CONTRÔLE D'INTEGRITE ET SECURITE DANS LES BASES DE DONNEES

J. Akoka – I. Wattiau

Rôle :

- Faire un système fiable et performant
- Assurer l'intégrité des données

Notions de base

- *Contrainte d'intégrité :*

  assertion qui doit toujours être vérifiée par les données de la base.

- *Base de données cohérente :*

  dont l'ensemble des contraintes d'intégrité est vérifié

Quelques types de contraintes

- contraintes de domaine
- contraintes d'obligation
- dépendances fonctionnelles
- dépendances référentielles ou d'inclusion
- contraintes arithmétiques
- contraintes d'intégrité temporelles

Transaction : unité de traitement séquentiel qui, appliquée à une base de données cohérente, restitue une base de données cohérente

Définition des contraintes d'intégrité

- À l'aide du langage de définition de données (SQL) pour la plupart des contraintes
- A l'aide de triggers dans le cas contraire

Sécurité des données

Objectif : ASSURER LA CONFIDENTIALITE

Principe :

- on identifie les utilisateurs par un nom ou un numéro
- on vérifie l'identité (exemple : mot de passe)
- on définit des autorisations

1er exemple : une matrice d'autorisation

	client	salarié	produit
DUPONT	10	11	11
DURANT	00	10	11
DUBOIS	10	00	11

1er bit : lecture 0 accès interdit

2ème bit : écriture 1 accès autorisé

2ème exemple : des niveaux d'autorisation

Niveau 1 : salarié
Niveau 2 : client
Niveau 3 : produit

DUPONT (PDG) : niveau 1
DURANT (rel. client) : niveau 2
DUBOIS (client) : niveau 3

Niveau du sujet accédant <= niveau de l'objet accédé

Modèle MAC (Mandatory Access Control)

1. Les triggers

Création de triggers

- *Exemple :*
  - ✓ **CREATE TRIGGER nom**
    BEFORE DELETE OR INSERT OR UPDATE ON table
    FOR EACH ROW
    WHEN (new.empno>0)
    DECLARE ..... <<<<déclarations>>>>
    BEGIN
    ..... <<<< bloc d'instructions PL/SQL>>>>
    END;

Le nom du trigger

- **doit être unique dans un même schéma**
- **peut être le nom d'un autre objet (table, vue, procédure) mais à éviter**

Option BEFORE/ AFTER/ INSTEAD OF

- elle précise le moment de l'exécution du trigger
- les triggers AFTER row sont plus efficaces que les BEFORE row parce qu'ils ne nécessitent pas une double lecture des données
- le trigger INSTEAD OF permet par exemple de contourner l'interdiction de mise à jour au travers de vues

Définition du trigger

- *Elle comprend le type d'instruction SQL qui déclenche le trigger :*

  DELETE, INSERT, UPDATE

  On peut en avoir une, deux ou les trois.

- *Pour UPDATE, on peut spécifier une liste de colonnes. Dans ce cas, le trigger ne se déclenchera que si l'instruction UPDATE porte sur l'une au moins des colonnes précisée dans la liste.*

- *S'il n'y a pas de liste, le trigger est déclenché pour toute instruction UPDATE portant sur la table.*

- ***La définition du trigger précise la table associée au trigger :***
  - ✓ **une et une seule table (BEFORE,AFTER)**
  - ✓ **une vue (INSTEAD OF).**

Types de triggers

- **Le type d'un trigger détermine :**
  - ✓ **quand ORACLE déclenche le trigger :**
    - ✓ **BEFORE : avant l'événement déclencheur**
    - ✓ **AFTER : après l'événement déclencheur**
    - ✓ **INSTEAD OF : à la place de l'événement déclencheur**
  - ✓ **combien de fois ORACLE déclenche le trigger.**
- **Le type du trigger est défini par l'utilisation de l'une ou l'autre des options suivantes :**

  BEFORE, AFTER, INSTEAD OF, FOR EACH ROW

ORACLE propose deux types de triggers

- **les triggers lignes qui se déclenchent individuellement pour chaque ligne de la table affectée par le trigger,**
- **les triggers globaux qui sont déclenchés une seule fois.**
- **Si l'option FOR EACH ROW est spécifiée, c'est un trigger ligne, sinon c'est un trigger global.**

- **Pour les triggers lignes, on peut introduire une restriction sur les lignes à l'aide d'une expression logique SQL : *c'est la clause WHEN :***
  - ✓ **Cette expression est évaluée pour chaque ligne affectée par le trigger.**
  - ✓ **Le trigger n'est déclenché sur une ligne que si l'expression WHEN est vérifiée pour cette ligne.**
  - ✓ **L'expression logique ne peut pas contenir une sous-question.**
  - ✓ **Par exemple, WHEN (new.empno>0) empêchera l'exécution du trigger si la nouvelle valeur de EMPNO est 0, négative ou NULL.**

Le corps du trigger est un bloc PL/SQL :

- ✓ Il peut contenir du SQL et du PL/SQL.
- ✓ Il est exécuté si l'instruction de déclenchement se produit et si la clause de restriction WHEN, le cas échéant, est évaluée à vrai.
- ✓ Il est différent pour un trigger ligne et pour un trigger global.

Les noms de corrélation

- ✓ Dans un trigger ligne, on doit pouvoir accéder aux ancienne et nouvelle valeurs de colonne de la ligne.
- ✓ Les noms de corrélation permettent de désigner ces deux valeurs : un nom pour l'ancienne et un pour la nouvelle.
- ✓ Si l'instruction de déclenchement du trigger est INSERT, seule la nouvelle valeur a un sens.
- ✓ Si l'instruction de déclenchement du trigger est DELETE, seule l'ancienne valeur a un sens.

