

# M2106 : Programmation et administration des bases de données

Cours 2/6 – LDD, LMD & LCT

Guillaume Cabanac

guillaume.cabanac@univ-tlse3.fr



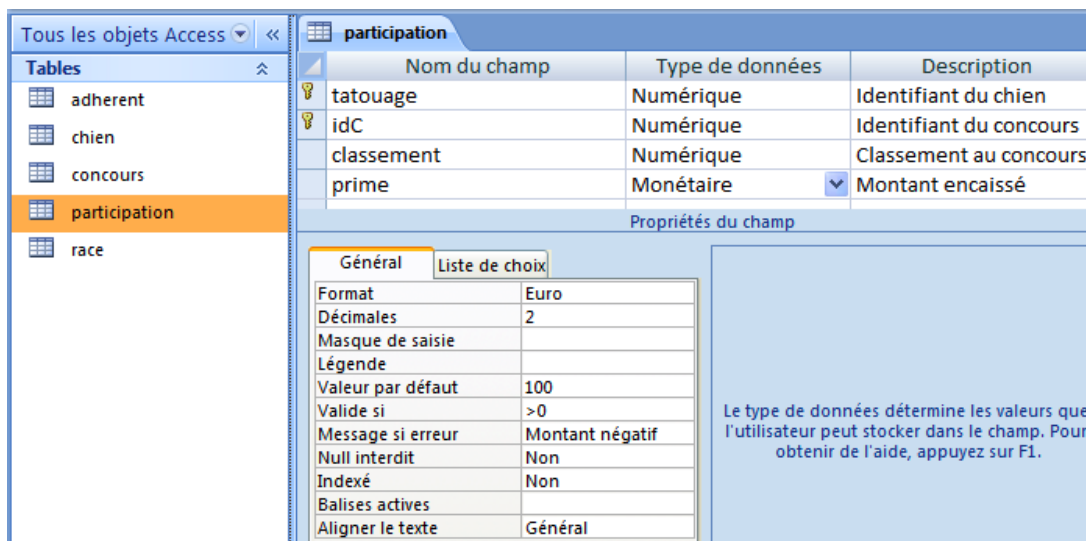
version du 18 février 2018

## Plan du cours

- 1 LDD : langage de définition des données
  - Colonnes
  - Tables
  - Contraintes
  - Tables et contraintes
- 2 LMD : langage de manipulation des données
  - Insertion : insert
  - Mise à jour : update
  - Suppression : delete
- 3 LCT : langage de contrôle des transactions
  - Validation : commit
  - Annulation : rollback

## Le LDD graphique avec Microsoft Access (1/2)

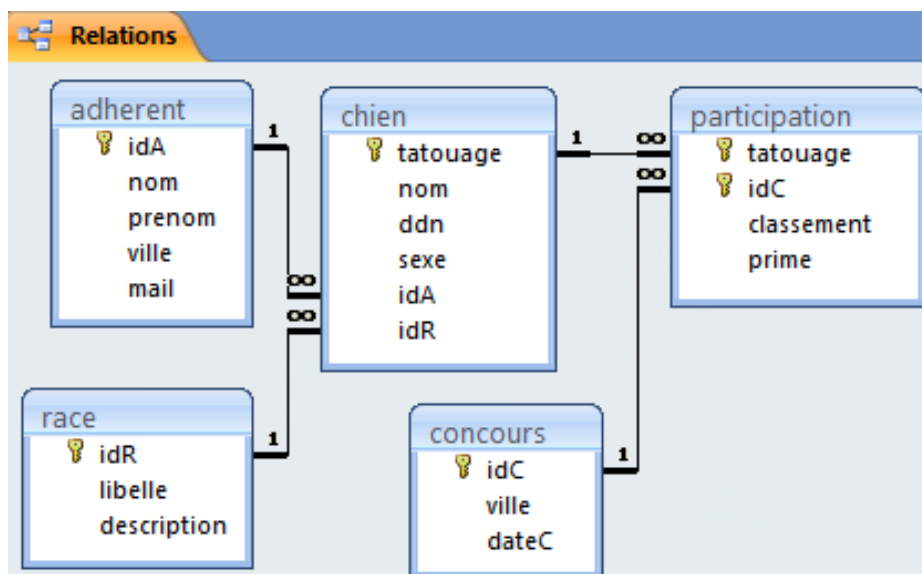
### Rappels de S1



Définition des tables avec leurs attributs et contraintes CK, PK, UN et NN

## Le LDD graphique avec Microsoft Access (2/2)

### Rappels de S1



Création des « liens » assurant l'intégrité référentielle entre les relations (FK)

