SQL le langage relationnel

Langage de Définition des Données

Présentation

- La fonction essentielle du LDD est la définition du schéma de la base de données (définition de la structure : tables) et des contraintes d'intégrité

Les dictionnaires

- Les informations portant sur la structuration et l'implantation des données sont rassemblées dans des relations spécifiques appelées DICTIONNAIRES.
- L'ensemble des dictionnaires forme une base spécifique appelée METABASE.
- Des instructions spécifiques permettent à l'administrateur de la base de données de lire, créer ou modifier les informations contenues dans les dictionnaires.

Représentation des données

- Dans une base relationnelle, les données sont représentées dans des tables qui matérialisent les relations.
- La notion de domaine n'est pas directement exprimable.
- Chaque attribut est caractérisé par un type de données.

Quelques types de données

- Type caractère : CHAR
  - CHAR(longueur) < 4000 car. / fixe
  - VARCHAR2(longueur) < 4000 car. / variable
  - VARCHAR(longueur) = VARCHAR2
- Type numérique : NUMBER
  - entiers, décimaux, nombres en virgule flottante (= nombres les plus souvent utilisés dans un ordinateur pour représenter des valeurs non entières. Ce sont des approximations de nombres réels. Sert à stocker les nombres dans la machine = double ou float nombre réel)

Quelques types de données

- Paramètres : NUMBER[(précision [,échelle])]
  - Précision : nombre entier de chiffres significatifs, de 1 à 38
  - Echelle : nombre de chiffres à droite de la décimale)
- Type date : DATE
  - Format standard : DD-MON-YY
- Type long : LONG VARCHAR
  - Chaînes de caractères < 2 gigas octets
  - Ni dans une expression, ni dans un prédicat

Quelques types de données

- Type binaire : RAW
  - Longueur maximale : 255 octets
  - Insertion ou modification d'une donnée de type Raw en spécifiant sa valeur hexadécimale
- Type binaire long : LONG RAW
  - <= 2 gigaoctets

Quelques types de données

- Type Gros Objets : LOB (Large Objects)
  - Gestion de données non structurées : image, sons, vidéo, texte
    - LOB interne
      - CLOB : documents (4 gigaoctets)
      - BLOB : graphiques, sons, vidéo (4 gigaoctets)
    - LOB externe
      - BFILE : permet de stocker une liste de pointeurs vers des fichiers externes

CREATION ET MODIFICATION DES TABLES

Création des tables

- On utilise le verbe CREATE

```
CREATE Table Etudiant
n°SS_etu number (6) not null
nom_etu char (20)
adr_etu char (80)
date_nais date
code_diplôme number (3)
moyenne number (4,2);
```

- Ceci permet la création de la relation au niveau du dictionnaire

Définition des clés

- La clé principale est définie à l'aide de la fonction Primary key
- Les autres clés (dites clés candidates) sont déclarées en utilisant la fonction unique

```
CREATE table professeur
num_prof number (6) NOT NULL PRIMARY KEY
nom_prof char (20) NOT NULL
n°SS_ens number (13) UNIQUE;
```

Définition des clés

- Autre formulation (après l'énumération des attributs)

```
CREATE TABLE commande
(numero_cde NUMBER,
date_cde DATE
…..
CONSTRAINT pk_num_cde PRIMARY KEY (numero_cde));
```

Définition des clés étrangères

- On appelle clé étrangère ou clé externe dans une table T2 toute colonne (ou combinaison de colonnes) qui apparaît comme clé primaire dans une table T1 (appelée table primaire ou table source).
- L'intégrité référentielle permet de faire référence par un attribut d'une relation à une clé d'une autre relation dont la valeur doit exister lorsque l'on donne une valeur à l'attribut considéré (fonction REFERENCES) .

Définition des clés étrangères

- Représentation

```
CREATE TABLE salle
( nom_salle varchar2(20),
nom_bat varchar2(20), ...
CONSTRAINT pk_salle PRIMARY KEY(nom_salle,nom_bat) )
CREATE TABLE cours
( nom_cours varchar2(20), ...
nom_salle varchar2(20),
nom_bat varchar2(20),
CONSTRAINT fk_salle_cours FOREIGN KEY(nom_salle,nom_bat)
   REFERENCES salle(nom_salle,nom_bat);
```

Mise à jour de la table référencée

CREATE TABLE client

(num_client number(3,0),

nom Varchar2(10),

...,

CONSTRAINTclient_pk PRIMARY KEY (num_client)

CREATE TABLE commande

(num_commande NUMBER(3,0)

CONSTRAINTcom_pk PRIMARY KEY

date_commande DATE not null,

num_client NUMBER(3,0)

CONSTRAINTcom_client_fk REFERENCES client

(ou bien CONSTRAINTcom_client FOREIGN KEY(num_client) REFERENCESclient (num_client)

Mise à jour d'une table référencée

NO ACTION : interdit la modification (update de la clef primaire ou delete) dans la table référencée.
Sur l'exemple, on peut supprimer un client qui n'a pas de commande mais pas un client qui est référencé dans une commande.

