RAPPELS

CREATE TABLE Pommier
(NumP NUMBER NOT NULL*,*
NomP VARCHAR2(25) NOT NULL
REFERENCES *Pomme (Variété),*
Plantation DATE*,*
NumR REAL DEFAULT*(5),*
PRIMARY KEY *(NumF, NomF),*
CHECK *NumR > NumP,*
UNIQUE *(DateLiv, NomF));*

DROP TABLE *nom*;

INSERT INTO *Pommier* VALUES (*12*, '*Pink Lady*');

UPDATE *Pommier* SET *Plantation* = '*12-JUL-06*',
NumR = 4 WHERE *NumP = 1;*

DELETE FROM

SELECT * FROM *Pommier*;

SELECT DISTINCT *NomP, NumR* FROM *Pommier*;

requête UNION *requête*
requête INTERSECT *requête*
requête EXCEPT *requête*

ORDER BY *attribut1* ASC, *attribut2* DESC ;

CREATE VIEW *nom* (*liste d'attributs)*
AS *requête SELECT*;

DROP VIEW *nom*;

Conditions de sélection
WHERE *NomP* LIKE '%.';
-> l'attribut finit par un point

WHERE *NomP* LIKE 'b_d'
-> l'attribut est 'bad', 'bdd'...
(_ remplace un seul caractère et % remplace une chaine de caractères)

WHERE *NomP* IS NULL;

WHERE *NumR* Not IN *(1,3,5,7,9)*;

WHERE *Plantation* BETWEEN '*01-JAN-04*' AND '*31-DEC-04*';
WHERE *NumP* >= ANY *(1, 2, 3)*;

WHERE *NumP* >= ALL *(1, 2, 3*);

WHERE (NOT) EXISTS (SELECT …);

Fonction de calcul
SELECT AVG (*nom_attribut*)
-> moyenne des valeurs prises par *nom_attribut*

SELECT SUM (*nom_attribut*)
-> somme des valeurs prises par *nom_attribut*

SELECT MIN (*nom_attribut*)
-> minimum des valeurs prises par *nom_attribut*

SELECT MAX (*nom_attribut*)
-> maximum des valeurs prises par *nom_attribut*

SELECT COUNT (*nom_attribut*)
-> nombre de valeurs dans l'ensemble du résultat (nombre de tuples)

SELECT *nom_attribut + nom_attribut;*

(SQL 89)
FROM *nom rel1, nom rel2* WHERE *rel1.attribut = rel2.attribut*;

(SQL 92)
FROM *nom de la relation 1* NATURAL JOIN *nom de la relation 2*;

FROM *nom de la relation 1* JOIN *nom de la relation 2* USING (*attribut*);

FROM *nom de la relation 1* JOIN *nom de la relation 2* ON *rel1.attribut = rel2.attribut*;

FROM *nom de la relation 1* LEFT OUTER JOIN *nom de la relation 2* ON *rel1.attribut = rel2.attribut*;

FROM *nom de la relation 1* RIGHT OUTER JOIN *nom de la relation 2* ON *rel1.attribut = rel2.attribut*;

FROM *nom de la relation 1* FULL OUTER JOIN *nom de la relation 2* ON *rel1.attribut = rel2.attribut*;

Partitionnement

GROUP BY

GROUP BY permet de partitionner la relation en sous-relations ayant les mêmes valeurs sur les attributs précisés : on peut alors appliquer des **fonctions de calcul** (voir rappel) aux attributs de chaque sous-groupe.
Des attributs ne peuvent être présents après le SELECT si et seulement si ils sont dans le GROUP BY.

SELECT *
FROM *nom de la relation*
WHERE *condition*
GROUP BY *attribut, attribut* ;

HAVING

La clause HAVING permet de poser une condition portant sur chacune des sous-parties générées par un GROUP BY. Les sous-parties ne remplissant pas cette condition sont écartées du résultat.

SELECT *
FROM *nom de la relation*
WHERE *condition*
GROUP BY *attribut, attribut*
HAVING *Fonction de calcul*; (ex : HAVING MAX(NumR)>= 5)

Ordre

SELECT …
FROM …
WHERE …
GROUP BY …
HAVING …
ORDER BY …

Définition des données

Il y a trois types de commandes de définition de données:
- `CREATE TABLE nom_relation …` *(on connait)*
- `DROP TABLE nom_relation` *(on connait)*
- `ALTER TABLE nom_rel …` : *(modifie la structure de la table)*
  - … `ADD nom_attribut type_attribut` *(ajoute un nouvel attribut)*
  - … `ADD CONSTRAINT contrainte` *(ajoute une nouvelle contrainte)*
  - … `ALTER nom_attribut nouveau_nom` *(change le nom d'un attribut)*
  - … `ALTER nom_att SET DEFAULT val_defaut` *(change/ajoute la valeur par défaut d'un attribut)*
  - … `ALTER nom_attribut DROP DEFAULT` *(enlève la valeur par défaut d'un attribut)*
  - … `DROP COLUMN nom_attribut` *(enlève un attribut)*
  - … `DROP CONSTRAINT nom_contrainte` *(enlève une contrainte)*