## Typage des colonnes

Liste des types les plus couramment utilisés en SQL sous Oracle 11

Type Oracle	Description
<code>char(n [byte char])</code>	Chaîne de caractères de longueur fixe de $n \in [1; 4000]$ octets ( <b>byte</b> ) ou caractères ( <b>char</b> ).
<code>varchar2(n [byte char])</code>	Chaîne de caractères de longueur variable de $n \in [1; 4000]$ octets ( <b>byte</b> ) ou caractères ( <b>char</b> ).
<code>number[(p[, e])]</code>	Nombre avec précision $p$ et échelle $e$ $\overbrace{\text{XXXX,SSS}}^{e=3}$ $p = 4 + 3 = 7$
<code>date</code>	Date valide dans [01/01/-4712 ; 31/12/9999]. Champs : day, month, hour, minute et second.
<code>clob</code>	Chaîne de caractères de longueur variable.
<code>blob</code>	Chaîne binaire.

NB : '[X]' indique un élément optionnel X et 'X|Y' indique une alternative.

## Typage des colonnes

Précisions sur `char(n)` et `varchar2(n)`

⚠ Les types `char(n)` et `varchar2(n)` stockent des chaînes de caractères de longueur fixe ou variable :

- la chaîne « WTF » est stockée en `char(5)`

W	T	F		
---	---	---	--	--

,
- la chaîne « WTF » est stockée en `varchar2(5)`

W	T	F
---	---	---

.



☠ La longueur des chaînes est exprimée en **octets** par défaut :  
`varchar2(5) = varchar2(5 byte) ≠ varchar2(5 char)`.

« Dès Noël où un zéphyr haï me vêt de glaçons würmiens, je dîne d'exquis rôtis de bœufs au kir à l'aÿ d'âge mûr & cætera ! »

→ 120 caractères codés sur 136 octets...

cf. encodage à longueur variable en UTF-8 (ex : '€' = 3 octets!)

## Création de table (1/2)

### Syntaxe

```
create table nomTable
(
  colonne1 type1,
  colonne2 type2,
  ...
  colonneN typeN
) ;
```

### Exemple

```
create table chien
(
  tatouage number(6),           -- 166469
  nom       varchar2(15 char), -- 'Éliñāçãô' (0-15 char.)
  ddn       date,              -- 15/01/2013 12:45:03
  sexe     char(1),           -- 'F'
  idA      number(3),         -- 42
  idR      char(2),          -- null
) ;
```

## Interrogation du méta-schéma

Le **méta-schéma** du SGBD mémorise des informations sur les objets créés : utilisateurs, tables, vues, procédures, etc.

### Tables de l'utilisateur courant ?

```
select table_name from user_tables ;

-- TABLE_NAME
-- -----
-- CHIEN
```

### Description d'une table existante ?

```
desc chien      ----> Par défaut, tous les champs peuvent être mis à null !
--
-- Name      Null Type
-- -----
-- TATOUAGE   NUMBER(6)
-- NOM        VARCHAR2(15 CHAR)
-- DDN        DATE
-- SEXE       CHAR(1)
-- IDA        NUMBER(3)
-- IDR        CHAR(2)
```

## Création de table (2/2)



### Table créée sans aucune contrainte

```
create table chien
(
  tatouage number(6),           -- 166469
  nom       varchar2(15 char), -- 'Éliñaçaô' (0-15 char.)
  ddn       date,              -- 15/01/2013 12:45:03
  sexe      char(1),          -- 'F'
  idA       number(3),        -- 42
  idR       char(2)          -- null [cardinalité (0,1) du MCD]
);
```

Chien	tatouage	nom	ddn	sexe	idA	idR	
	166469	Éliñaçaô	15/01/2013	F	42		OK
		Milou	12/05/2011	M	42	CA	PK vide
	99		27/11/2012	M	4	RO	nom vide
	166469	Lassie	15/07/2011	F	54	CA	PK doublon
	666	Cerbère	08/03/2011	X	999		sexe incorrect
	42	Rantanplan	01/05/2009	M	14	ZZ	FK inexistant