CASCADE : les lignes qui référencent sont détruites (on delete cascade) ou mises à jour (on update cascade). Par exemple, si on déclare la clef étrangère num_client dans la table COMMANDE avec la clause on delete cascade, la suppression d'un client entraîne la suppression de toutes ses commandes.

Mise à jour d'une table référencée

- SET NULL : la clef étrangère prend la valeur NULL en cas de suppression (on delete set null) ou de modification (on update set null) de la clef référencée.
- SET DEFAULT : sur le même principe que le point précédent, la clef étrangère prend une valeur par défaut en cas de suppression/modification de la clef référencée

Modification d'une table

- Modification d'une table : ALTER
- Exemple : ALTER table etudiant
  - ajout d'une colonne : ADD

```
ALTER table etudiant
ADD (num_SS) number (13);
```

  - modification des caractéristiques d'une colonne

```
ALTER table etudiant
MODIFY (adr_etu char (100));
```

Modification d'une table

- Ajout ou suppression de contraintes : on peut ajouter des contraintes à une table contenant déjà des tuples :

  ALTER Table Client

  ADD CONSTRAINT ck_ville CHECK (Ville IN('Toulouse', 'Grenoble', 'Paris')); → le contenu de la table est alors vérifié et des rejets sont possibles.

- Les contraintes peuvent également être supprimées :

  ALTER Table Client

  DROP CONSTRAINT ck_client_nomcli;

Autres opérations sur une table

- Suppression d'une table : DROP
  - DROP table ETUDIANT
- Changement du nom d'une table : RENAME
  - RENAME <ancien_nom> TO <nouveau_nom>

- Exemple : enlever la colonne Ville de la table Client
  - Soit la relation suivante : Client (cocli, nomcli, ville)

```
CREATE Table Travail
   AS SELECT cocli, nomcli FROM Client;
DROP Table Client;
RENAME Travail TO Client;
```

OPÉRATIONS SUR LES DONNÉES

Insertion de données

- Chargement des données : INSERT
- Syntaxe

```
INSERT into table
VALUES(valeur_1, valeur_2);
```

- Exemple

INSERT into table Etudiant
VALUES (009658, ‘ AUREL Marc’, ‘ 6 rue des tulipes, 69001 Lyon’, to_date (‘ 01/02/1972, DD/MM/YY’), 32, 12.45);

Création et insertion simultanées

- Syntaxe

```
CREATE table nom_table
as SELECT….;
```

- Exemple

```
CREATE recap_étudiant
as SELECT   n°SS_étu, nom_étu
            COUNT(nb_inscriptions) total
            FROM etudiant
            GROUP BY nom_étu;
```

Modification des données

- On utilise la clause UPDATE
- Permet de modifier les valeurs d'un ou de plusieurs attributs dans une ou plusieurs occurrences de la relation
- Syntaxe

```
UPDATE table
SET attr_1 = {expr_1/SELECT…}
…
[WHERE< prédicat>];
```

Modification des données

- Exemple

```
UPDATE inscription
SET adr_étu = ‘ 10 rue de la paix, 69002 Lyon’
WHERE nom_étu = ‘ AUREL Marc’;
```

Suppression de lignes

- On utilise la clause TRUNCATE
- Permet de supprimer toutes lignes d'une table
- Syntaxe
  - Drop storage : blocs mémoire libérés et réaffectés à la base
  - Reuse storage : conserver les blocs mémoire alloués à la base

```
TRUNCATE nom_table
[DROP/REUSE STORAGE];
```

Les index

- C'est un objet optionnel associé à une table. Il est utilisé comme accélérateur dans l'exécution des requêtes
- Il existe deux types de structure pour l'indexation :
  - B-tree (pour mémoire)
  - Bitmap

