
2. On remarque que si on ne précise pas de nom explicite à une contrainte, alors Oracle en choisit qui n'est pas très évocateur.
   1. `ORA-00001: unique constraint (RTHION.SYS_C0025620) violated`
   2. `ORA-00001: unique constraint (RTHION.SYS_C0025624) violated`
   3. `ORA-00001: unique constraint (RTHION.KEY_RID_MID_DATE) violated`
   4. `ORA-01400: cannot insert NULL into ("RTHION"."REVIEWER"."NAME")`
   5. `ORA-01407: cannot update ("RTHION"."RATING"."STARS") to NULL`
   6. `ORA-02290: check constraint (RTHION.SYS_C0025617) violated`
   7. `ORA-02290: check constraint (RTHION.SYS_C0025626) violated`
   8. `ORA-02290: check constraint (RTHION.SYS_C0025627) violated`

9. `ORA-02290: check constraint (RTHION.CHK_SPIELBERG) violated`
10. `ORA-02290: check constraint (RTHION.CHK_CAMERON) violated`

3. C'est bien le cas.

Solution de l'exercice 3

1.
```
ALTER TABLE Rating
ADD CONSTRAINT fk_Rating_Reviewer
FOREIGN KEY (rID)
REFERENCES Reviewer(rID)
ON DELETE SET NULL;

ALTER TABLE Rating
DROP CONSTRAINT fk_Rating_Reviewer;

ALTER TABLE Rating
ADD CONSTRAINT fk_Rating_Reviewer
FOREIGN KEY (rID)
REFERENCES Reviewer(rID)
ON DELETE SET NULL;

ALTER TABLE Rating
DROP CONSTRAINT fk_Rating_Reviewer;

ALTER TABLE Rating
ADD CONSTRAINT fk_Rating_Movie
FOREIGN KEY (mID)
REFERENCES Movie(mID)
ON DELETE CASCADE;

ALTER TABLE Rating
DROP CONSTRAINT fk_Rating_Movie;
```

2. $Rating[rID] \subseteq Reviewer[rID]$ et $Rating[mID] \subseteq Movie[mID]$

Solution de l'exercice 4

1.
- `ORA-02291: integrity constraint (RTHION.FK_RATING_REVIEWER) violated - parent key not found`
- `ORA-02291: integrity constraint (RTHION.FK_RATING_MOVIE) violated - parent key not found`
- `ORA-02291: integrity constraint (RTHION.FK_RATING_REVIEWER) violated - parent key not found`
- `ORA-02291: integrity constraint (RTHION.FK_RATING_REVIEWER) violated - parent key not found`
- `ORA-02292: integrity constraint (RTHION.FK_RATING_MOVIE) violated - child record found`
- `ORA-02292: integrity constraint (RTHION.FK_RATING_MOVIE) violated - child record found`

2.
```
SELECT count(*) FROM RATING WHERE rID = 208;
SELECT count(*) FROM RATING WHERE rID IS NULL;
DELETE FROM Reviewer WHERE rID = 208;
SELECT count(*) FROM RATING WHERE rID = 208;
SELECT count(*) FROM RATING WHERE rID IS NULL;

SELECT count(*)  FROM Rating WHERE mID = 101;
SELECT count(*) FROM RATING WHERE mID IS NULL;
```

```
DELETE FROM MOVIE WHERE mID = 101;
SELECT count(*)  FROM Rating WHERE mID = 101;
SELECT count(*) FROM RATING WHERE mID IS NULL;
```

3. On interroge deux tables système pour connaître les contraintes sur les tables de la base. On obtient quelque chose semblable à ce qui suit.

```
MOVIE     SYS_C0025614       MID
MOVIE     SYS_C0025615       TITLE
MOVIE     SYS_C0025616       YEAR
MOVIE     SYS_C0025617       YEAR
MOVIE     CHK_SPIELBERG       YEAR
MOVIE     CHK_SPIELBERG       DIRECTOR
MOVIE     CHK_CAMERON         YEAR
MOVIE     CHK_CAMERON         DIRECTOR
MOVIE     SYS_C0025620       MID
MOVIE     KEY_TITLE_YEAR     TITLE
MOVIE     KEY_TITLE_YEAR     YEAR
RATING   SYS_C0025625       STARS
RATING   SYS_C0025626       STARS
RATING   SYS_C0025627       RATINGDATE
RATING   KEY_RID_MID_DATE   RID
RATING   KEY_RID_MID_DATE   MID
RATING   KEY_RID_MID_DATE   RATINGDATE
RATING   FK_RATING_REVIEWER RID
RATING   FK_RATING_MOVIE     MID
REVIEWER SYS_C0025622       RID
REVIEWER SYS_C0025623       NAME
REVIEWER SYS_C0025624       RID
```

Solution de l'exercice 5

1. Avec $X = A_1 \dots A_m$ et $Y = B_1 \dots B_n$, on obtient la requête :

```
SELECT count(*)
FROM R R1 INNER JOIN R R2 on
   R1.A1 = R2.A1 AND
   ...
   R1.Am = R2.Am
WHERE
   R1.B1 <> R2.B1 AND
   ...
   R1.Bn <> R2.Bn;
```

2. En SQL, il faut comparer le résultats des requêtes suivantes qui calculent les cardinalités des projections (éventuellement en SQL avec un `MINUS` ou un `FROM DUAL`) :

```
SELECT COUNT(*)
FROM  (SELECT DISTINCT A1, ..., AM
       FROM r)

SELECT COUNT(*)
FROM  (SELECT DISTINCT A1, ..., AM, B1, ..., BM
       FROM r)
```

3. En théorie, la propriété est intéressante car elle remplace un éventuel produit cartésien $r \times r$ sur lequel on teste $\forall t_0, t_1.t_0[X] = t_1[X] \Rightarrow t_0[Y] = t_1[Y]$ par deux parcours linéaires de r. En pratique, avec la charge sur le serveur et les caches... ce n'est pas franc. Sur le cas d'espèce, la première DF n'est pas satisfaite, mais la seconde l'est.

```
SELECT COUNT(*)
FROM  (SELECT DISTINCT AA, AB, AC, AD, AE, AF, AG, AH, AI
       FROM dep_fun);

SELECT COUNT(*)
FROM  (SELECT DISTINCT AA, AB, AC, AD, AE, AF, AG, AH, AI, AJ
       FROM dep_fun);

SELECT count(*)
FROM dep_fun R1 INNER JOIN dep_fun R2 on
  R1.AA = R2.AA AND
  R1.AB = R2.AB AND
  R1.AC = R2.AC AND
  R1.AD = R2.AD AND
  R1.AE = R2.AE AND
  R1.AF = R2.AF AND
  R1.AG = R2.AG AND
  R1.AH = R2.AH AND
  R1.AI = R2.AI
WHERE R1.AJ <> R2.AJ;
```

4. C'est une base d'Armstrong pour l'ensemble $\{AA, AB, AC, AD, AE, AF, AG, AH, AI \rightarrow AJ\}$