## Rajout de contraintes PK, FK et CK

### Altération de la table pour rajouter des contraintes

```
alter table chien add
(
  constraint pk_chien      primary key(tatouage),
  constraint fk_chien_idA  foreign key(idA) references adherent,
  constraint fk_chien_idR  foreign key(idR) references race,
  constraint ck_chien_sexe check(sexe in ('M', 'F'))
);
```

Chien	tatouage	nom	ddn	sexe	idA	idR	
	166469	Éliñaçaô	15/01/2013	F	42		OK
		Milou	12/05/2011	M	42	CA	Réglé : pk_chien
	99		27/11/2012	M	4	RO	KO : nom vide
	166469	Lassie	15/07/2011	F	54	CA	Réglé : pk_chien
	666	Cerbère	08/03/2011	X	6		Réglé : ck_chien_sexe
	42	Rantanplan	01/05/2009	M	14	ZZ	Réglé : fk_chien_idR

Attention : par défaut toute colonne peut contenir la valeur null !

## Rajout de contraintes NN

### Altération de la table pour rajouter la contrainte NN

```
alter table chien modify
(
  nom constraint nn_chien_nom not null,
  sexe constraint nn_chien_sexe not null,
  idA constraint nn_chien_idA not null -- cardinalité (1,1) du MCD
);
```

Chien

tatouage	nom	ddn	sexe	idA	idR
166469	Éliñaçaô	15/01/2013	F	42	
	Milou	12/05/2011	M	42	CA
99		27/11/2012	M	4	RO
166469	Lassie	15/07/2011	F	54	CA
666	Cerbère	08/03/2011	X	6	
42	Rantanplan	01/05/2009	M	14	ZZ

OK

Réglé : pk\_chien

Réglé : nn\_chien\_nom

Réglé : pk\_chien

Réglé : ck\_chien\_sexe

Réglé : fk\_chien\_idR

NB : une PK ne peut pas contenir la valeur null → inutile de le spécifier.

## The big picture : création de table en 3 étapes

```
-- Étape 1. Table créée sans aucune contrainte
create table chien
(
  tatouage number(6), -- 166469
  nom varchar2(15 char), -- 'Éliñaçaô' (0-15 char.)
  ddn date, -- 15/01/2013 12:45:03
  sexe char(1), -- 'F'
  idA number(3), -- 42
  idR char(2) -- null [cardinalité (0,1) du MCD]
);

-- Étape 2. Altération de la table pour rajouter des contraintes PK, FK et CK
alter table chien add
(
  constraint pk_chien primary key(tatouage),
  constraint fk_chien_idA foreign key(idA) references adherent,
  constraint fk_chien_idR foreign key(idR) references race,
  constraint ck_chien_sexe check(sexe in ('M', 'F'))
);

-- Étape 3. Altération de la table pour rajouter la contrainte NN
alter table chien modify
(
  nom constraint nn_chien_nom not null,
  sexe constraint nn_chien_sexe not null,
  idA constraint nn_chien_idA not null -- cardinalité (1,1) du MCD
);
```

## Complément sur la mise en place de contraintes

Le SGBD exécute une requête `alter table` en 5 étapes :

- 1 Vérification **syntaxique**.  
Exemple : `alter chien table...` est invalide.
- 2 Vérification **sémantique**.  
Exemple : `unique(lo1)` est invalide car `lo1` n'existe pas dans `chien`.
- 3 Vérification de la **validité** des données.  
Exemple : `check(prime > 0)` est invalide dès qu'une ligne ne respecte pas cette condition (ex : une prime de `-5€` existante).
- 4 Mise en place des **traitements** nécessaires pour **garantir la contrainte**.
- 5 Enregistrement de la contrainte dans le **méta-schéma**.  
Exemple : une contrainte `check` apparaît dans `user_constraints`.

NB : l'exécution d'un `alter` invalide (étapes 1, 2 ou 3) est stoppée.

