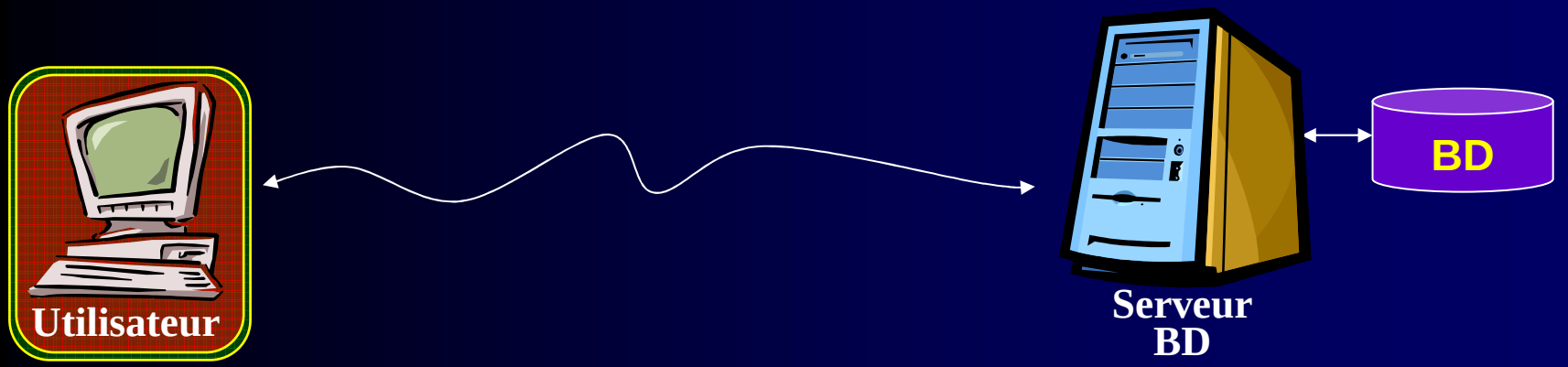
Attaques et Contrôle d'accès pour BD relationnelles

B. Nguyen

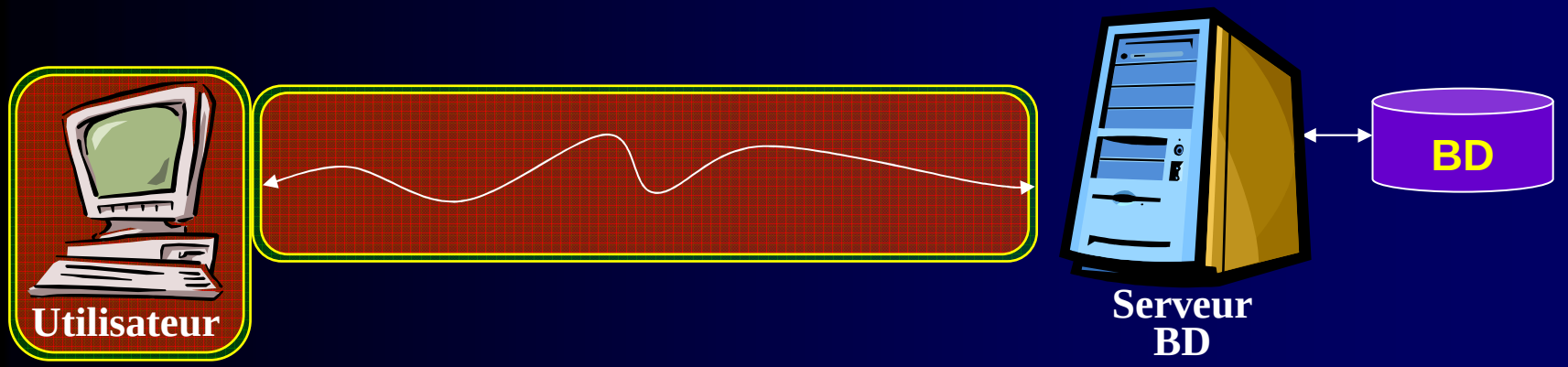
5A A2S

T1 : Identification/authentification



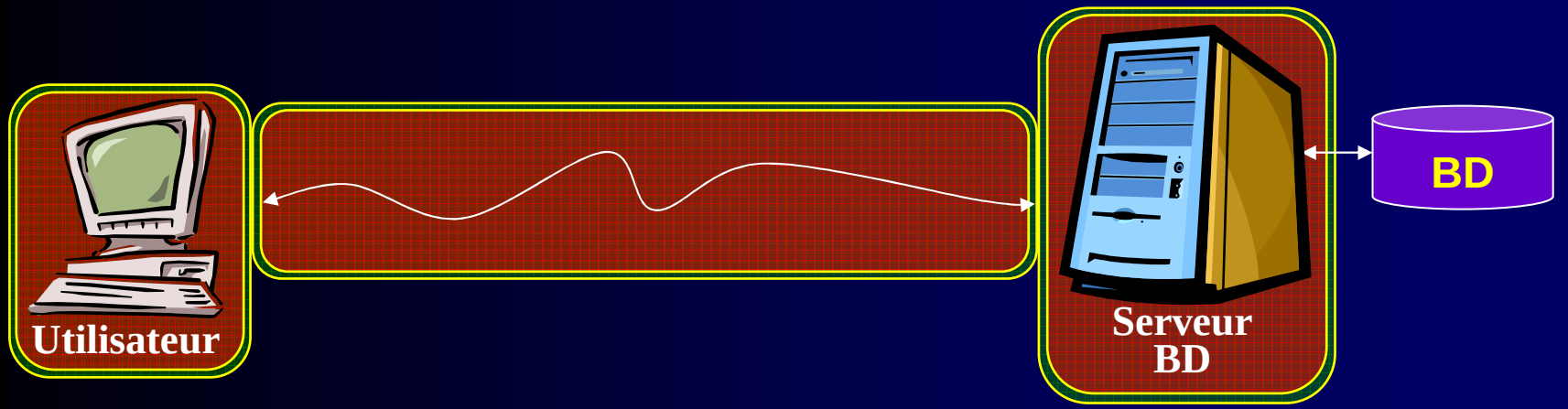
- **Au minimum : login (identification) + password (authentification)**
- **Assuré par le SGBD et/ou l'OS et/ou l'application**
- **Authentification forte :**
  - Conjonction de 2 éléments d'authentification distincts parmi :
    - Ce que l'entité connaît : password, pin code, etc
    - Ce que l'entité détient : carte à puce, token, badge RFID, etc
    - Ce que l'entité est : empreinte biométrique