- ✓ **La nouvelle valeur est appelée**
  :new.colonne
- ✓ **L'ancienne valeur est appelée**
  :old.colonne
- ✓ ***Exemple :*** **IF :new.salaire < :old.salaire .....**
- ✓ **Si un trigger ligne BEFORE modifie la nouvelle valeur d'une colonne, un éventuel trigger ligne AFTER déclenché par la même instruction voit le changement effectué par le trigger BEFORE.**

L'option REFERENCING :

✓**Si une table s'appelle NEW ou OLD, on peut utiliser REFERENCING pour éviter l'ambiguïté entre le nom de la table et le nom de corrélation.**

✓***Exemple :***

```
CREATE TRIGGER nomtrigger
BEFORE UPDATE ON new
REFERENCING new AS newnew
FOR EACH ROW
BEGIN
        :newnew.colon1:= TO_CHAR(:newnew.colon2);
END;
```

Les prédicats conditionnels INSERTING, DELETING et UPDATING

- **Quand un trigger comporte plusieurs instructions de déclenchement (par exemple INSERT OR DELETE OR UPDATE), on peut utiliser des prédicats conditionnels (INSERTING, DELETING et UPDATING) pour exécuter des blocs de code spécifiques pour chaque instruction de déclenchement.**

- ***Exemple :***

```
CREATE TRIGGER ...
BEFORE INSERT OR UPDATE ON employe
.......
BEGIN
......
IF INSERTING THEN ..... END IF;
IF UPDATING THEN ..... END IF;
......
END;
```

- **UPDATING peut être suivi d'un nom de colonne :**

```
CREATE TRIGGER ...
BEFORE UPDATE OF salaire, commission ON employe
.......
BEGIN
......
IF UPDATING ('salaire') THEN ..... END IF;
......
END;
```

Nombre de triggers par table

- ✓ **On peut avoir au maximum un trigger de chacun des types suivants pour chaque table :**

  BEFORE UPDATE row
  BEFORE DELETE row
  BEFORE INSERT statement
  BEFORE INSERT row
  BEFORE UPDATE statement
  BEFORE DELETE statement
  AFTER UPDATE row
  AFTER DELETE row
  AFTER INSERT statement
  AFTER INSERT row
  AFTER UPDATE statement
  AFTER DELETE statement.

- ✓ **Même pour UPDATE, on ne peut pas en avoir plusieurs avec des noms de colonnes différents.**

Instructions SQL autorisées

- ✓ les instructions du LMD sont autorisées
- ✓ les instructions du LDD ne sont pas autorisées
- ✓ les instructions de contrôle de transaction (ROLLBACK, COMMIT) ne sont pas autorisées.

Ordre de traitement des lignes

- ✓ On ne peut pas gérer l'ordre des lignes traitées par une instruction SQL.
- ✓ On ne peut donc pas créer un trigger qui dépende de l'ordre dans lequel les lignes sont traitées.

Triggers en cascade

- ✓ Un trigger peut provoquer le déclenchement d'un autre trigger.
- ✓ ORACLE autorise jusqu'à 32 triggers en cascade à un moment donné.

Activation d'un trigger

- ✓ **Un trigger peut être activé ou désactivé.**
- ✓ **S'il est désactivé, ORACLE le stocke mais l'ignore.**
- ✓ **On peut désactiver un trigger si :**
  - ❋ **il référence un objet non disponible**
  - ❋ **on veut charger rapidement un volume de données important ou recharger des données déjà contrôlées.**
- ✓ **Par défaut, un trigger est activé dès sa création.**
- ✓ **Pour désactiver un trigger, on utilise l'instruction**

  ALTER TRIGGER avec l'option DISABLE :

  ALTER TRIGGER nomtrigger DISABLE;
- ✓ **On peut désactiver tous les triggers associés à une table avec la commande :**

  ALTER TABLE nomtable DISABLE ALL TRIGGERS;
- ✓ **A l'inverse on peut réactiver un trigger :**

  ALTER TRIGGER nomtrigger ENABLE;

  ou tous les triggers associés à une table :

  ALTER TABLE nomtable ENABLE ALL TRIGGERS;

Recherche d'information sur les triggers

- ✓Les définitions des triggers sont stockées dans les tables de la métabase, notamment dans les tables USER_TRIGGERS, ALL_TRIGGERS et DBA_TRIGGERS

La procédure

raise_application_error(error_number,error_message)

- ✓error_number doit être un entier compris entre -20000 et -20999
- ✓error_message doit être une chaîne de 500 caractères maximum.
- ✓Quand cette procédure est appelée, elle termine le trigger, défait la transaction (ROLLBACK), renvoie un numéro d'erreur défini par l'utilisateur et un message à l'application.

Gestion d'exceptions

- ✓ **Si une erreur se produit pendant l'exécution d'un trigger, toutes les mises à jour produites par le trigger ainsi que par l'instruction qui l'a déclenché sont défaites.**
- ✓ **On peut introduire des exceptions en provoquant des erreurs.**
  - **Une exception est une erreur générée dans une procédure PL/SQL.**
  - **Elle peut être prédéfinie ou définie par l'utilisateur.**
  - **Un bloc PL/SQL peut contenir un bloc EXCEPTION gérant les différentes erreurs possibles avec des clauses WHEN.**
  - **Une clause WHEN OTHERS THEN ROLLBACK; gère le cas des erreurs non prévues.**

Exceptions prédéfinies - quelques exemples

NO_DATA_FOUND

cette exception est générée quand un SELECT INTO ne retourne pas de lignes

DUP_VAL_ON_INDEX

tentative d'insertion d'une ligne avec une valeur déjà existante pour une colonne à index unique

ZERO_DIVIDE

division par zéro

etc

Utilisation des triggers

- Vérification de l'intégrité
- Répercussion des mises à jour sur données redondantes
- Historisation systématique
- Audit de la base
- Répercussion de mises à jour sur copies de données réparties

Quelques exemples d'utilisation de triggers

VERIFICATION DE L'INTEGRITE

EMPLOYE (NUMEMP,NOM,NUMSERV,NUMCHEF,SALAIRE)
PROJET (NUMPROJ,NOMPROJ,NUMRESPONSABLE)
PARTICIPE (NUMEMP,NUMPROJ)

a) refuser de nommer responsable d'un projet quelqu'un qui n'y participe pas

```
Create or replace trigger t1 before insert or update of numresponsable on projet
For each row
When (new.numresponsable is not null)
Declare noemp integer;
Begin
Noemp:=0;
Select numemp into noemp from participe where numemp=:new.numresponsable and numproj=:new.numproj;
If (noemp=0) then raise_application_error(-20100,'cet employé ne participe pas au projet !') ;
End if ;
End ;
```

VERIFICATION DE L'INTEGRITE

```
Create or replace trigger t2 before delete on participe
For each row
Declare noemp integer;
Begin
Noemp:=0;
Select numresponsable into noemp from projet where numresponsable=:old.numemp
and numproj=:old.numproj;
If (noemp=0) then raise_application_error(-20101,'cet employé est le responsable du
projet !') ;
End if ;
End ;
```

b) au contraire, au moment de l'affectation d'un responsable à un projet, vérifier qu'il y participe, sinon le mettre comme participant aussi.