## Interrogation du méta-schéma

### Description de la table chien ?

```
desc chien      ----> Désormais, seuls DDN et IDR peuvent être nulls :-)  
--  
-- Name      Null      Type  
-----  
-- TATOUAGE  NOT NULL  NUMBER(6)  
-- NOM       NOT NULL  VARCHAR2(10 CHAR)  
-- DDN       DATE  
-- SEXE      NOT NULL  CHAR(1)  
-- IDA       NOT NULL  NUMBER(3)  
-- IDR       CHAR(2)
```

### Contraintes définies sur la table chien ?

```
select constraint_name, constraint_type, status, search_condition  
from user_constraints  
where table_name = 'CHIEN'  
order by constraint_name ;  
  
-- CONSTRAINT_NAME      CONSTRAINT_TYPE  STATUS      SEARCH_CONDITION  
-----  
-- CK_CHIEN_SEXE        C                ENABLED     sexe in ('M', 'F')  
-- FK_CHIEN_IDA         R                ENABLED  
-- FK_CHIEN_IDR         R                ENABLED  
-- NN_CHIEN_IDA         C                ENABLED     "IDA" IS NOT NULL  
-- NN_CHIEN_NOM         C                ENABLED     "NOM" IS NOT NULL  
-- NN_CHIEN_SEXE        C                ENABLED     "SEXE" IS NOT NULL  
-- PK_CHIEN             P                ENABLED
```

## Création d'une table avec ses contraintes (1/3)

### En une seule étape

```
create table chien
(
  tatouage    number(6)          constraint pk_chien    primary key,
  nom         varchar2(10 char) constraint nn_chien_nom  not null,
  ddn         date,
  sexe        char(1)           constraint nn_chien_sexe not null,
  idA         number(3)         constraint nn_chien_idA not null,
  idR         char(2),
  constraint fk_chien_idA foreign key(idA) references adherent,
  constraint fk_chien_idR foreign key(idR) references race,
  constraint ck_chien_sexe check(sexe in ('M', 'F'))
);
```

### Conventions d'écriture de code SQL :

- Indenter le code pour aligner les attributs, les types et les contraintes,
- Nommer les contraintes pk\_table, fk\_table\_col, ck\_table\_col, etc.
- Placer les contraintes en ligne pour pk non composées, ainsi que les nn.

## Création d'une table avec ses contraintes (2/3)

```
create table adherent
(
  idA         number(3)          constraint pk_adherent    primary key,
  nom         varchar2(20 char) constraint nn_adherent_nom  not null,
  prenom      varchar2(20 char) constraint nn_adherent_prenom not null,
  ville       varchar2(15 char) constraint nn_adherent_ville not null,
  mail        varchar2(20 char),
  constraint un_adherent_nomPrenom unique(nom, prenom),
  constraint ck_adherent_mail check(mail like '%@_%.__%')
);

create table race
(
  idR         char(2)           constraint pk_race        primary key,
  libelle     varchar2(15 char) constraint nn_race_libelle not null,
  description varchar2(50 char),
  constraint un_race_libelle unique(libelle)
);
```

### Remarques :

- Contraintes d'unicité un\_adherent\_nomPrenom et un\_race\_libelle,
- Contrainte de validité des données ck\_adherent\_mail.



## Création d'une table avec ses contraintes (3/3)

```
create table concours
(
  idC          number          constraint pk_concours      primary key,
  ville        varchar2(15 char) constraint nn_concours_ville not null,
  dateC        date            constraint nn_concours_dateC not null
);

create table participation
(
  tatouage     number(6),
  idC           number,
  classement   number(3),
  prime        number(6, 2),
  constraint pk_participation      primary key(tatouage, idC),
  constraint fk_participation_tatouage foreign key(tatouage) references chien,
  constraint fk_participation_idC   foreign key(idC) references concours,
  constraint ck_participation_classement check(classement > 0),
  constraint ck_participation_prime check(prime > 0),
  constraint ck_participation_primeClass check(prime is null or classement is not null)
);
```

### Remarques :

- Contrainte pk\_participation sur clé primaire composée,
- Contrainte de validité des données ck\_participation\_primeClass :  
prime → classement ⇔ ¬prime ∨ classement.

## Bilan sur les contraintes

Il existe 5 types de contraintes :

- PK** Les valeurs d'une colonne **Primary Key** (clé primaire) sont uniques, non nulles et indexées.
- FK** Les valeurs d'une colonne **Foreign Key** (clé étrangère) sont incluses dans les valeurs de la cible (qui est une colonne Unique).
- UN** Les valeurs d'une colonne **Unique** n'ont pas de doublons (null autorisés).
- CK** Les valeurs d'une colonne **Check** respectent une condition booléenne spécifiée.
- NN** Les valeurs d'une colonne **Not Null** sont toutes renseignées (null interdit).