Index bitmap

- Exemple

Num_étu	Nom_étu
002752	Loulou
002861	Babette
236241	Riri
462732	Fifi

Index bitmap

- Résultat de la table index

	Index bitmap			
Num_étu	Nom_étu = ‘ Loulou’	Nom_étu = ‘ Babette’	Nom_étu = ‘ Riri’	Nom_étu = ‘ Fifi’
002752	1	0	0	0
002861	0	1	0	0
236241	0	0	1	0
462732	0	0	0	1

Pour l’étudiant 236241, la valeur de l’index bitmap est 0010

Création d'index

- Un index est créé automatiquement chaque fois qu'une clé primaire ou une contrainte d'unicité sur une colonne est définie pour une table.
- Dans les autres cas, il faut créer un index explicite

```
CREATE [bitmap] INDEX
ON nom_table;
```

Création d'index

- Exemple

```
CREATE INDEX NomEt
ON Etudiant (nom_étu);
```

- Suppression d'un index

```
DROP INDEX NomEt;
```

Prise en compte des mises à jour

- On utilise les ordres COMMIT er ROLLBACK

  COMMIT;

  ROLLBACK;
- Principe :
  - COMMIT : les mises à jour sont validées et leur effet est effectivement matérialisé dans la base. Les valeurs fournies doivent être compatibles en type et en nombre avec les colonnes à mettre à jour.
  - ROLLBACK : toutes les transactions effectuées depuis le dernier COMMIT sont annulées (la base est à nouveau dans un état cohérent).
  - Remarque : Toute fin de session ou de programme génère toujours un COMMIT.

Les clusters

- Ils correspondent à un mode de rangement des lignes des tables d'une BD.
- Cluster = ensemble de tables rangées ensemble parce qu'elles partagent une ou plusieurs colonnes et qu'elles sont souvent utilisées conjointement dans des opérations de jointure.
- Une table ne peut appartenir qu'à un seul cluster

Les clusters

- Exemple

```
CREATE CLUSTER Inscription
code_diplôme, number (3);
```

Avec lors de la création de la table étudiant :

```
CREATE table Etudiant
(…)
CLUSTER Inscription (code_diplôme);
```

Et

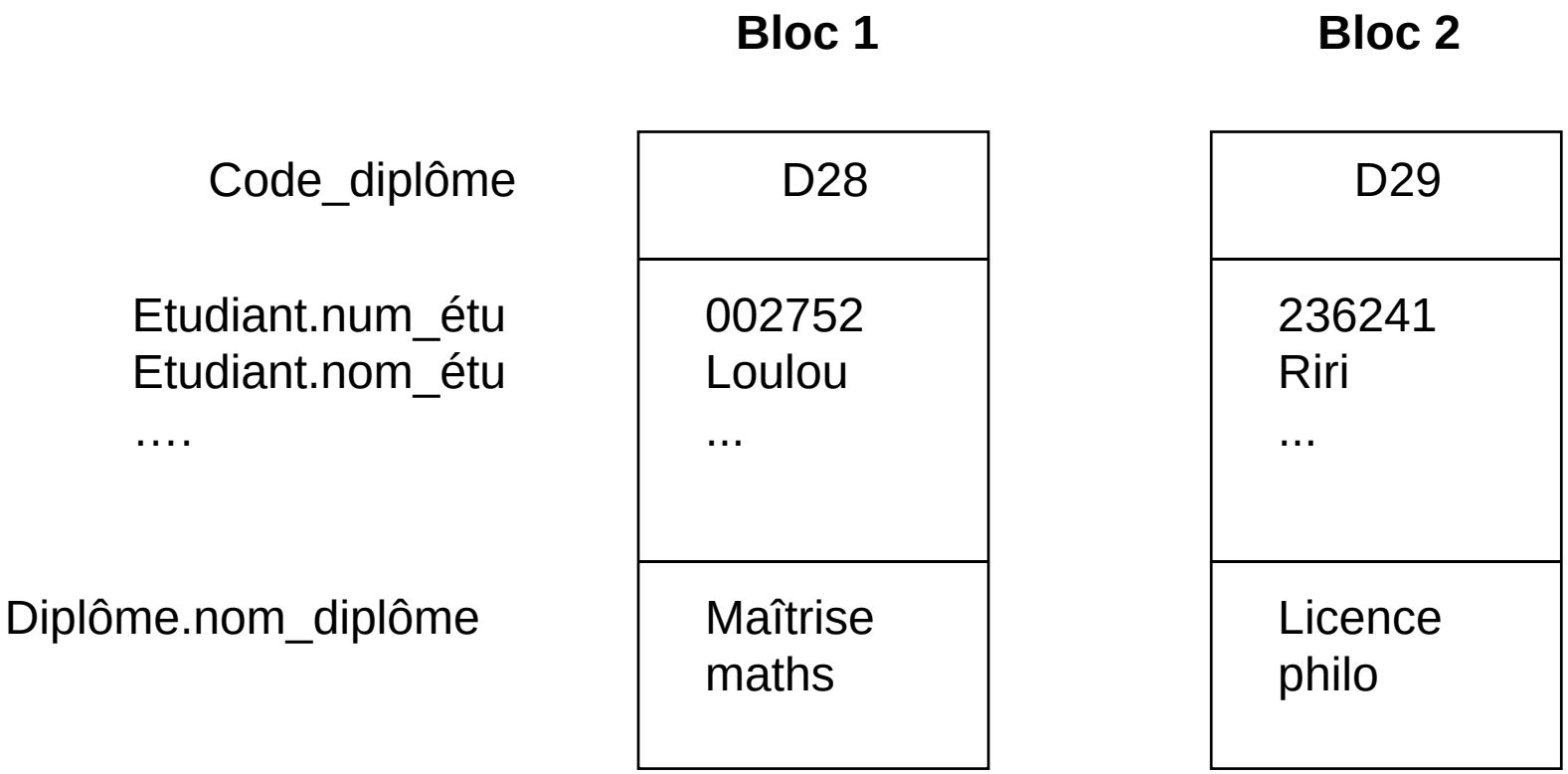
```
CREATE table Diplôme
code_diplôme number (3)
(….)
CLUSTER Inscription (code_diplôme);
```