« FORCAGE » DE L'INTEGRITE

```
Create or replace trigger t3 after insert or update of numresponsable on projet
For each row
When (new.numresponsable is not null)
Declare noemp integer;
Begin
Noemp:=0;
Select numemp into noemp from participe where numemp=:new.numresponsable
and numproj=:new.numproj;
If (noemp=0) then insert into participe values
( :new.numresponsable, :new.numproj,null) ;
End if ;
End ;
```

c) vérifier que la somme des quotas de participation d'un employé aux projets n'excède jamais 100.

VERIFICATION DE L'INTEGRITE

```
Create or replace trigger t4 before insert or update on participe
For each row when (new.quota is not null)
Declare somme integer;
Begin
Somme:=0;
Select sum(quota) into somme from participe where numemp=:new.numemp;
If somme+:new.quota > 100 then raise_application_error(-20200,'attention au surmenage');
End if;
End;
```

On considère les tables relationnelles ci-dessous :
COMMANDE (ncom, montant, datcom)
LIGNECOMMANDE (ncom, nlig, qte, prixunit)
Comment assurer la mise à jour automatique du montant de la commande ?

1ère solution : trigger ligne

REPERCUSSION DE M.A.J.

```
Create or replace trigger t5 after insert or update or delete on lignecommande
For each row
Declare montantligne number; nummaj integer;
Begin
If inserting then montantligne := :new.qte * :new.prixunit; nummaj := :new.ncom; end if;
If deleting then montantligne := - :old.qte * :old.prixunit; nummaj := :old.ncom; end if;
If updating then montantligne := :new.qte * :new.prixunit - :old.qte * :old.prixunit; nummaj :=
:new.ncom; end if;
Update commande set montant=montant+montantligne where ncom=nummaj;
End;
```

2ème solution : trigger global

Create trigger t6 after insert or update or delete on lignecommande

Begin

Update commande

Set montant=(select sum(qte*prixunit) from lignecommande where ncom=commande.ncom);

End;

REPERCUSSION DE M.A.J.

On considère les tables relationnelles ci-dessous :
COMMANDE(ncom, montant, datcom, état)
ANCCOMMANDE(ncom, montant, datcom)

HISTORISATION

On peut écrire un trigger qui, dès que la table des commandes contient plus de 100 commandes retire, s'il y en a, les commandes soldées et les archive dans ANCCOMMANDE.

On considère les deux tables ci-dessous :
TABLE1(clé col1 col2 col3)
TABLE2(clé col1 col2 col3)
On suppose que TABLE2 est la copie de TABLE1 sur un autre site.

REPLICATION

On peut écrire un trigger qui répercute toutes les mises à jour de TABLE1 sur TABLE2.

On considère la table T ci-dessous :
T(clé col1 col2)
et on lui adjoint une table d'audit :
TA(opération datop)
de sorte que toute mise à jour sur T entraîne l'enregistrement de l'opération (INSERT, UPDATE, DELETE) et son heure dans la table T1.

AUDIT

Trigger INSTEAD OF

Oracle offre la possibilité à l'utilisateur de créer un *trigger* (déclencheur) sur la vue.

```
CREATE OR REPLACE TRIGGER TRG_INSTEAD_INSERT_VUE
INSTEAD OF INSERT
ON VW_NOM_VUE
DECLARE
                …
BEGIN
                INSERT INTO table1 …
                INSERT INTO table2 …
END TRG_INSTEAD_INSERT_VUE;
/
```

Un *trigger* `INSTEAD OF` ne peut être défini que sur une vue et peut remplacer les insertions, les mises à jour et/ou les suppressions une vue ne supporte pas les *triggers* `BEFORE` et `AFTER`.

Ce *trigger* est déclaré `INSTEAD OF INSERT`, c'est-à-dire qu'il va s'exécuter lors d'une insertion et va remplacer celle-ci.

Exemple de trigger INSTEAD

```
CREATE OR REPLACE VIEW VW_EMP_CLERK AS
Select empno "Numéro", ename "Nom", deptno "Dept.", sal "Salaire“
From   EMP
Where  JOB = 'CLERK';
Vue créée.
```

```
select * from VW_EMP_CLERK ;
Numéro Nom          Dept.    Salaire
---------- ----------    ----------    ----------

7369      SMITH          20        880
7876     ADAMS          20       1210
7900      JAMES          30       1045
7934      MILLER         10       1430
4 ligne(s) sélectionnée(s).
```

```
Insert into VW_EMP_CLERK
values( 9994, 'Schmoll', 20, 2500 ) ;
Insertion dans la table EMP 1 ligne créée.
Mais cette insertion ne se voit pas au travers de la
vue, car le champ JOB n’est pas renseigné.
```

```
CREATE OR REPLACE TRIGGER TRG_BIR_VW_EMP_CLERK
INSTEAD OF INSERT ON VW_EMP_CLERK
FOR EACH ROW
Begin
Insert into EMP ( empno, ename, deptno, sal, job )
Values (:NEW."Numéro", :NEW."Nom", :NEW."Dept.", :NEW."Salaire", 'CLERK' ) ;
End ;
/
Déclencheur créé.
```

Triggers sur événements système

```
CREATE OR REPLACE TRIGGER logon_trig
AFTER logon ON SCHEMA
BEGIN
INSERT INTO log_trig_table (user_id, log_date, action)
VALUES (user, sysdate, 'début de connexion');
END ;
CREATE OR REPLACE TRIGGER logoff_trig
AFTER logoff ON SCHEMA
BEGIN
INSERT INTO log_trig_table (user_id, log_date, action)
VALUES (user, sysdate, 'fin de connexion');
END ;
```

- Pour chaque évt. système déclenchant, Oracle ouvre un domaine de transaction autonome

Triggers et procédures

```
CREATE TRIGGER test
BEFORE INSERT ON EMP
CALL log_execution;
```