T2 : Chiffrement des communications (HS)



- **Technologie éprouvée (ex: SSL)**
- **Assure la confidentialité des messages**
- **Techniques cryptographiques complémentaires**
  - Hachage (ex: SHA) : intégrité des messages
  - Signature (ex: via PKI) : authentification et non répudiation du message

T3 : Mécanismes de contrôle d'accès



- **Contrôle d'accès sophistiqué dans les SGBD**
  - Autorisations affectées à des utilisateurs ou rôles
  - Peut porter sur des objets d'une granularité variée : tables, vues, procédures stockées …

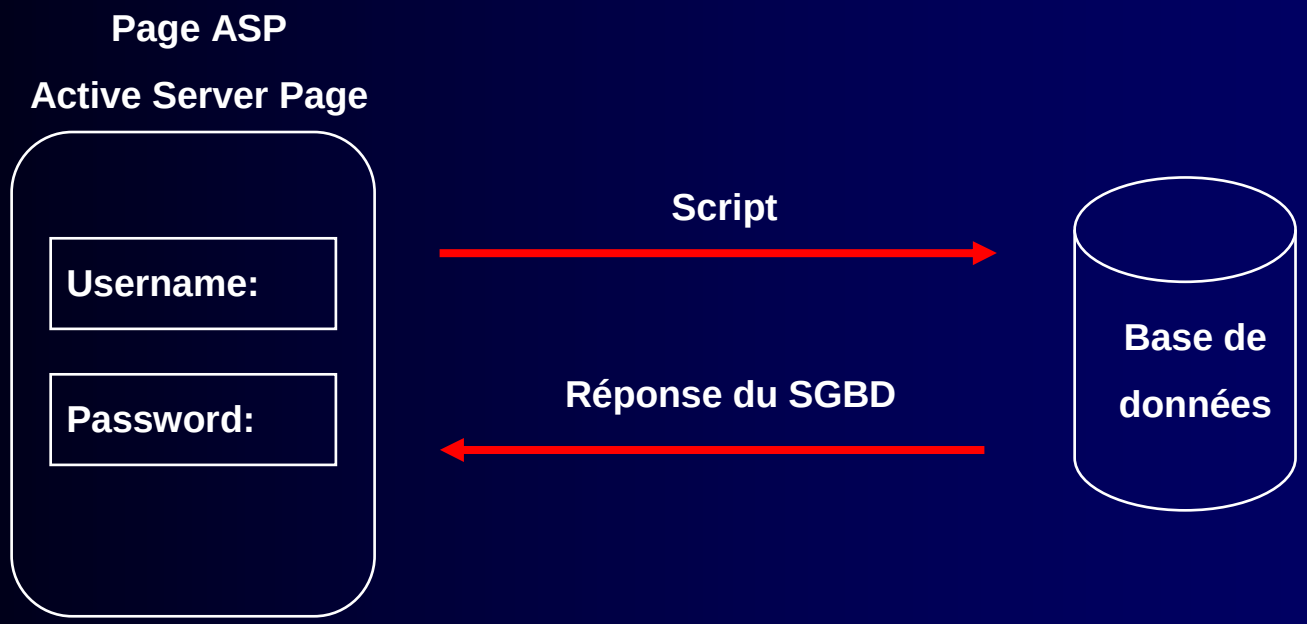
Quelques attaques sur les SGBD

Attaques du mécanisme d'authentification

- **Utilisateurs par défaut d'une base de données**
  - Possibilité d'attaques contre ces comptes s'ils n'ont pas été reconfigurés
  - Exemple : utilisateur SCOTT (mot de passe TIGER) sous ORACLE
- **Gestion des mots de passe**
  - Règles par défaut pour accéder à des comptes pré-créés (ex: Université)
  - Transmission en clair du mot de passe entre le client et le serveur
  - Administrateur malveillant
- **Pour limiter ces risques**
  - Reconfigurer tous les comptes par défaut
  - Chiffrer les communications (ex: Secure Sockets Layer)
  - Gestion de certificats via un tiers de confiance
  - Authentification mutuelle via le protocole Kerberos

Attaque par injection de code SQL

- Attaque contre une base de données via une application proxy



Attaque par injection de code SQL : Connect

- **Exemple de programme PHP exécuté par le serveur :**

```
$login = $_POST['login']; $pass = $_POST['pass'];
$con=mysqli_connect("example.com","ben","abc123","my_db");
$q = "SELECT * FROM Users WHERE login = ' $login' AND password = ' $pass' ";
$result = mysqli_query($con, $q);
$row = mysqli_fetch_array($result); // On fait l'hypothèse d'un seul résultat ou null
$id = $row['id'];
If($id!=null) print " connected with id $id ";
```

- **Si un utilisateur fournit Benjamin comme Username et duratrouver comme mot de passe, la requête suivante est envoyée au SGBD :**

```
SELECT * FROM Users WHERE username = 'benjamin' AND password =
    'duratrouver' ;
```