Les clusters

- On pourrait mettre dans un même cluster les informations sur les diplômes et les étudiants.
- Chaque fois qu'un code_diplôme est créé, on crée dans le cluster un bloc contenant toutes les lignes des 2 relations correspondant à ce code.

Les clusters

- On obtient :

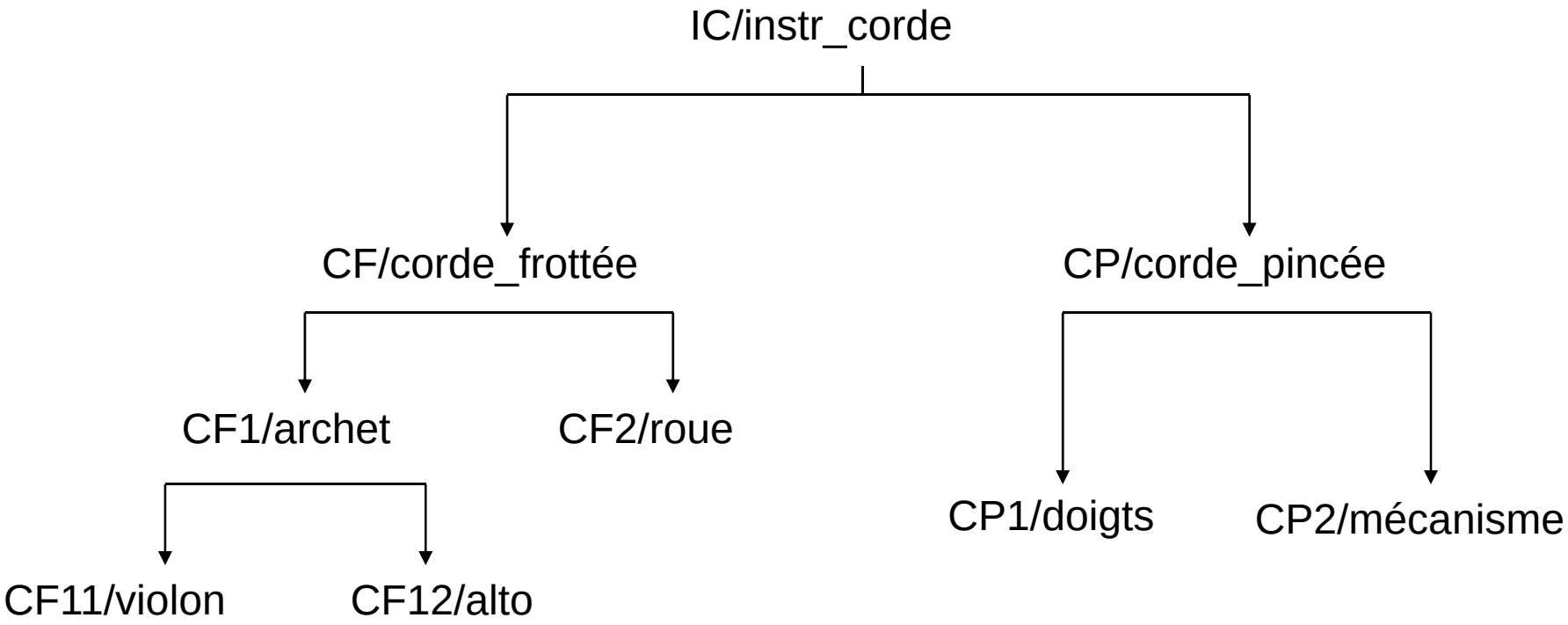


Structures arborescentes

- Il est possible de manipuler des structures de données de type liste ou arbre.
- Pour cela, il faut introduire dans la relation un attribut représentant un lien vers l'élément suivant ou prédécesseur dans la liste ou l'arbre.
- Relation (clé, ….., ….., clé parent)

Les structures arborescentes

Soit la relation Instrument (n° instrument, nom, n° instrument père)
implanté par la relation :
Instrument (num_I, nom_I, num_IP) dont les valeurs sont :



Parcours d'un arbre

- Le lien vers les ascendants se fait par la clé parent.
- On obtient comme résultat d'un ordre SQL les lignes dans l'ordre du parcours de l'arbre
- Pour cela, il faut définir un sens de la liaison et le point de départ dans la structure. Le sens de la liaison se définit par le mot Prior.

Sens de la liaison

- S'effectue par la clause Connect by
  - Syntaxe : connect by condition

    Condition : attr_1 = prior attr_2
    ou prior attr_1 = attr_2

  - Prior définit le sens de parcours de l'arbre
- Point de départ : se définit par la clause Start with ; constitue le point de départ de la recherche
  - Syntaxe : start with prédicat

Exemple 1

```
Select *
from instrument
connect by num_IP = prior num_I
start with nom_I = ‘ corde_frottée’
```

Résultat :

CF	corde_frottée
CF1	archet
CF11	violon
CF12	alto
CF2	roue

Exemple 2

```
Select level,*
from instrument
connect by num_IP = prior num_I
start with nom_I = ‘ corde_frottée’
```

Résultat :

1	CF	corde_frottée
2	CF1	archet
3	CF11	violon
3	CF12	alto
2	CF2	roue

Extension

- On peut utiliser d'autres clauses
  - where : n'empêche pas le parcours de l'arbre si la ligne ne répond pas au prédicat