- Un trigger peut appeler une procédure
  - Instruction CALL

Triggers pour la sécurité

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT OR UPDATE OR DELETE ON emp
BEGIN
IF TO_CHAR(sysdate, 'DY' IN ('SAT','SUN'))
THEN RAISE_APPLICATION_ERROR (-20505,
'Modification sur la table EMP impossible le week-end);
END IF;
END;
```

Conseils

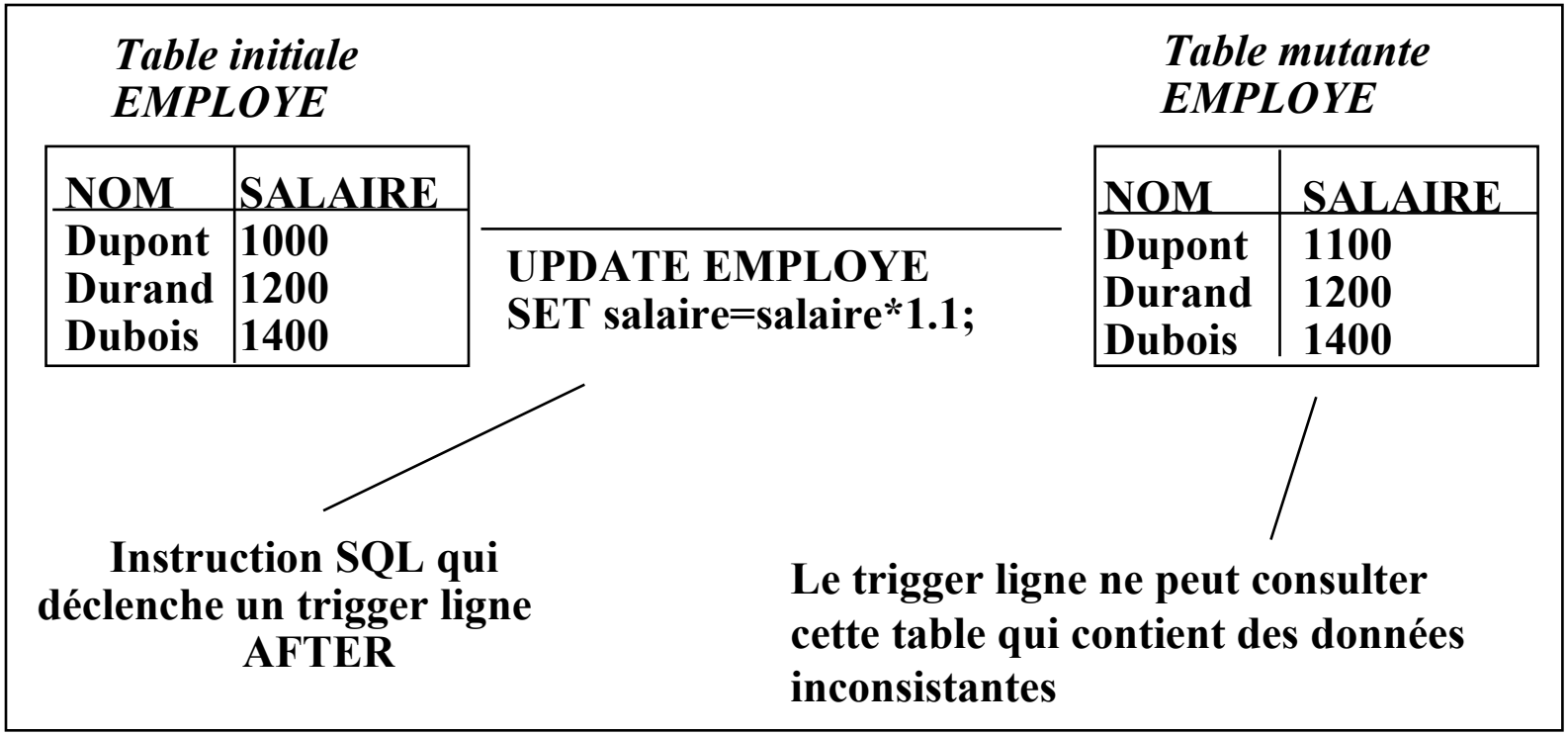
- **Utiliser des triggers pour garantir que, lorsqu'une opération spécifique est effectuée, les actions liées ont aussi réalisées.**
- **N'utiliser les triggers que pour les opérations globales, centralisées qui doivent être déclenchées indifféremment de l'utilisateur ou de l'application.**
- **N'utiliser des triggers que s'il n'y a pas d'autre possibilité intégrée dans Oracle (par exemple des contraintes déclaratives définies avec la table).**
- **Ne pas créer des triggers récursifs :**
  - ✓ ***par exemple, un trigger AFTER UPDATE sur une table qui ferait un UPDATE sur la même table.***
- **Limiter la taille des triggers : à la première utilisation, le trigger est compilé (préférer une procédure stockée, déjà compilée).**

Limites des triggers

- Certaines contraintes complexes ne peuvent être vérifiées au moyen de triggers car le déclenchement du trigger ne peut être différé au COMMIT
- Limite Oracle : la condition du WHEN ne peut faire intervenir de SELECT

Limites des triggers (2)

Oracle : Un trigger ligne ne peut pas lire et/ou modifier la table concernée (appelée table mutante) par l'instruction (INSERT, UPDATE ou DELETE) qui a déclenché ce trigger. *Exemple :*



✓ **On peut utiliser des tables temporaires.**

Solutions au problème de la table mutante

- Si possible, ajouter une donnée calculée (dans une autre table) pour éviter d'accéder à cette table
- Remplacer par un trigger global
- Utiliser une table temporaire
- Utiliser des variables globales
- Utiliser un trigger INSTEAD OF
- L'intercepter sous forme d'EXCEPTION et l'ignorer

2. Le langage PL/ SQL

Introduction

- Extension du langage SQL
- norme SQL3
- permet :
  - l'utilisation d'un sous-ensemble du langage SQL
  - l'introduction de structures procédurales
  - la gestion des erreurs
  - l'optimisation de l'exécution des requêtes

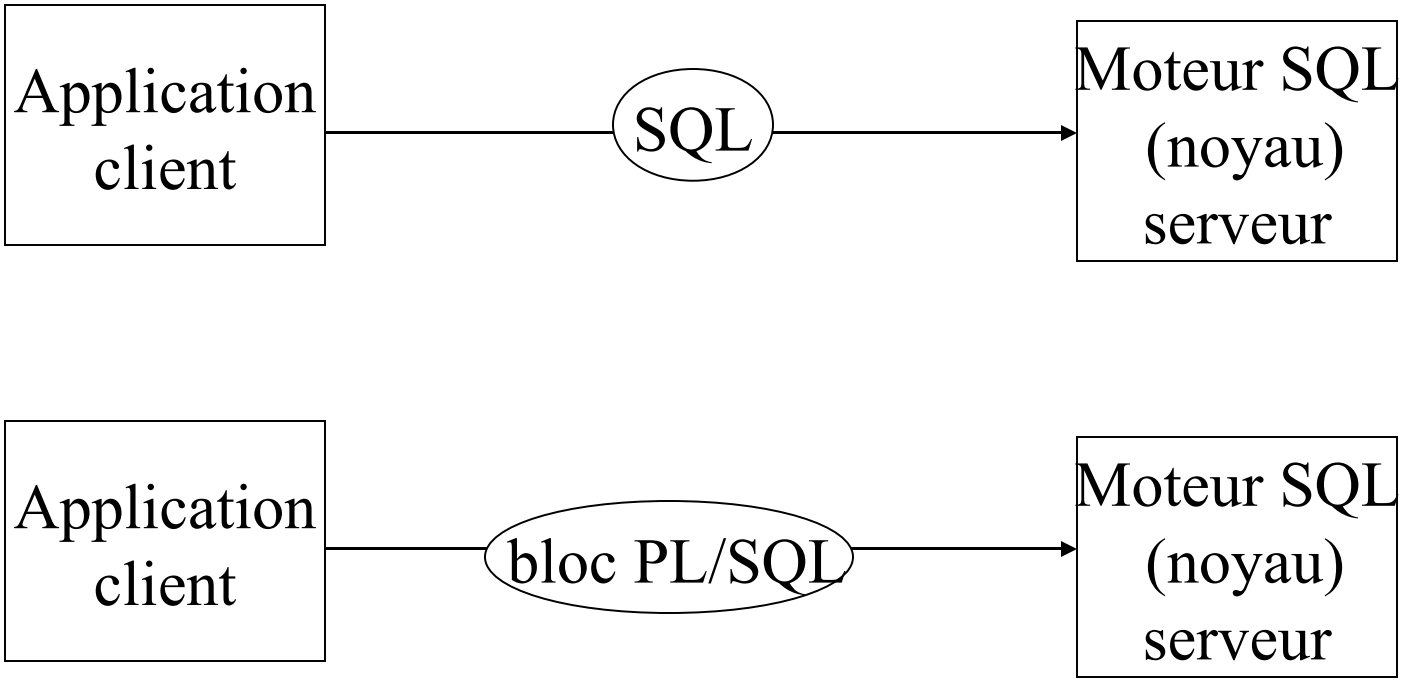
Introduction - suite

- SQL = L4G, déclaratif
- C, COBOL, C++, Java = L3G, procédural
- PL/SQL = déclaratif + procédural
- PL/SQL= SQL +
  - variables et types prédéfinis et définis par l ’utilisateur
  - structures de contrôles IF-THEN-ELSE et boucles
  - procédures et fonctions
  - types d ’objets et méthodes
  - gestion d’erreurs

Exemple