  - Si la base retourne un n-uplet
    - Connection ok
    - Une information retournée par le tuple permet la connexion e.g. l'attribute ID

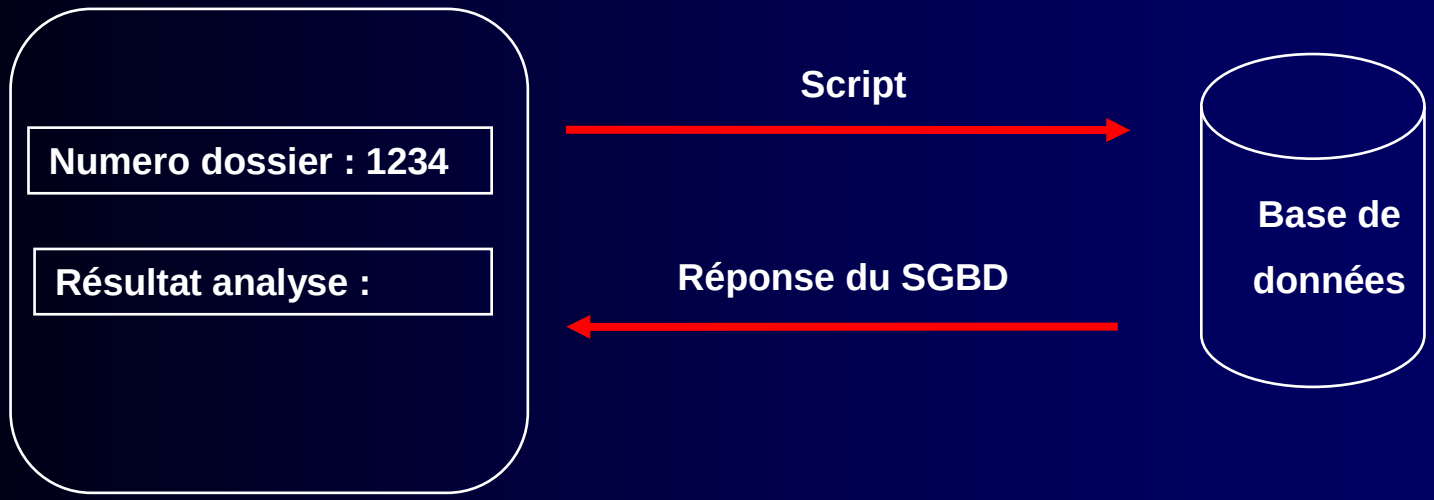
Attaque par injection de code SQL : Connect

- **Réalisation de l'attaque :**
  - L'attaquant fournit le mot de passe suivant :
    - *Toto ' OR '1'='1*
  - Dans ce cas, la requête suivante est envoyée à la base :

    *SELECT * FROM Users WHERE username = 'benjamin' AND password = 'Toto' OR '1'='1 ;*
  - Dans ce cas, le SGBD retourne l'ensemble des n-uplets de la relation Users
    - L'attaquant va se retrouver connecté en tant que premier utilisateur de la base
    - Il s'agira souvent de l'administrateur de la base. Mais sinon, il faudra manipuler la requite (par exemple avec LIMIT 1 OFFSET X)
    - Ne pas oublier de commenter la fin de la requite (avec --)
    - Eventuellement injecter dans un autre champ

Attaque par injection de code SQL : Lecture

- **Autre possibilité d'attaque : lire davantage d'informations**



Attaque par injection de code SQL: Lecture

- **Script exécuté par le serveur :**

```
Dim sql
Sql = "SELECT resultat FROM Dossier_analyse
      WHERE          username = ' $username'
      AND      numero_dossier = ' $numero-dossier ' ; "
```

- **Si Benjamin demande à connaître les résultats de ses analyses correspondant au numéro de dossier 1234, la requête suivante est envoyée au SGBD :**

```
SELECT resultat FROM Dossier_analyse WHERE username = ‘Benjamin'
      AND      numero_dossier = 1234 ;
```

Attaque par injection de code SQL : Lecture

- **Réalisation de l'attaque :**
  - L'attaquant fournit le numéro de dossier suivant :
    - *1234 OR 1=1*
  - La requête suivante est envoyée à la base :

```
SELECT    resultat FROM Dossier_analyse
WHERE     username = 'Benjamin'
AND       numero_dossier = 1234 OR 1=1 ;
```

  - Le SGBD retourne l'ensemble des n-uplets de la relation Dossier_analyse
    - L'attaquant connaît les résultats d'analyse de tous les utilisateurs de la base

Attaque par injection de code SQL : Instructions multiples

- **Globalement**
- **Autre attaque générant une mise à jour :**
  - On aurait tout aussi bien pu saisir le numéro suivant :
    - *1234 ; DELETE FROM Dossier_analyse*
  - La requête envoyée à la base aurait été :

```
SELECT      resultat FROM Dossier_analyse
WHERE       username = 'Benjamin'
AND         numero_dossier = 1234 ; DELETE FROM Dossier_analyse ;
```

  - Et le contenu de toute la table Dossier_analyse est supprimé !

Attaque par injection de code SQL : Requêtes union

- **Réalisation de l'attaque :**
  - L'attaquant fournit le numéro de dossier suivant :
    - *1234 UNION SELECT pass FROM users WHERE ID = 1*
  - La requête suivante est envoyée à la base :

```
SELECT          resultat
FROM Dossier_analyse
WHERE           username = 'Benjamin'
AND             numero_dossier = 1234
UNION
SELECT pass
FROM users
WHERE ID = 1;
```

On retrouvera donc tous les mots de passe dans le résultat de la requête !

Attaque par injection de code SQL : Requêtes union

- On peut en général « unionner » plein de types de requêtes différentes, en particulier, des requêtes qui interrogent les tables système.