  - and : le parcours s'arrête dès que le prédicat n'est pas satisfait
- Exclusion : un ordre select écrit avec une clause connect by ne doit pas comporter de jointure

LES RELATIONS DERIVEES

Les relations dérivées

- Lors de l'interrogation de la base, nous avons mis en jeu des opérateurs relationnels conduisant à la création d'une relation résultat.
- Cette relation peut être considérée comme dérivée si:
  - on désire conserver :
    - soit le mode d'obtention
    - soit les résultats sous forme de tables
  - on utilise respectivement :
    - des vues
    - des photographies

Les vues

- Une vue représente une relation virtuelle souvent considérée comme une fenêtre dynamique sur la base.
- Une vue n'a pas d'existence propre. Seule sa description est stockée sous forme d'une requête faisant intervenir des tables (ou des vues).
- Il est possible de manipuler la vue comme une relation ordinaire.

Création d'une vue

- Création de la vue

```
CREATE VIEW nom_view
AS SELECT
FROM
[WHERE] ;
```

- Exemple

```
CREATE VIEW v_enseignant
AS SELECT * FROM enseignant
WHERE grade = ' MCF';
```

Mise-à-jour d'une vue

- Par insert, delete, update si et seulement si :
  - la vue est construite sur une seule table
  - select ne contient pas : group by, connect by, start with
- Exemple

```
UPDATE v_enseignant
SET salaire = salaire*1.1;
```

Contre-exemple

- Considérons les relations :

  Etudiant (num_étu, nom_étu, num_projet)

  Projet (num_projet, professeur responsable)

- Définissons la vue :

  CREATE VIEW affectation
  AS SELECT num_étu, nom_étu, professeur responsable
  FROM Etudiant, Projet
  WHERE Etudiant.num_projet = Projet.num_projet

Contre-exemple

- Effectuons la mise-à-jour suivante :

‘ Aude a pour professeur responsable André au lieu de Jacques’

```
UPDATE VIEW affectation
SET professeur responsable = ‘ André’
where nom_étu = ‘ Aude’
```

- On a alors une incohérence dans la base car, après modification, on peut obtenir André et Jacques comme directeurs du projet numéro 2.