```
DECLARE
vfiliere VARCHAR2(10):=‘ Histoire ’;
vnom VARCHAR2(15) :=‘ Dupont ’;
vprenom VARCHAR2(15) :=‘ Charles ’;
BEGIN
        UPDATE etudiant
        SET filiere=vfiliere
        WHERE nom=vnom AND prenom=vprenom;
        IF SQL%NOTFOUND THEN
                INSERT INTO etudiant (ID, nom, prenom, filiere)
                VALUES (seqetud.NEXTVAL,vnom,vprenom,vfiliere);
        END IF;
END;
```

Principe


Application client
SQL
Moteur SQL (noyau) serveur
Application client
bloc PL/SQL
Moteur SQL (noyau) serveur

Structure de bloc

- Un programme ou une procédure PL/SQL est constitué d'un ou de plusieurs blocs
- Un bloc comporte trois sections :
  - déclaration : description des structures et des variables utilisées dans le bloc
  - corps : instructions du programme + gestion des erreurs
  - traitement des erreurs

Structure de bloc

[DECLARE
déclaration des variables locales au bloc, constantes, exceptions et curseurs.]
BEGIN
    instructions PL/SQL et SQL
    possibilité de blocs imbriqués.
[EXCEPTION
    traitement des erreurs.]
END;

Instructions

- Instructions d ’affectation
- Instructions SQL : CLOSE, COMMIT, DELETE, FETCH, INSERT, LOCK TABLE, OPEN, ROLLBACK, SAVEPOINT, SELECT, SET TRANSACTION, UPDATE
- Instructions de contrôle itératif ou répétitif
- Instructions de gestion des curseurs
- Instructions de gestion des erreurs
- chaque instruction est terminée par ;

Gestion des variables

- 2 types de variables : scalaire et composé
- Types scalaires :
  - CHAR, NUMBER, DATE et VARCHAR2 + BOOLEAN, SMALLINT, BINARY_INTEGER, DECIMAL, FLOAT, INTEGER, REAL, ROWID
  - déclaration : nom_variable type
    - ou bien nomvariable nom_table.nom_colonne%type
  - variable hôte :
    - définie dans l ’environnement extérieur au bloc
    - utilisée dans le bloc
    - champ d ’écran en SQL*FORMS, variable de liaison SQL*PLUS
    - dans un bloc :nom

Gestion des variables

- Types composés
  - RECORD et TABLE
  - RECORD :
    - déclaration : nom_variable nomtable%type
    - ou bien nomvariable nomcurseur%type
    - ou bien énumération des rubriques :

      TYPE nomtype IS RECORD (nomchamp1 typechamp1, nomchamp2 typechamp2, …);

      nomvariable nomtype;
  - TABLE :
    - structure composée d ’éléments d ’un même type scalaire,
    - accessible grâce à un indice de type BINARY_INTEGER (entier signé)

      TYPE nomtype IS TABLE OF typechamp, INDEX BY BINARY INTEGER;

      nomvariable nomtype;

Gestion des variables

- On peut aussi affecter une variable initiale à la déclaration : nomvariable type := valeur;
- On peut aussi définir des constantes par :
  - nomvariable type DEFAULT valeur;
  - ou bien nomvariable CONSTANT type := valeur;
- Visibilité : limitée au bloc et à ses bloc imbriqués, sauf si elle y est renommée
- Conversion de types :
  - explicites avec fonctions TO_DATE, TO_CHAR, TO_NUMBER;
  - implicites par conversion automatique.

Affectation d ’une valeur à une variable

- Opérateur d ’affectation
- Instruction FETCH
- Instruction SELECT … INTO

Opérateur d ’affectation

- Variable de type simple : nom_var := valeur;
- Variable de type composée :
  - RECORD : nomvar.nomchamp
  - TABLE : nomvar(valeur indice)

Valeur résultat d ’une requête

SELECT liste d ’expressions
INTO liste de variables
FROM …

SELECT liste d ’expressions
INTO nom enregistrement
FROM ...

Instructions de contrôle

- Structure alternative - 2 formes

```
IF condition THEN
instructions ;
[ELSE instructions;]
END IF;