## Compléments LDD à propos des tables (1/2)

### Création d'une table à partir du résultat d'une requête

```
-- Création "par recopie" de la structure et d'une partie
-- des données de la table chien
create table chienToulousain as
select *
from chien
where idA in (select idA
             from adherent
             where ville = 'Toulouse') ;
```

### Renommage d'une table

```
alter table chien rename to toutou ;
```

## Compléments LDD à propos des tables (2/2)

### Suppression de table : attention au problème des « enfants sans parents »

```
drop table chien ;
-- Error: ORA-02449: unique/primary keys in table referenced by foreign keys
-- *Cause: An attempt was made to drop a table with unique or
--         primary keys referenced by foreign keys in another table.

drop table participation ;
-- table PARTICIPATION dropped.

drop table chien ;
-- table CHIEN dropped.
```

### Suppression de table et des FK entrantes

```
drop table chien cascade constraints ;
```

## Compléments LDD à propos des colonnes

### Ajout d'une colonne

```
alter table chien
add prixAchat number(6,2) constraint nn_chien_prixAchat not null ;
```

### Renommage d'une colonne

```
alter table chien rename column sexe to genre ;
```

### Modification d'une colonne

```
alter table chien modify prixAchat number(10, 2) ; -- aug. de la précision
alter table chien modify prixAchat default 50 ; -- spéc. valeur par défaut
```

### Suppression d'une colonne

```
alter table chien drop column prixAchat ;
```

## Compléments LDD à propos des contraintes

### Désactivation et activation de contrainte

```
alter table chien disable constraint ck_chien_sexe ;

alter table chien enable constraint ck_chien_sexe ; -- ∃ option novalidate

-- cas particulier de la PK, qui peut être traité sans connaître
-- le nom de la contrainte pk_chien
alter table chien disable primary key ;
alter table chien enable primary key ;
```

### Suppression de contrainte

```
alter table chien drop constraint ck_chien_sexe ;

-- cas particulier de la PK, qui peut être traité sans connaître
-- le nom de la contrainte (cascade supprime au préalable les FK -> PK)
alter table chien drop primary key ;
alter table chien drop primary key cascade ;
```

## Insertion de lignes (1/7)

### Syntaxe pour insérer une seule ligne

```
insert into nomDeTable [(col1, ..., colN)]
  values (val1, ..., valN) ;
```

### Exemples

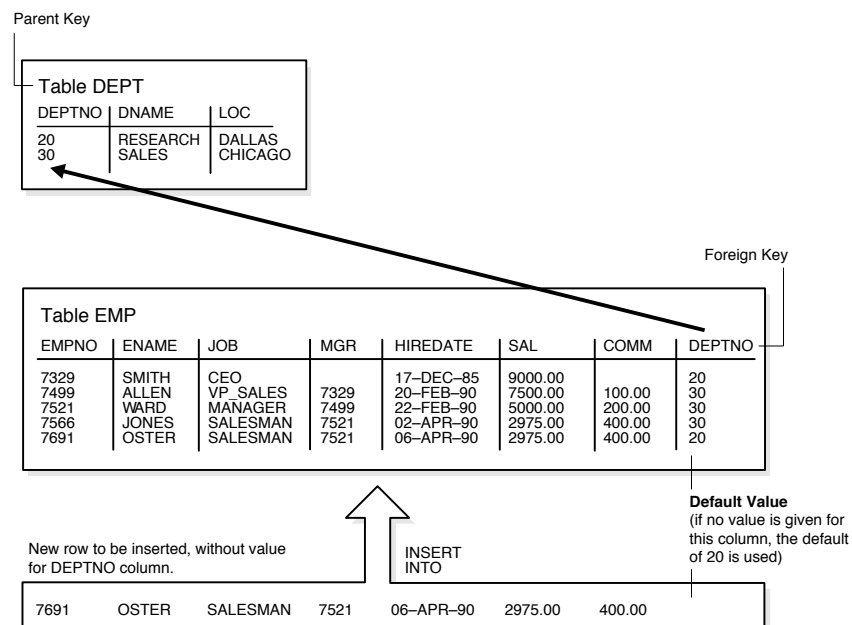
```
-- L'ordre des valeurs est celui des colonnes de la table.
insert into chien
  values (666, 'Cerbère', '06-JUN-1966', 'M', 42, null) ;

-- L'ordre est spécifié.
-- Valeur "default" pour les colonnes non listées.
insert into chien (nom, sexe, tatouage, idA, ddn)
  values ('Cerbère', 'M', 666, 42, '06-JUN-1966') ;
```