Select table_name, column_name

From information_schema.columns

… récupère les noms des tables et des attributs. Utile pour retrouver la table login !

Protections contre l'injection

- **Utiliser des preparedStatement**

```
$stmt = $dbConnection->prepare('SELECT * FROM users WHERE login = ? and password = ?');
$stmt->bind_param('ss', $name, $pass);
$stmt->execute();
```

- **Syntaxe :**

```
bool mysqli_stmt::bind_param ( string $types , mixed &$var1 [, mixed &$... ] )
```

- **Types :**

  s : string, i : int, d : double, b : blob

- **Utiliser des fonctions d'échappement**
- **Chiffrer le contenu des tables et utiliser des MDP forts !**

Remarques

- **Prendre bien garde à protéger toutes les requêtes, y compris les requêtes générées automatiquement pour la gestion du CMS.**
- **La gestion des chaînes de caractères est à risque.**
- **Donner des *droits d'accès* corrects sur les tables, et *ne pas se connecter en root/admin avec les scripts*.**
- **Exercices :**

  http://www.root-me.org/fr/Challenges/Web-Serveur/SQL-injection-authentification

  http://www.root-me.org/fr/Challenges/Web-Serveur/SQL-injection-string

  http://www.root-me.org/fr/Challenges/Web-Serveur/SQL-injection-numerique

Attaque par Buffer Overflow

- **Les conditions de l'attaque**
  - Un langage de programmation permissif (ex: C),
  - Des fonctions laxistes (ex: strcpy(), strcat(), sprintf() …),
  - Un code mal testé,
  - Des variables écrasables par un paramètre d'entrée,
  - Un accès au code permettant de le désassembler
- **Exemple (source Ghost Rider)**

```
main(int argc, char **argv) {
char *somevar;
char *important;
somevar = (char *)malloc(sizeof(char)*4);
important = (char *)malloc(sizeof(char)*14);
strcpy(important, "command"); /*This one is the important variable*/
stcrpy(somevar, argv[1]);
….. Code here ….
}
```

Attaque par Buffer Overflow (suite)

- **Etape 1 : modification du programme pour lui faire imprimer adresses + contenus, avec valeur du paramètre = ‘TOTO’ :**

```
$mon_programme TOTO
    0×8049700: T(0×616c62)
    0×8049701: O(0×616c)
    0×8049702: T(0×61)
    0×8049703: O(0×0)
    0×8049704: (0×0)
    0×8049705: (0×0)
    0×8049706: (0×0)
    0×8049707: (0×0)
    0×8049708: (0×0)
    0×8049709: (0×19000000)
    0×804970a: (0×190000)
    0×804970b: (0×1900)
    0×804970c: (0×19)
    0×804970d: (0×63000000)
    0×804970e: (0×6f630000)
    0×804970f: (0×6d6f6300)
    0×8049710: c (0×6d6d6f63)
    0×8049711: o (0×616d6d6f)
    0×8049712: m (0×6e616d6d)
    0×8049713: m (0×646e616d)
    0×8049714: a (0×646e61)
    0×8049715: n (0×646e)
    0×8049716: d (0×64)
    0×8049717: (0×0)
```

Le décalage d’adresse est déterminé

Attaque par Buffer Overflow (suite)

- **Etape 2: attaque**

$mon_programme TOTO----------newcommand

0×8049700: T(0×646e6573)
0×8049701: O(0×2d646e65)
0×8049702: T(0×2d2d646e)
0×8049703: O(0×2d2d2d64)
0×8049704: - (0×2d2d2d2d)
0×8049705: - (0×2d2d2d2d)
0×8049706: - (0×2d2d2d2d)
0×8049707: - (0×2d2d2d2d)
0×8049708: - (0×2d2d2d2d)
0×8049709: - (0×2d2d2d2d)
0×804970a: - (0×2d2d2d2d)
0×804970b: - (0×2d2d2d2d)
0×804970c: - (0×2d2d2d2d)
0×804970d: - (0×6e2d2d2d)
0×804970e: - (0×656e2d2d)
0×804970f: - (0×77656e2d)
0×8049710: n (0×6377656e)
0×8049711: e (0×6f637765)
0×8049712: w (0×6d6f6377)
0×8049713: c (0×6d6d6f63)
0×8049714: o (0×616d6d6f)
0×8049715: m (0×6e616d6d)
0×8049716: m (0×646e616d)
0×8049717: a (0×646e61)
0×8049718: n (0×646e)
0×8049719: d (0×64)
0×804971a: (0×0)

Attaques contre les bases de données statistiques

- **Exemple :**
  - Relation Analyse(Patient, H/F, Age, Mutuelle, Leucocyte)

Patient	H/F	Age	Mutuelle	Leucocyte
Dupont	H	30	MMA	6000
Durand	F	25	LMDE	3000
Dulac	F	35	MMA	7000
Duval	H	45	IPECA	5500
Dubois	H	55	MGEN	3500
Dumont	H	38	MMA	7500
Dupré	F	32	IPECA	7200
Dupuis	F	50	MGEN	6800
Dufour	H	45	MAAF	4000
Dumas	H	40	Rempart	3800

Attaque contre les bases de données statistiques (suite)

- **Base de données statistique**
  - Base de données qui permet d'évaluer des requêtes qui dérivent des informations d'agrégation
  - Par exemple : des totaux, des moyennes
  - Mais pas des requêtes qui dérivent des informations particulières

- **Exemple :**
  - La requête « quelle est la moyenne du taux de leucocytes des patients ayant plus de 30 ans ? » est permise
  - La requête « quel est le taux de leucocytes de Dupont ? » est interdite

Attaque contre les bases de données statistiques (suite)

- **Exemple d'attaque simple :**
  - *U* veut découvrir le taux de Leucocyte de Dubois
  - *U* sait par ailleurs que Dubois est un adhérent masculin de la MGEN.