Utilisateurs

Créer/Supprimer

CREATE USER *nom*	**(oblig.)** le nom **unique** d'utilisateur
IDENTIFIED BY *mot_de_passe* ou EXTERNALLY	**(oblig.)** identification : mdp Oracle ou mdp OS
DEFAULT TABLESPACE *nom*	espace disque pour objets propres à l'user
TEMPORARY TABLESPACE *nom*	espace disque temporaire (tris, etc)
QUOTA *nombre* K ou *nombre* M ou UNLIMITED	taille de l'espace de travail (en Ko, Mo ou illimité)
PROFILE *nom*	affecté par défaut
PASSWORD EXPIRE;	si on met ça, on doit changer le mdp à la 1ere co

DROP USER *nom* CASCADE	**Supprime l'user**. Avec CASCADE, supprimer tous ses objets (relations, vues, contraintes, procédures...)

Permissions

Privilèges objets
Droits limités à un objet de la base : SELECT, UPDATE, INSERT, DELETE... (manipulation de données)

GRANT *liste privilèges* ou ALL PRIVILEGES	Donne les privilèges de la liste ou tt les privilèges
ON TABLE *liste nomrel* ou ON *liste nomvue*	Sur telles relations ou sur telles vues
TO *liste usernames* ou PUBLIC	Aux user de la liste ou à tt les users
WITH GRANT OPTION;	(optionel) les utilisateur de la liste après TO peuvent redistribuer les privilèges aprs GRANT

- SELECT, DELETE, INSERT, EXECUTE s'appliquent sur relations/vues entières. SI on veut limiter à certains attributs / certaines valeurs des attributs, il faut créer une nouvelle vue avec seulement ces attributs puis donner les privilèges sur cette vue. Les privilèges sur la vue dépendent aussi des privilèges sur la relation initiale. Si on a aucun droit sur la relation, on ne pourra pas modifier la vue.
- UPDATE et REFERENCES s'appliquent sur un attribut /une liste d'attributs qui doivent donc être précisés ex : GRANT UPDATE (*attribut1, attribut2*) ON TABLE *Pomme* TO *Sbdy*
- Le créateur d'une relation a tous les droits sur celle-ci et peut donc les accorder à d'autres users

Privilèges systèmes
Droits d'exécuter des actions portant sur tout le système : CREATE/ALTER/DROP (définition de données)

GRANT *liste privilèges* ou ALL PRIVILEGES	Donne les privilèges de la liste ou tt les privilèges
TO *liste usernames* ou PUBLIC	Aux user de la liste ou à tt les users
WITH ADMIN OPTION;	(optionel) les utilisateur de la liste après TO Peuvent redistribuer les privilèges après GRANT

GRANT CREATE TABLE TO Pomme -> Pomme peut créer des tables dans SON schéma
GRANT CREATE **ANY** TABLE TO Pomme -> Pomme peut créer des tables dans TOUS les schémas

Suppression des privilèges
Un utilisateur peut seulement retirer les privilèges qu'il a reçus. Ce n'est pas possible de retirer uniquement le GRANT OPTION.
A l'exécution d'un REVOKE, l'utilisateur A à qui les privilèges ont été retirés perd ces privilèges sauf s'il les a aussi reçus d'une autre source indépendante de l'utilisateur B (qui exécute le REVOKE). Les personnes ayant reçus leurs privilèges par l'utilisateur A ou par quelqu'un les ayant reçus de A perdent aussi leurs privilèges sauf s'ils les ont aussi reçus d'une source indépendante de A.

REVOKE *liste privilèges* ou ALL PRIVILEGES
ON *liste nomrel, nomvue*
TO *liste usernames* ou PUBLIC;

Enlève les privilèges de la liste ou tt les privilèges
Sur telles relations ou sur telles vues
Aux user de la liste ou à tt les users

Maintien de la cohérence

Contraintes d'intégrité : propriétés que doivent respecter les données pour être conformes au monde réel qu'elles représentent.

Une transaction commence par une instruction `SET TRANSACTION READ ONLY;` ou `SET TRANSACTION READ WRITE;`.

Ensuite, on a des interrogations et/ou des mises à jour (SELECT, UPDATE).

Il faut toujours lancer la commande `commit` (pour sauvegarder) ou `rollback` (pour annuler commplètement) à la fin d'une transaction.

Base de donnée cohérente : base de données inactive dont toutes les données vérifient les ocntraintes d'intégrité.

Transaction : unité cohérente de traitement

Transactions

ex :

```
SET TRANSACTION READ WRITE;

INSERT INTO Pomme VALUES (…);
UPDATE Pomme SET …;
INSERT INTO Pomme VALUES (…);

COMMIT;
```

Contraintes

Le format général d'un contrainte est :
`CONSTRAINT nom-contrainte expression_contrainte vérification`
Seule `expression_contrainte` est toujours obligatoire.