IF condition THEN
instructions;
[ELSIF condition THEN
instructions;]
[ELSE
instructions;]
END IF;
```

Instructions de contrôle

- Structure répétitive - 1ère forme

  LOOP

  instructions

  END LOOP;

  peut contenir des LOOP emboîtées

  on en sort par EXIT :

  IF condition THEN EXIT;

  ou bien

  EXIT WHEN condition;

Instructions de contrôle

- Structure répétitive - 2ème forme

  FOR varindice IN [REVERSE] valeurdébut .. valeurfin

  LOOP

    instructions;

  END LOOP

    la variable indice est locale à la boucle, elle ne doit pas être déclarée

    la valeur de début doit être une constante ou une variable déclarée

    même chose pour la valeur de fin

    le pas est 1

    l ’incrémentation est positive ou négative (REVERSE)

Instructions de contrôle

- Structure répétitive - 3ème forme

```
WHILE condition
LOOP
    instructions;
END LOOP;
```

Curseur

- Zone de travail de l'environnement utilisateur qui contient les informations nécessaires à l'exécution d'un ordre SQL
  - texte source de l'ordre SQL
  - forme « traduite » de l 'ordre
  - tampon correspondant à une ligne résultat
  - statut (état du curseur)
  - divers

Curseur implicite

- Géré automatiquement par le noyau
- quand on exécute un ordre SELECT sous SQL*PLUS,
- quand un ordre SELECT ramène une seule ligne résultat dans les autres interfaces
- pour les instructions UPDATE, INSERT et DELETE dans tous les cas

Curseur explicite

- Pour gérer les ordres SELECT renvoyant plusieurs lignes
- 4 étapes :
  - déclaration du curseur
  - ouverture du curseur
  - traitement des lignes
  - fermeture du curseur

Déclaration du curseur

- Un curseur est associé à une requête SELECT
- Il est déclaré dans la section DECLARE

  CURSOR nomcurseur IS requête;

  ou bien

  CURSOR nomcurseur (nomparamètre type …)

  IS requête;

  le paramètre peut être utilisé dans le SELECT, le WHERE ou ORDER BY

Exemple de déclaration

```
DECLARE
CURSOR c1 IS SELECT nom FROM pilote
        WHERE salaire > 1000;
CURSOR c2 (psalaire NUMBER(7,2), pcom NUMBER(7,2) IS
        SELECT ename FROM emp
        WHERE salaire > psalaire AND comm > pcom ;
```

Ouverture du curseur

- OPEN nomcurseur;
- OPEN curseur (paramètres effectifs);
- association par position ou par nom

```
OPEN c1;

OPEN c2(12000, 2500);

OPEN c2(pcom => 2500, psal => 12000);
```

Fermeture du curseur

- CLOSE nomcurseur;
- il est possible d ’utiliser plusieurs fois le même curseur, éventuellement avec des paramètres différents, au cours de l ’exécution d ’un même programme
- il faut alors l ’ouvrir et le fermer à chaque utilisation

Traitement des lignes

- FETCH nomcurseur INTO listevariables;
- FETCH nomcurseur INTO nomenregistrement;
- les lignes obtenues par l ’exécution de la requête SQL sont distribuées une à une par l’exécution d ’un ordre FETCH inclus dans une structure répétitive

Exemple de procédure

```
Declare     -- déclaration du curseur
CURSOR C_EMP ( PN$Num IN EMP.empno%Type )
              IS SELECT empno,ename ,job
              FROM  EMP
              WHERE empno = PN$Num ;
 -- variables d'accueil
LN$Num EMP.empno%Type ; LC$Nom EMP.ename%Type ;        LC$Job EMP.job%Type ;
BEGIN
OPEN C_EMP(7369) ;              -- ouverture du curseur avec passage du paramètre 7369
LOOP
FETCH C_EMP INTO LN$Num, LC$Nom, LC$Job ;     -- Lecture d'une ligne
EXIT WHEN C_EMP%NOTFOUND; -- sortie lorsque le curseur ne ramène plus de ligne
dbms_output.put_line( 'Employé ' || To_char(LN$Num) || ' ' ||LC$Nom ) ;
END LOOP ;
CLOSE C_EMP ; -- fermeture du curseur
OPEN C_EMP( 7521 ) ; -- ouverture du curseur avec passage du paramètre 7521
LOOP
FETCH C_EMP INTO LN$Num, LC$Nom, LC$Job ;
EXIT WHEN C_EMP%NOTFOUND ;
dbms_output.put_line( 'Employé ' || To_char(LN$Num) || ' ' || LC$Nom ) ;
END LOOP ;
CLOSE C_EMP ;
END ;
```

Exemple de fonction

```
CREATE OR REPLACE FUNCTION F_Test_Augmentation
(PN$Numemp IN EMP.empno%Type, PN$Pourcent IN NUMBER)
RETURN NUMBERISLN$Salaire EMP.sal%Type ;
BEGIN
SELECT sal INTO LN$Salaire From EMP WHERE empno = PN$Numemp ;
  -- augmentation virtuelle de l'employé
LN$Salaire := LN$Salaire * PN$Pourcent ;
RETURN ( LN$Salaire ) ; -- retour de la valeur
END;
/
Fonction créée.

Declare LN$Salaire emp.sal%Type ;
Begin
Select sal Into LN$Salaire From EMP Where empno = 7369 ;
dbms_output.put_line( 'Salaire de 7369 avant augmentation ' || To_char( LN$Salaire ) ) ;
dbms_output.put_line( 'Salaire de 7369 après augmentation ' || To_char( F_Test_Augmentation( 7369, 1.1 ) ) ) ;
End ;
/
Salaire de 7369 avant augmentation 880
Salaire de 7369 après augmentation 968
Procédure PL/SQL terminée avec succès.
```

3. Gestion des utilisateurs (Oracle)

Création d'utilisateurs

CREATE USER utilisateur
IDENTIFIED {BY mot de passe | EXTERNALLY }
[DEFAULT TABLESPACE tablespace]
[TEMPORARY TABLESPACE tablespace]
[QUOTA { entier [K,M] | UNLIMITED } ON tablespace…]
[PROFILE profile][PASSWORD EXPIRE]

- Tablespace par défaut : SYSTEM
- Quota en Ko ou en Mo dans ce tablespace pour cet utilisateur
- Sans profil, l'utilisateur reçoit le profil par défaut, qui est, sauf chgt, sans limitation de ressource
- PASSWORD EXPIRE oblige l'utilisateur à changer son mot de passe avant sa prochaine connexion

Exemple