- **Requête 1**

```
SELECT COUNT ( Patient )
FROM   Analyse
       WHERE  H/F = 'H'
       AND    Mutuelle = 'MGEN' ;
```

Résultat : 1

- **Requête 2**

```
SELECT SUM ( Leucocyte )
       FROM   Analyse
       WHERE  H/F = 'H'
       AND    Mutuelle = 'MGEN' ;
```

Résultat : 3500

➔ Le système doit refuser de répondre à une requête pour laquelle la cardinalité du résultat est inférieure à une certaine borne *b*

Attaque contre les bases de données statistiques (suite)

- **Requête 3**

  SELECT COUNT (Patient)
  FROM Analyse

  Résultat : 10

- **Requête 4**

  SELECT COUNT (Patient)
  FROM Analyse
  WHERE NOT (H/F = 'H'
  AND Mutuelle = ‘MGEN’) ;

  Résultat: 9

- **Requête 5**

  SELECT SUM (Leucocyte)
  FROM Analyse

  Résultat : 54300

- **Requête 6**

  SELECT SUM (Leucocyte)
  FROM Analyse
  WHERE NOT (H/F = 'H'
  AND Mutuelle = ‘MGEN’) ;

  Résultat : 50800 ; 54300 – 50800 = 3500

Conséquence :

Le système doit aussi refuser de répondre à une requête pour laquelle la cardinalit du résultat est supérieure à $N - b$, où N est la cardinalité de la relation initiale

Attaque contre les bases de données statistiques (suite)

- **Problème :**
  - Mais limiter les requêtes à celles pour lesquelles le résultat a une cardinalité *c* telle que $b \leq c \leq N - b$ n'est pas suffisant pour éviter la compromission
  - Exemple : si $b = 2$, les requêtes auront une réponse si *c* est telle que $2 \leq c \leq 8$

- **Requête 7**

```
SELECT COUNT ( Patient )
        FROM    Analyse
        WHERE   H/F = 'H' ;
```

Résultat : 6

- **Requête 8**

```
SELECT COUNT ( Patient )
        FROM    Analyse
        WHERE H/F = 'H'
        AND     NOT (Mutuelle = 'MGEN') ;
```

Résultat : 5

- **Conséquence**
  - *U* peut déduire qu'il existe exactement un patient masculin qui a la MGEN comme mutuelle,
  - Il s'agit de Dubois, puisque *U* sait que cette description correspond à Dubois

Attaque contre les bases de données statistiques (suite)

- **Conséquence (suite)**
  - Le taux de Leucocyte de Dubois est facilement découvert de la façon suivante :

- **Requête 9**

```
SELECT SUM ( Leucocyte )
        FROM   Analyse
        WHERE  H/F = 'H' ;
```

Résultat : 30300

- **Requête 10**

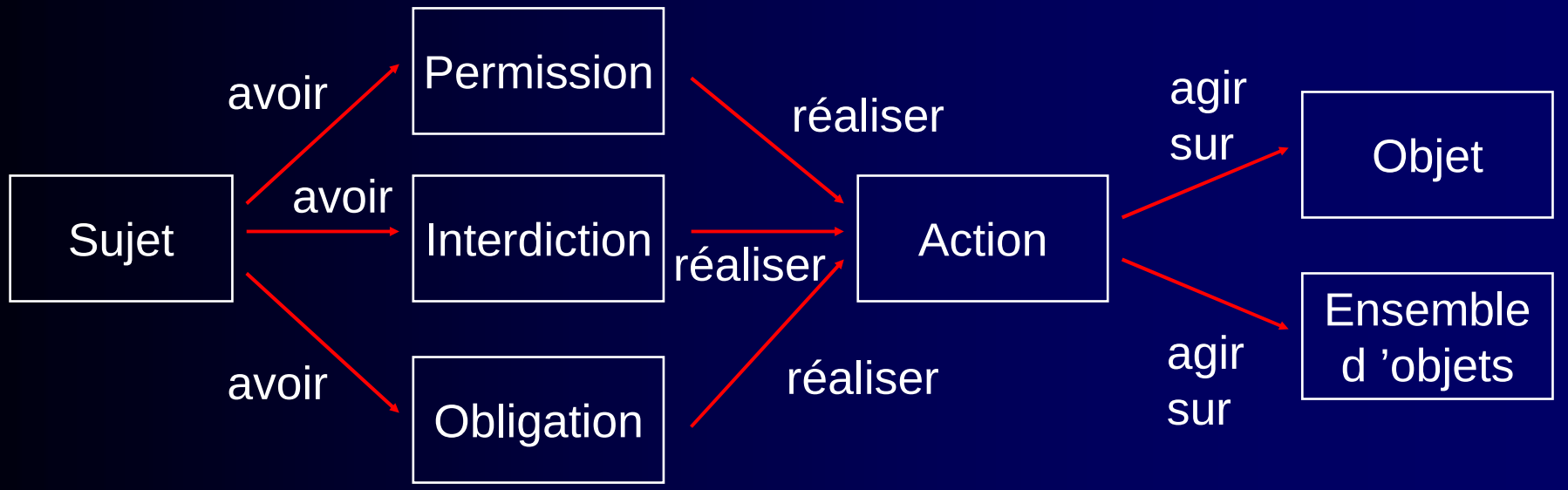
```
SELECT SUM ( Leucocyte )
        FROM   Analyse
        WHERE  H/F = 'H'
        AND    NOT (Mutuelle = ‘MGEN’) ;
```