Il est bon de prendre l'habitude de nommer toutes ses contraintes afin de pouvoir le modifier ultérieurement.

```
numEtu NUMBER CONSTRAINT cle_numEtu REFERENCES Etudiant (num);
prix NUMBER CONSTRAINT prix_min CHECK (prix>5);
```

Types de `contraintes`

- Définition de domaine : TypePomme = {Gala, PinkLady, Reinette} -> un domaine c'est une sous-classe de VARCHAR, de DATE…
- Type de données : LIBELLE : TEXTE
- Plage de valeur : 1≤ N° PRODUIT ≤ 1000
- Énumération de valeurs : **VILLE IN** (PARIS, LYON, BORDEAUX, ÉPINAL)
- Contrainte de non-nullité : **NOT NULL**
- Contrainte entre attributs d'un même tuple : PrixHT ≤ PrixTTC
- Contrainte temporelle : compare l'ancienne valeur à la valeur après mise à jour
- Unicité : **UNIQUE**
- Clé : clés primaires
- Contrainte d'agrégats : **HAVING**
- Contraintes référencielles : clés étrangères
- Dépendances d'inclusion (un peu comme des clés étrangères) :
  - sous-type : toutes les valeurs de l'attribut NumA de Pomme doivent être des valeurs de l'attribut NumA de Arbre parce que Pomme.NumA est un clé qui fait référence à Arbre.NumA
  - inclusion entre attributs non clé : même chose mais les deux attributs sont dans la même relation

Création de domaine

```
CREATE DOMAIN nom AS type_données
      DEFAULT valeur (facultatif)
      contrainte (facultatif)

contrainte –>
CHECK (condition de sélection SQL)
```

Exemple
CREATE DOMAIN Couleur AS varchar(20)
CONSTRAINT domain-c1
CHECK (VALUE IS NOT NULL)
CONSTRAINT domain-c2
CHECK (VALUE IN ('Rouge', 'Bleu', 'Vert'));

Pour les autres (hors clés et définis dans le rappel/dans la partie Types de contraintes) : CHECK (condition)
ex : CHECK (att1 > att2)

Vérifications

Il y a deux type de `vérification` des contraintes :
- `NOT DEFERRABLE` : vérification à la fin de chaque mise à jour. C'est le défaut en SQL 92.
- `DEFERRABLE [INITIALLY DEFERRED ou INITIALLY IMMEDIATE]` : vérification
  - soit différée au `commit` (`INITIALLY DEFERRED`)
  - soit à l'exécution de chaque requête SQL de mise à jour (`INITIALLY IMMEDIATE`)

NOT DEFERRABLE et INITIALLY IMMEDIATE ont le même résultat final. La différence est que l'on ne peut pas modifier une contrainte NOT DEFERRABLE, alors qu'on peut modifier une contrainte DEFERRED pour changer le mode (DEFERRED ou IMMEDIATE) via un SET CONSTRAINT :

```
SET CONSTRAINT ALL DEFERRED;
SET CONSTRAINT cle1, cle2 IMMDIATE;
```

Cela est particulièrement utile lors de références cycliques (*rel1 a une clé référentielle ver rel2 et rel2 a une clé référentielle vers rel1)*. Si les contraintes de clés référentielles sont toutes les deux DEFERRABLE, il suffit d'ajouter les nouveaux tuples dans la même transaction et de finir par commit.

Trigger

Trigger référenciel : Action qui se déclenche automatiquement pour satisfaire les contraontes référentielles. Les triggers sont associés à la définition de contraintes référentielles.

Il y a des triggers `ON DELETE` et des triggers `ON UPDATE`, dans les deux cas ils sont suivi soit de :
- `NO ACTION` : il ne se passe rien lors de la mise à jour / la suppression d'un tuple.
- `CASCADE` : le changement (mise à jour ou suppression d'un tuple) est répercuté sur tous les tuples qui référencent le tuple modifié.
- `SET DEFAULT` : la clé étrangère prend la valeur par défaut si elle existe.
- `SET DEFAULT` : la clé étrangère prend la valeur NULL si c'est possible.

```
num_projet NUMBER REFERENCES Projet ON UPDATE SET NULL ON DELETE SET NULL);
```
—> Si num_projet de Projet est modifié, ou si on supprime un tuple de projet, l'attribut num_projet de la table ci-dessus est mis à nul pour les tuples concernés.

```
num_projet NUMBER REFERENCES Projet ON UPDATE CASCADE ON DELETE CASCADE);
```
—> Si num_projet de Projet est modifié, l'attribut num_projet de la table ci-dessus est modifié de manière identique pour les tuples concernés. De même, si un tuple de projetest supprimé, les tuples concernés de cette table sont eux aussi supprimés. Cela ne marche pas lors d'un DROP TABLE.