Les photographies

- Ces relations dérivées introduisent la possibilité de conserver le résultat de requêtes à une date donnée. On fige donc un état de la relation dans le temps.
- On utilise la primitive Snapshot.
- Les photographies se créent comme des vues et ont nécessairement une référence temporelle.

Les photographies

- Exemple

```
CREATE SNAPSHOT étudiant_inscrit (96)
AS SELECT*
FROM étudiant
WHERE date = ’15 juil 2007’;
```

- On peut utiliser normalement une photographie.
- La mise-à-jour n’a pas de sens.

Les photographies

- La réplication asynchrone permet de propager des modifications effectuées sur la base à intervalles réguliers sur des bases distantes.

```
CREATE SNAPSHOT nom_cliché
REFRESH
[{FAST/COMPLETE/FORCE}]
[START WITH date]
[NEXTdate];
```

  - FAST : définit un mode de rafraîchissement rapide qui met à jour sur les bases distantes uniquement les lignes modifiées de la base maître.
  - COMPLETE : définit un mode de rafraîchissement complet qui ré-exécute la requête du snapshot.
  - FORCE : effectue un rafraîchissement rapide si possible sinon, complet. C'est l'option par défaut.

INTEGRITE ET CONFIDENTIALITE

Contraintes d'intégrité

- Les contraintes d'intégrité permettent d'assurer la cohérence des données de la base. Le système d'intégrité permet la mise en œuvre de la vérification des contraintes.
- Il existe plusieurs types de contraintes : statiques, dynamiques, simples et ensemblistes.

Contraintes d'intégrité

- Contraintes statiques : constatation d'un état de la base
- Contraintes dynamiques : passage d'un état de la base
- Contraintes simples : portent sur une valeur d'un attribut
- Contraintes ensemblistes : portent sur plusieurs valeurs d'un attribut

Contraintes d'intégrité

Contraintes statiques	Simple	Le salaire d'un employé est supérieur au SMIC
	Ensembliste	La moyenne des salaires du service informatique est supérieur à X €
Contraintes dynamiques	Simple	Le nouveau salaire est supérieur à l'ancien
	Ensembliste	La moyenne des nouveaux salaires du service informatique est inférieur à la moyenne des anciens salaires augmentée de 5%

Contraintes d'intégrité

- Les contraintes sous Oracle sont décrites au niveau du dictionnaire de données.
- Il existe 5 types de contraintes :
  - caractère obligatoire ou facultatif
  - unicité des lignes
  - clé primaire
  - intégrité référentielle ou clé étrangère
  - contraintes des valeurs

Contraintes d'intégrité

- Exemple

```
CREATE table ligne_commande
(num_cli number NOT NULL
, num_art number NOT NULL
, qte_cmd number (2) NOT NULL CHECK(qte_cmd>0)
, qte_livrée number (2)
, CONSTRAINT C1 PRIMARY KEY(num_cli, num_art)
, ….
, CONSTRAINT C4 CHECK (qte_cmd<qte_livrée)
);
```

La confidentialité

- Tout utilisateur peut créer ses propres relations et les manipuler. Il peut autoriser ou non d'autres utilisateurs à lire et/ou à modifier certaines de ses relations.
- L'ordre GRANT permet d'accorder des privilèges à d'autres utilisateurs.

La confidentialité

- Ces privilèges sont :
  - droit de lecture : select
  - droit d'insertion : insert
  - droit de modification : update ou update <nom_attr> (donc peut être limité à certaines colonnes)
  - Exemple :

```
GRANT SELECT, INSERT
ON Etudiant TO André;
```

La confidentialité

- Un utilisateur peut donner un privilège à un autre et, en même temps, lui autoriser à transmettre ce privilège.
- Exemple

```
GRANT SELECT on Etudiant
to André WITH GRANT option identified by mot_de_passe
```

- André pourra autoriser un autre utilisateur à lire la table Etudiant
- Quand tout le monde peut consulter une table, on utilise le mot Public

```
GRANT SELECT ON etudiant TO public
```