Résultat : 26800 ; 30300 - 26800 = 3500

Le contrôle d’accès

Politique de contrôle d'accès = ensemble de règles

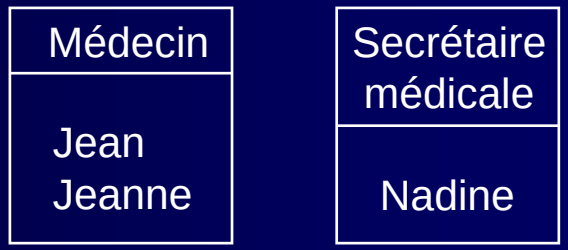
- **Précise qui est autorisé à faire quoi sur quelles données et dans quelles conditions**
- **Format des règles :**



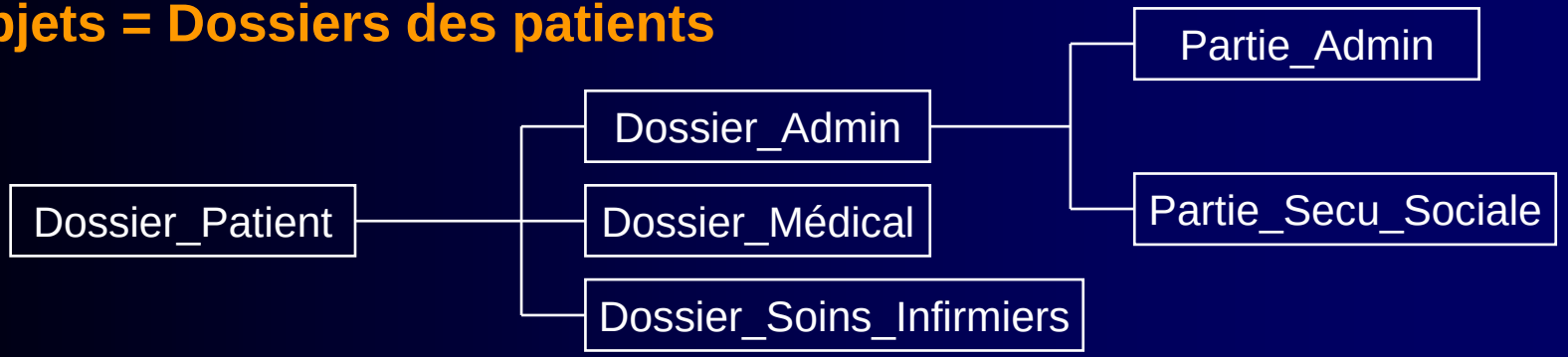
- **Un utilisateur peut être une personne physique ou 'agent' e.g. utilisé par des pages PHP.**

Ex. Système d'information médical

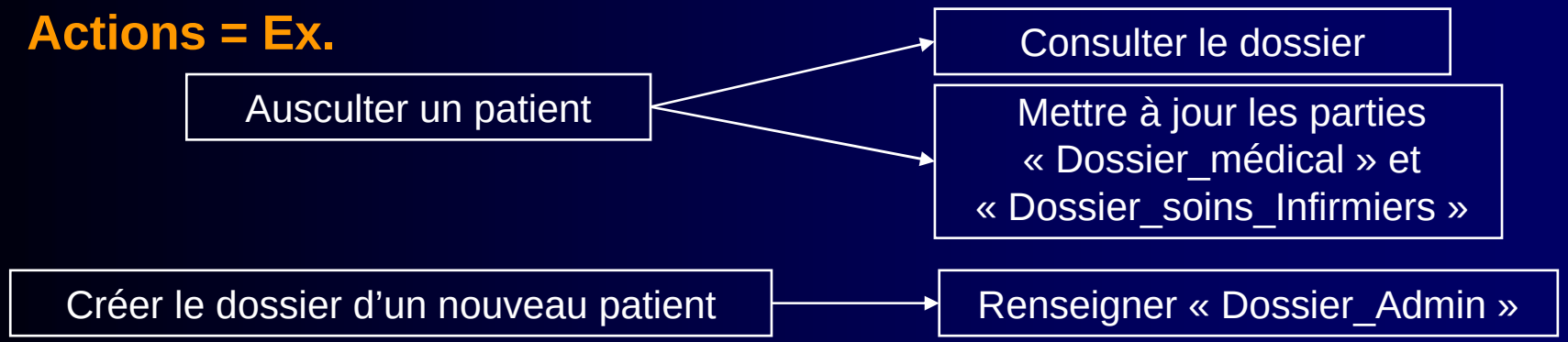
- **Sujets = Personnels du groupe médical**



- **Objets = Dossiers des patients**



- **Actions = Ex.**



Exemple de règles

- **Règles indépendantes du contenu**
  - Les plus simples
  - Règle qui permet d'accéder à un objet indépendamment du contenu de cet objet
  - Exemple R1 : La secrétaire médicale a la permission de gérer le « Dossier_Admin » des patient du groupe médical
    - Permet de consulter et de mettre à jour n'importe quelle information du « Dossier_Admin » de n'importe quel patient du groupe

- **Règles dépendant du contenu**
  - La permission d'accéder à un objet dépend du contenu de cet objet
  - Exemple R2 : Le médecin a la permission de consulter l'intégralité du dossier de ses propres patients
    - Permet de consulter un dossier médical à condition qu'il s'agisse d 'un patient de ce médecin

Exemple de règles (suite)

- **Règles dépendant du contexte**
  - La permission d'accéder à un objet dépend d'une condition associée au contexte d'exécution et indépendante du contenu de cet objet

- **Exemples :**
  - R4 : En l 'absence de la secrétaire médicale, le médecin a le droit de modifier le « dossier_admin » d'un nouveau patient
  - R5 : La secrétaire médicale a accès au « Dossier_Admin » du patient uniquement pendant les heures de travail
  - R6 : La secrétaire médicale a accès au « Dossier_Admin » du patient uniquement à partir d'un poste interne à la clinique
  - R7 : en cas d'urgence, tout membre de l'équipe soignante à accès au dossier du patient
    - Contrôle a posteriori de la réalité de la situation d'urgence