## Insertion de lignes (2/7)

Illustration issue de la documentation Oracle

Figure 5-4 DEFAULT Column Values



## Insertion de lignes (3/7)

### Syntaxe pour insérer des lignes à partir d'une requête

```
insert into nomDeTable [(col1, ..., colN)]
  select val1, ..., valN
  from ...
  where ...
  group by ...
  having ...
  order by ... ;
-- ou toute autre requête avec union, intersect, minus...
```

### Exemple

```
-- Ajout des chiens de la Société Canine de Bordeaux
insert into chien (tatouage, nom, ddn, sexe, idA)
  select tatouage, nom, ddn, sexe, idA
  from chienSocieteCanineBordeaux ;
```

## Insertion de lignes (4/7)

### Illustration

```
insert into adherent
  values (42, 'Satan', 'Satan', 'Enfer', null) ;
-- 1 rows inserted.

select * from chien ;
-- no rows selected

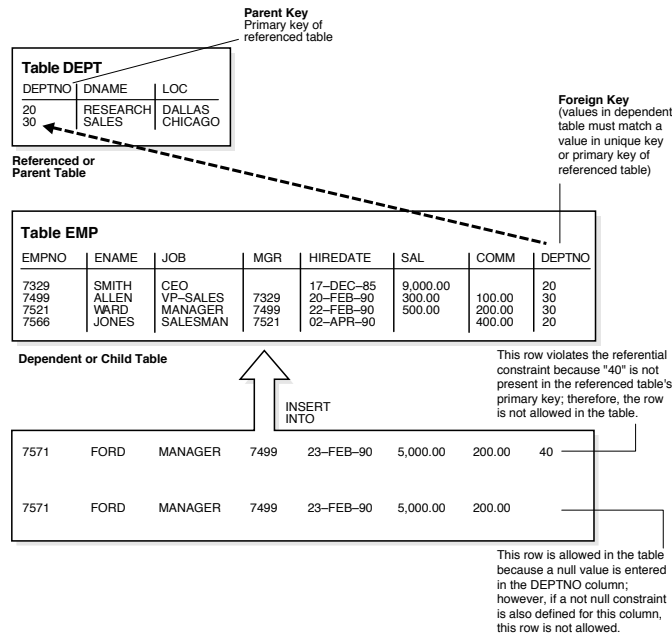
insert into chien
  values (666, 'Cerbère', '06-JUN-1966', 'M', 42, null) ;
-- 1 rows inserted.

select * from chien ;
--   TATOUAGE NOM          DDN          SEXE          IDA IDR
--   -----
--           666 Cerbère    06-JUN-66 M           42
```

## Insertion de lignes (5/7)

Illustration issue de la documentation Oracle

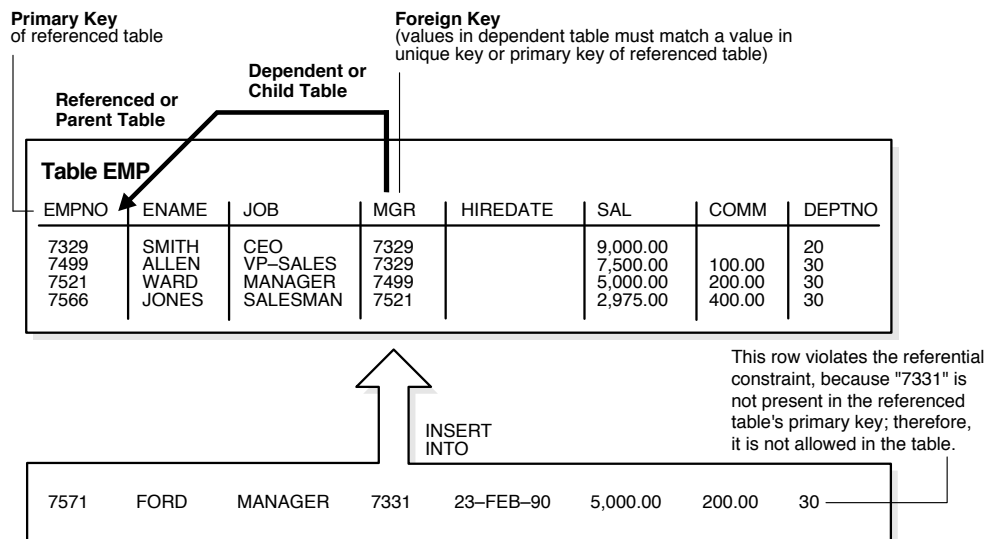
Figure 21-1 Referential Integrity Constraints