```
CREATE USER biblio
IDENTIFIED BY auteur
DEFAULT TABLESPACE data
TEMPORARY TABLESPACE temp
QUOTA UNLIMITED ON data
QUOTA UNLIMITED ON indx
PASSWORD EXPIRE;
```

Modification d'un utilisateur

ALTER USER utilisateur
IDENTIFIED {BY mot de passe | EXTERNALLY }
[DEFAULT TABLESPACE tablespace]
[TEMPORARY TABLESPACE tablespace]
[QUOTA { entier [K,M] | UNLIMITED } ON tablespace…]
[PROFILE profile]
[DEFAULT ROLE { role,…| ALL [EXCEPT role,…]| NONE}]
[PASSWORD EXPIRE];

- Permet d'activer les rôles qui doivent déjà avoir été affectés

Suppression d'un utilisateur

DROP USER utilisateur [CASCADE];

CASCADE supprime les objets créés par l'utilisateur

Utilisateurs prédéfinis

- Il y a deux super-utilisateurs (administrateurs) prédéfinis : sys et system
- Le rôle prédéfini sysdba (ou dba dans d'anciennes versions) peut être octroyé par les utilisateurs sys et system

Vues du dictionnaire

* Table DBA_USERS qui comporte notamment les colonnes : USERNAME, PROFILE, LOCK_DATE, EXPIRY_DATE, DEFAULT_TABLESPACE, TEMPORARY_TABLESPACE

* Table DBA_TS_QUOTAS contient les quotas alloués aux utilisateurs

Profil utilisateur

- Un profil est un ensemble nommé de limitations de ressources
- Permet de contrôler les ressources utilisés par un utilisateur

CREATE PROFILE profile LIMIT

SESSIONS_PER_USER {entier | DEFAULT | UNLIMITED}

connexions en parallèle par utilisateur

CPU_PER_SESSION {entier | DEFAULT | UNLIMITED} *en centièmes de secondes*

CONNECT_TIME {entier | DEFAULT | UNLIMITED} *en secondes*

IDLE_TIME {entier | DEFAULT | UNLIMITED}

temps max d'inactivité en continu pour une session en secondes

LOGICAL_READS_PER_SESSION {entier | DEFAULT | UNLIMITED}

nb max de blocs de données lus par session

LOGICAL_READS_PER_CALL {entier | DEFAULT | UNLIMITED}

nb max de blocs de données lus par appel au noyau

COMPOSIT_LIMIT {entier | DEFAULT | UNLIMITED}

PRIVATE_SGA {entier [K,M] | DEFAULT | UNLIMITED};

taille allouée à l'espace privé d'une session dans la SGA

Définition de la limite composée

ALTER RESOURCE COST

CONNECT_TIME 3

LOGICAL_READS_PER_SESSION 10;

Définit l'option COMPOSIT_LIMIT à

3 * CONNECT_TIME + 10 * LOGICAL_READS_PER_SESSION

- Quand une limite est atteinte, Oracle fait échouer l'opération en cours, annule la transaction et envoie un code erreur

Gestion des comptes utilisateur

- A partir de la version 8 d'Oracle
- Limitation des paramètres d'utilisation des comptes
- Options visibles dans la table DBA_PROFILE

CREATE PROFILE profile LIMIT

FAILED_LOGIN_ATTEMPTS {entier | DEFAULT | UNLIMITED}

PASSWORD_LOCK_TIME {entier | DEFAULT | UNLIMITED}

nb de jours de blocage d'un compte après un nb d'échecs consécutifs

PASSWORD_GRACE_TIME {entier | DEFAULT | UNLIMITED}

nb de jours donnés pour changer un mot de passe (après avertissement)

PASSWORD_LIFE_TIME {entier | DEFAULT | UNLIMITED}

PASSWORD_REUSE_TIME {entier | DEFAULT | UNLIMITED};

nb de jours avant que le mot de passe courant puisse être réutilisé

Gestion des privilèges

- Privilèges système : concernent la connexion à la base Oracle et permettent à un utilisateur un certain nombre d'actions sur la définition d'objets de la base
- Privilèges objets : permettent des accès aux objets désignés, accordés par le propriétaire de ces objets

Privilèges systèmes

GRANT {privilège système | rôle }

[{privilège système | rôle },…]

TO

{utilisateur | rôle | public} [,{utilisateur | rôle} ,…]

[WITH ADMIN OPTION];

- Avec WITH ADMIN OPTION, les privilèges reçus peuvent être transmis à d'autres utilisateurs ou rôles
- REVOKE privilège FROM {utilisateur|rôle|public};

Privilèges système

Privilèges système	CREATE	ALTER	DROP	Divers
CLUSTER	oui			
ANY CLUSTER	oui	oui	oui	
DATABASE	oui			
DATABASE LINK	oui			
PUBLIC DATABASE LINK	oui		oui	
ANY INDEX	oui	oui	oui	
LIBRARY	oui		oui	
ANY LIBRARY	oui		oui	
PROCEDURE	oui			execute
ANY PROCEDURE	oui	oui	oui	execute

- ANY permet à un utilisateur d'agir sur les objets de tous les utilisateurs

Privilèges système (suite)

Privilèges système	CREATE	ALTER	DROP	Divers
PROFILE	oui	oui	oui	
RESOURCE COST		oui		
ROLE	oui			
ANY ROLE		oui	oui	grant
ROLLBACK SEGMENT	oui	oui	oui	
SESSION	oui	oui		
SEQUENCE	oui			
ANY SEQUENCE	oui	oui	oui	select
SNAPSHOT	oui			
ANY SNAPSHOT	oui	oui	oui	
SYNONYM	oui			
ANY SYNONYM	oui		oui	
PUBLIC SYNONYM	oui		oui	
SYSTEM		oui		
TABLE	oui			
ANY TABLE	oui	oui	oui	lock, comment, select, insert, update, delete
TABLESPACE	oui	oui	oui	manage
TRIGGER	oui			
ANY TRIGGER	oui	oui	oui	
USER	oui	oui	oui	become
VIEW	oui			
ANY VIEW	oui		oui	

Privilèges système (suite)

Privilèges système	Actions permises
ANALYZE ANY	Analyser tout cluster, table, index
AUDIT ANY	Auditer tout objet de la base