Exemple de règles (suite)

- **Délégation et transfert de droit**
  - Règles liées à l'administration de la politique de contrôle d'accès
- **Exemple :**
  - R8 : Un médecin du groupe médical a la permission d'autoriser la secrétaire médicale à mettre à jour la prescription contenue dans le « Dossier_médical » du patient
- **Contrepartie de la délégation**
  - La secrétaire médicale ayant reçu autorisation a la permission de mettre à jour la prescription du « Dossier_médical » du patient

Modèle discrétionnaire (DAC)

- **DAC = Discretionary Access Control**
  - Contrôle d'accès discrétionnaire
- **Principes de DAC**
  - Le créateur d'un objet fixe la politique de contrôle d'accès sur cet objet
  - Les sujets reçoivent des permissions pour réaliser des actions sur des objets
  - Les sujets ont l'autorisation de transférer certaines permissions à d 'autres sujets
  - Modèle par essence décentralisé
    - ⇗ très souple
    - ⇘ difficile à administrer

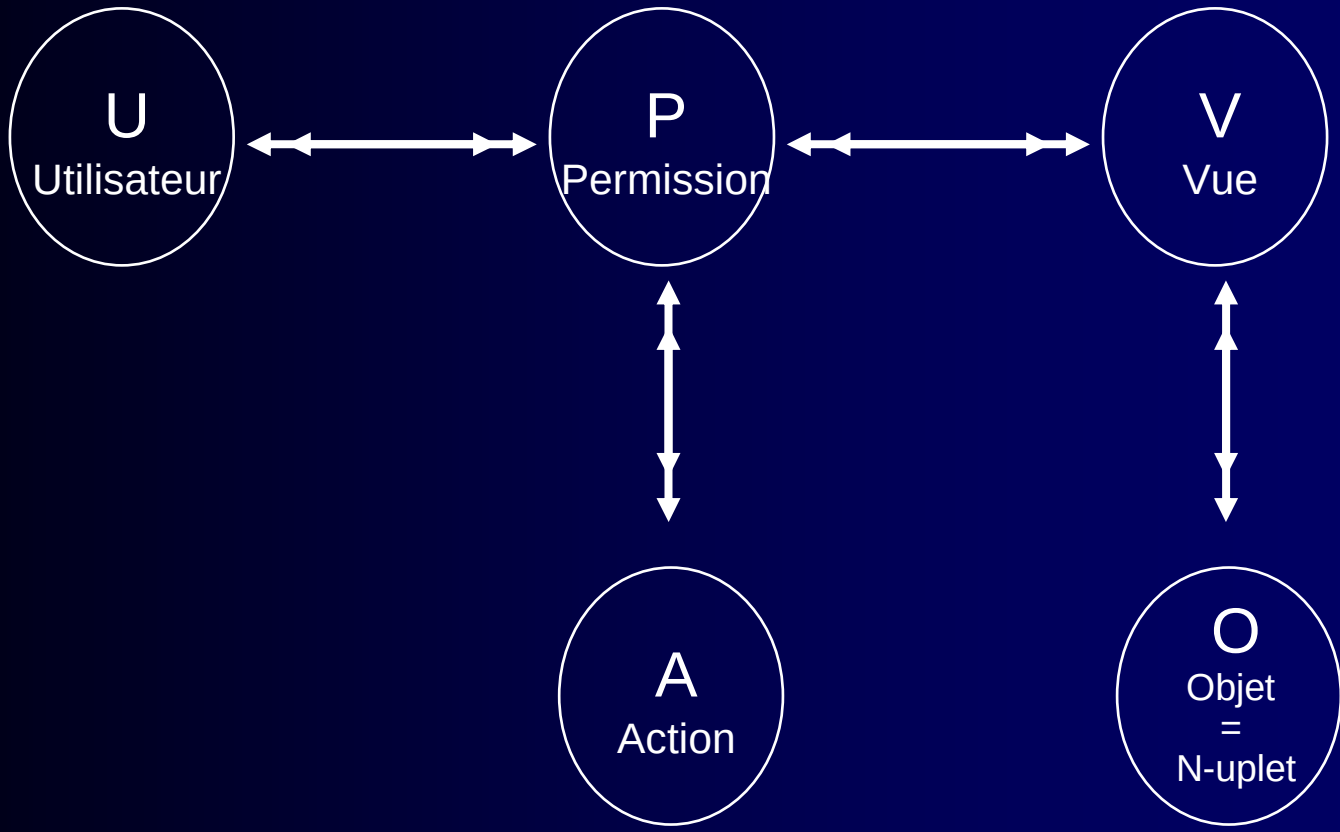
Modèle discrétionnaire (DAC)

- **Exemple de matrice d'accès**
  - La matrice peut être immense

	Nom	Salaire	. . .
Dupont	read write	read	
Robert			
Durand	read		

- **Deux approches pour renseigner la matrice accès**
  - Capacité (Capability List)
    - la matrice est gérée par ligne
    - Une liste d'autorisations, appelée capability list, est affectée à chaque utilisateur
  - ACL(Access Control List)
    - la matrice est gérée par colonne
    - Une liste d'autorisations est affectée à chaque objet

Principe du modèle DAC proposé par SQL



Remarque : modèle fermé, basé exclusivement sur des autorisations

Commandes SQL Grant

GRANT <liste privileges>

ON <table ou vue>

TO <liste utilisateurs>

[WITH GRANT OPTION] ;

– WITH GRANT OPTION
  - est optionnel
  - signifie que l ’utilisateur qui obtient le privilège peut ensuite accorder ce privilège à un autre utilisateur