## Insertion de lignes (6/7)

Illustration issue de la documentation Oracle

Figure 21-2 Single Table Referential Constraints



## Insertion de lignes (7/7)

### Cas des colonnes aux valeurs auto-incrémentées

Il n'existe pas de type entier auto-incrémenté en *Oracle 11g*, contrairement à MS Access. On utilise alors une **séquence**.

#### Syntaxe de création de séquence

```
create sequence nomDeLaSequence  
  [start with N]          -- par défaut N = 1  
  [increment by M] ;    -- par défaut M = 1
```

#### Exemple d'utilisation

```
create sequence seqId ; -- accès à seqId.nextVal et seqId.currVal  
  
insert into chien  
  values (seqId.nextVal, 'Lassie', sysdate, 'F', 5, 'CO') ;
```

## Mise à jour de lignes (1/3)

#### Syntaxe 1 : colonne = valeur

```
update nomDeTable  
  set col1 = val1 [, ..., colN = valN]  
  [where ... ] ;
```

NB : valI peut être fournie par une sous-requête.

#### Exemples

```
-- Clint déménage et nous communique son mail.  
update adherent  
  set ville = 'Toulouse', mail = 'clint@eastwood.biz'  
  where nom = 'Eastwood'  
    and prenom = 'Clint' ;  
  
-- Tous les chiens sont en fait des bâtards.  
update chien  
  set idR = null ;
```

## Mise à jour de lignes (2/3)

### Illustration

```
select * from adherent ;
--  IDA NOM          PRENOM          VILLE          MAIL
-----
--  42 Satan        Satan          Enfer
--  51 Mulder        Fox            Muret          mulder@fbi.gov

update adherent
  set prenom = 'Lucifer'
  where idA = 42 ;
-- 1 rows updated.

select * from adherent ;
--  IDA NOM          PRENOM          VILLE          MAIL
-----
--  42 Satan        Lucifer        Enfer
--  51 Mulder        Fox            Muret          mulder@fbi.gov
```

## Mise à jour de lignes (3/3)

### Syntaxe 2 : colonnes = valeurs d'une sous-requête

```
update nomDeTable
  set (col1, ..., colN) = (select col1, ..., colN from ...)
  [where ... ] ;
```

NB : la sous-requête doit retourner une ligne au plus.

### Exemple

```
-- Hypothèse : Les colonnes chien.nbPart et chien.primMax ont été créées précédemment.
-- NB : la sous-requête est synchronisée = exécutée à chaque ligne de chien traitée.
update chien
  set (nbPart, primeMax) = (select count(*), max(prime)
                           from participation
                           where tatouage = chien.tatouage) ;

-- équivalent à where participation.tatouage = chien.tatouage
```



## Suppression de lignes (1/2)

### Syntaxe

```
delete [from] nomDeTable  
[where ... ] ;
```

### Exemples

```
-- Suppression de toutes les participations.  
delete participation ;  
  
-- Suppression des adhérents qui ne sont pas propriétaire de chiens.  
delete adherent  
  where idA not in (select idA -- sous-requête calculée une seule fois  
                  from chien) ;  
  
-- Idem, mais avec une requête synchronisée.  
delete adherent  
  where 0 = (select count(*) -- sous-requête calculée pour chaque ligne  
            from chien  
            where idA = adherent.idA) ; -- ⇔ where chien.idA = adherent.idA
```