AUDIT SYSTEM	Auditer toute commande, privilège de la base
GRANT ANY PRIVILEGE	Accorder tout privilège système
RESTRICTED SESSION	Pouvoir se connecter à une instance de façon restreinte
UNLIMITED TABLESPACE	pas de quota
FORCE TRANSACTION	force sa propre transaction distribuée dans la base locale
FORCE ANY TRANSACTION	force tte transaction distribuée dans la base locale
CREATE ANY DIRECTORY	Créer un alias de répertoire dans la base

Privilèges objets

GRANT {privilège objet [,privilège objet, …] | ALL}

TO {utilisateur | rôle | public} [,{utilisateur | rôle} ,…]

[WITH GRANT OPTION];

Avec WITH GRANT OPTION, les privilèges reçus peuvent être transmis à d'autres utilisateurs

Privilèges : ALTER, DELETE, EXECUTE,
IDNEX, INSERT, REFERENCES,
SELECT, UPDATE

Le groupe PUBLIC

- Tous les utilisateurs
- On peut créer des synonymes publics :

  CREATE PUBLIC SYNONYM nom FOR schéma.nom;

  permet d'utiliser la table nom sans mentionner son schéma

Les privilèges dans la métabase

- Vue DBA_TAB_PRIVS (grantee,table_name,grantor,privilege)
- Vue DBA_COL_PRIVS (grantee,table_name,column_name,privilege)

Concept de rôle

- Ensemble de privilèges système ou privilèges objets sous un nom
- Permet de simplifier la gestion de la sécurité
- Permet de regrouper les utilisateurs sous un nom
- Permet d'associer un ensemble de privilèges système à un type d'utilisateur (développeur, administrateur, …)

Gestion des rôles

- Création :

  CREATE ROLE nomrole {[NOT IDENTIFIED] | [IDENTIFIED BY motdepasse] | [IDENTIFIED EXTERNALLY]};

  Par défaut : pas de mot de passe

- Affectation de privilèges à un rôle:

  GRANT nomprivilege TO nomrole;

- Affectation d'un rôle à un utilisateur :

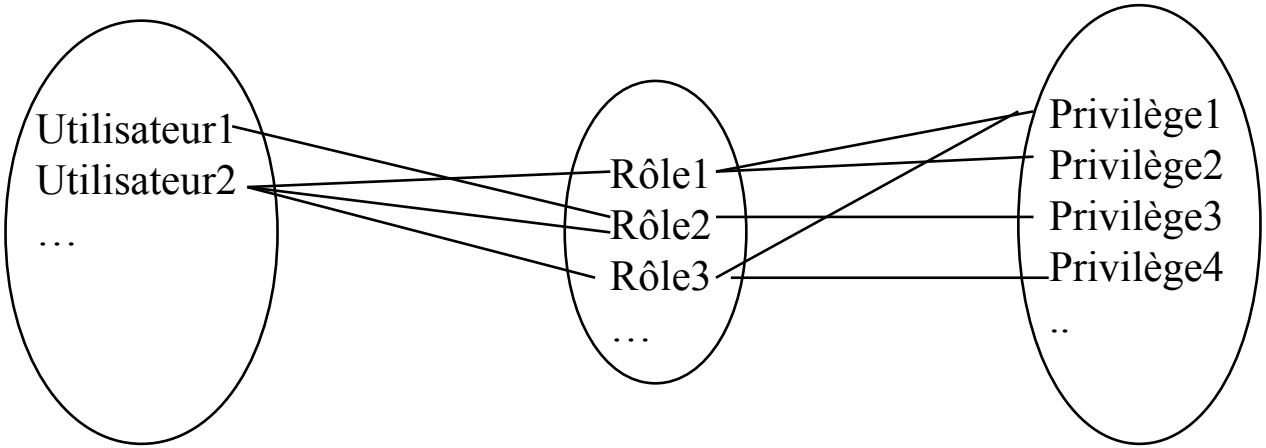
  ALTER USER… DEFAULT ROLE; ou

  GRANT role TO user;

- Activation d'un rôle:

  SET ROLE nomrole; *pour un utilisateur connecté*

Modèle RBAC (Role- Based Access Control)



La vue : un outil de contrôle d'intégrité

- Au moyen des vues, on peut restreindre l'accès:
  - à certaines colonnes d'une table : Création d'une vue par projection
    - Exemple : la table des salariés en omettant les informations personnelles
  - à certaines lignes d'une table : création d'une vue par restriction
    - Exemple : la table des fournisseurs situés à Londres

4. Autres problèmes d’intégrité-confidentialité

Sécurité des données : le contrôle d'inférence

But : autoriser l'interrogation de bases de données statistiques sans divulguer des informations individuelles

➜*Problème de la déduction d'information confidentielle par inférence*

NOM	SEXE	PROFESSION	SALAIRE
alex	m	ingénieur	20 000
andré	m	chauffeur	10 000
charles	m	comptable	12 000
isabelle	f	secrétaire	10 000
jacky	m	comptable	15 000
pierre	m	ingénieur	30 000
raoul	m	comptable	30 000
simone	f	ingénieur	25 000

– Les questions autorisées ne doivent faire intervenir que des sommes ou des cardinalités d'ensembles

– Je pense que simone est la seule femme ingénieur

je veux
1- le vérifier
2- connaître son salaire

Q1 : Combien y a t-il femmes ingénieurs ? ➝ 1

Q2 : Quel est le salaire total des femmes ingénieurs ? ➜ 25 000

Objectif du contrôle d'inférence :

rendre prohibitif le coût d'obtention de telles informations

1ère solution :

CONTROLE SUR LA TAILLE DU RESULTAT

on ne donne pas de réponses portant sur moins d'un certain nombre $\leftarrow$ k d'informations

Exemple :

Q1 : Combien y a t-il d'ingénieurs ?

Q2 : Combien y a t-il d'hommes ingénieurs?

Q3 : Salaire total des ingénieurs

Q4 : Salaire total des ingénieurs de sexe masculin

Autres solutions

- éviter de répondre à des questions dont le résultat recouvre partiellement celui de questions précédentes ➜ coûteux
- partitionner la base de données et n'autoriser que les questions portant sur un groupe en entier ➜ construction des groupes
- distorsion volontaire des résultats :

  on ajoute à la réponse une valeur pseudo-aléatoire qui dépend des données ➜ donner la bonne réponse aux personnes autorisées
- choix aléatoire des échantillons : les questions posées ne sont plus appliquées à toute la base de données mais à un échantillon aléatoire qui échappe à l'usager

Sécurité des données : la cryptographie

- Fonctions de cryptage et de décryptage

Sécurité des données : le brouillage

- Brouiller les données sensibles pour transférer une base à tester: outils DataMasker, Masketeer

Sécurité des données : contrôle des flux

Problème : X n'a pas droit d'accès à R
Y a droit d'accès à R
Y fait une copie R' de R qu'il envoie à X

Solution : ➜ CLASSES DE SECURITE

Le transfert d'information de Y vers X n'est possible que si X a une classe de sécurité au moins aussi privilégiée que Y