## Suppression de lignes (2/2)

### Illustration

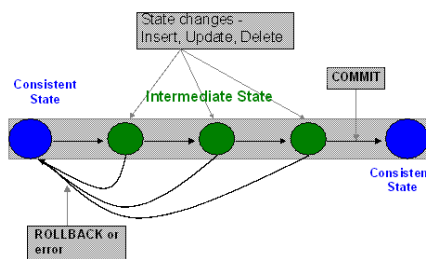
```
select * from adherent ;  
-- IDA NOM PRENOM VILLE MAIL  
-----  
-- 42 Satan Lucifer Enfer  
-- 51 Mulder Fox Muret mulder@fbi.gov  
  
delete adherent  
  where mail is null ;  
-- 1 rows deleted.  
  
select * from adherent ;  
-- IDA NOM PRENOM VILLE MAIL  
-----  
-- 51 Mulder Fox Muret mulder@fbi.gov
```

## Principe des transactions (1/3)

### Définition

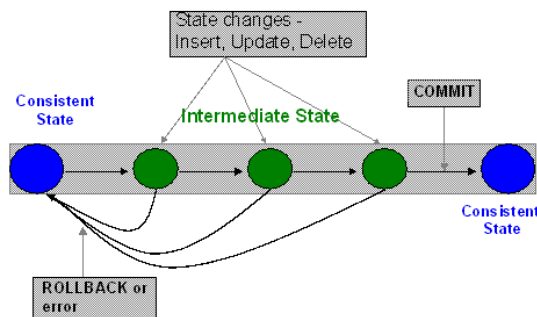
Une **transaction** est une séquence de une ou plusieurs **instructions SQL**, caractérisée par :

- 1 un **état initial** des données qui est **cohérent** : le début de transaction  
→ connexion ou **commit**,
- 2 des **états intermédiaires** des données qui sont **incohérents**
- 3 un **état final** des données **cohérent** : la fin de transaction  
→ déconnexion « propre », **commit** ou exécution d'un ordre du LDD



[http://help.sap.com/saphelp\\_46c/helpdata/en/41/7af4bca79e11d1950f0000e82de14a/Image673.gif](http://help.sap.com/saphelp_46c/helpdata/en/41/7af4bca79e11d1950f0000e82de14a/Image673.gif)

## Principe des transactions (2/3)



[http://help.sap.com/saphelp\\_46c/helpdata/en/41/7af4bca79e11d1950f0000e82de14a/Image673.gif](http://help.sap.com/saphelp_46c/helpdata/en/41/7af4bca79e11d1950f0000e82de14a/Image673.gif)

**commit** Valide les modifications de la transaction courante.  
→ écriture sur disque.

**rollback** Annule les modifications effectuées durant la transaction.

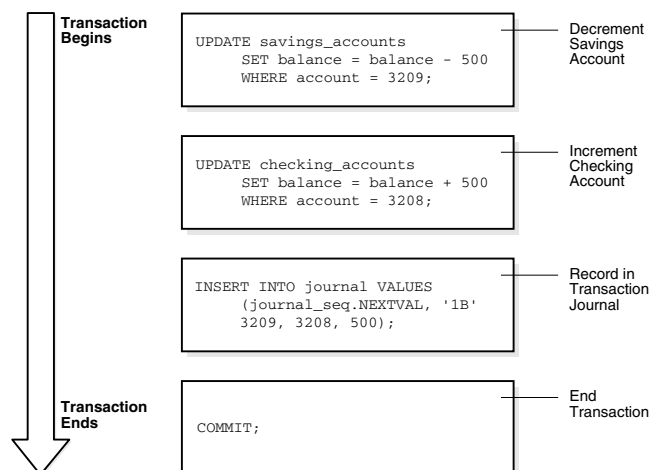
NB : une panne du SGBD ou une perte de connexion fait un **rollback** automatique.

⇒ Analogie avec un fichier de traitement de texte.

## Principe des transactions (3/3)

Les **transactions** permettent la mise en œuvre de modifications impliquant plusieurs instructions SQL, sur le **principe du « tout ou rien »**.

Figure 10-1 A Banking Transaction



Exemple de transaction validée par **commit**

## Sources

Les illustrations sont reproduites à partir du document suivant :

- Oracle Database Documentation Library 11.1  
(<http://www.oracle.com/pls/db111/homepage>)
  - Oracle® Database : Concepts 11g Release 1 (11.1) B28318-06
- Oracle Database Documentation Library 11.2  
(<http://www.oracle.com/pls/db112/homepage>)
  - Oracle® Database : Concepts 11g Release 2 (11.2) E40540-01