Privilèges SQL

- **Principaux privilèges**
  - SELECT : permet la consultation de la table
  - INSERT : permet l ’insertion de nouvelles données dans la table
  - UPDATE : permet la mise à jour de n ’importe quelle colonne de la table
  - UPDATE(nom_colonne) : permet la mise à jour d ’une colonne spécifique de la table
  - DELETE : permet de supprimer n ’importe quelle donnée de la table
- **S’applique aussi aux fonctions d’administration**
  - CREATE/ALTER/DROP TABLE : Modifier la définition d’un objet
  - EXECUTE : Compiler et exécuter une procédure utilisée dans un programme
  - REFERENCE : référencer une table dans une contrainte
  - INDEX : Créer un index sur une table
  - …

Commande SQL Revoke

REVOKE [GRANT OPTION FOR] <liste privileges>

ON <table ou vue>

FROM <liste utilisateurs>

[option] ;

- [GRANT OPTION FOR]
  - signifie que seul le droit de transfert est révoqué
- [option] = RESTRICT ou CASCADE
  - Supposons que A accorde le privilège p à B et B accorde ensuite p à C
  - CASCADE : si A révoque p à B alors C perd aussi le privilège
  - RESTRICT : si A révoque p à B alors la révocation échoue

Et si un utilisateur U a reçu le privilège p de A et de B (sans relation entre A et B) ?

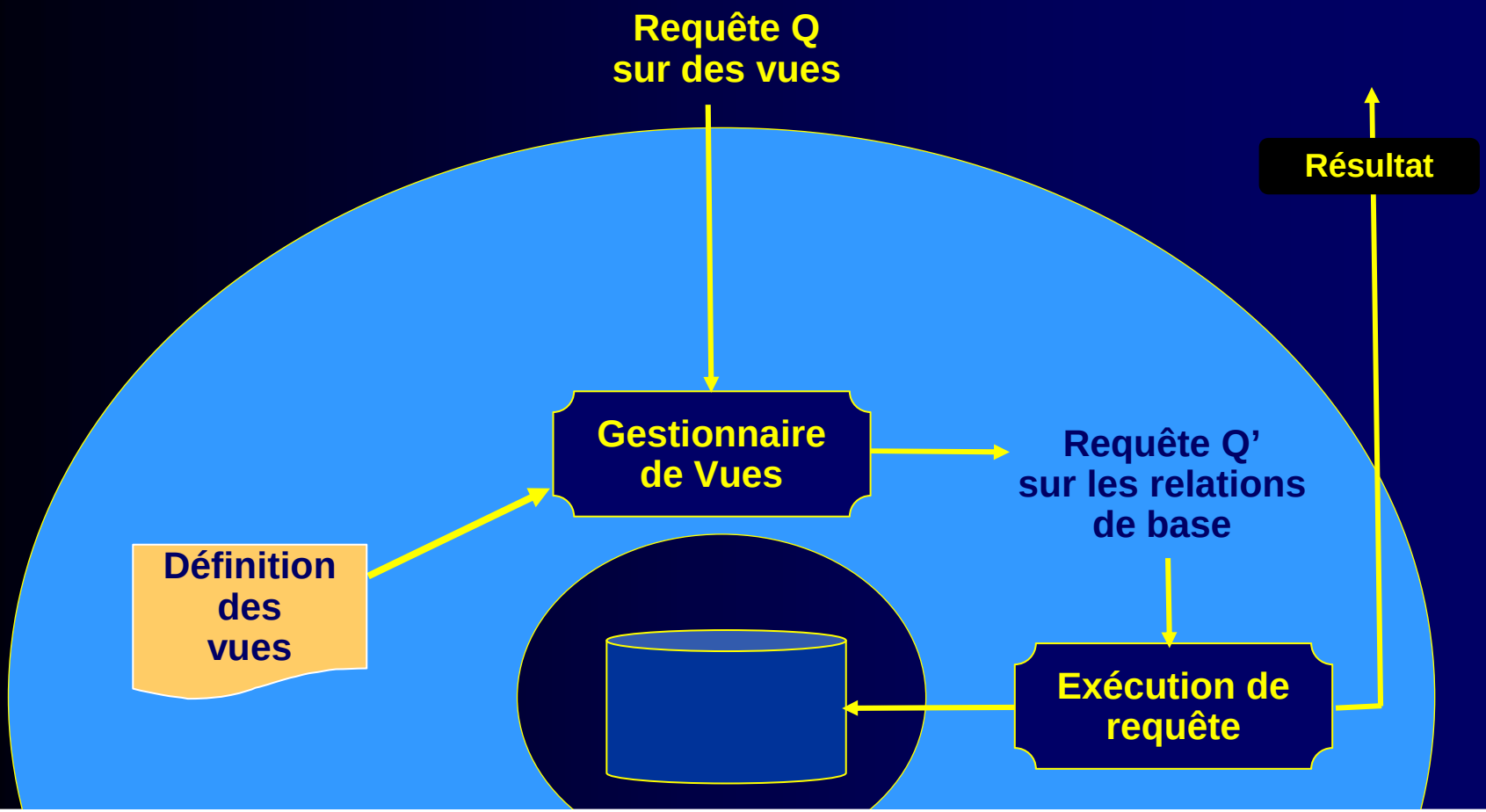
Gestion des vues

- **Les vues permettent d'implémenter l'indépendance logique en créant des** objets virtuels
- **Vue = expression d'un requête SQL**
- **Le SGBD stocke la** définition **et non le résultat**
- **Exemple : la vue du dossier patient**

```
CREATE VIEW dossier_patient AS
SELECT *
FROM   dossier_admin DA, dossier_medical DM, dossier_soins_infirmiers DSI
WHERE       DA.id_patient = DM.id_patient AND
            DA.id_patient = DSI.id_patient
```

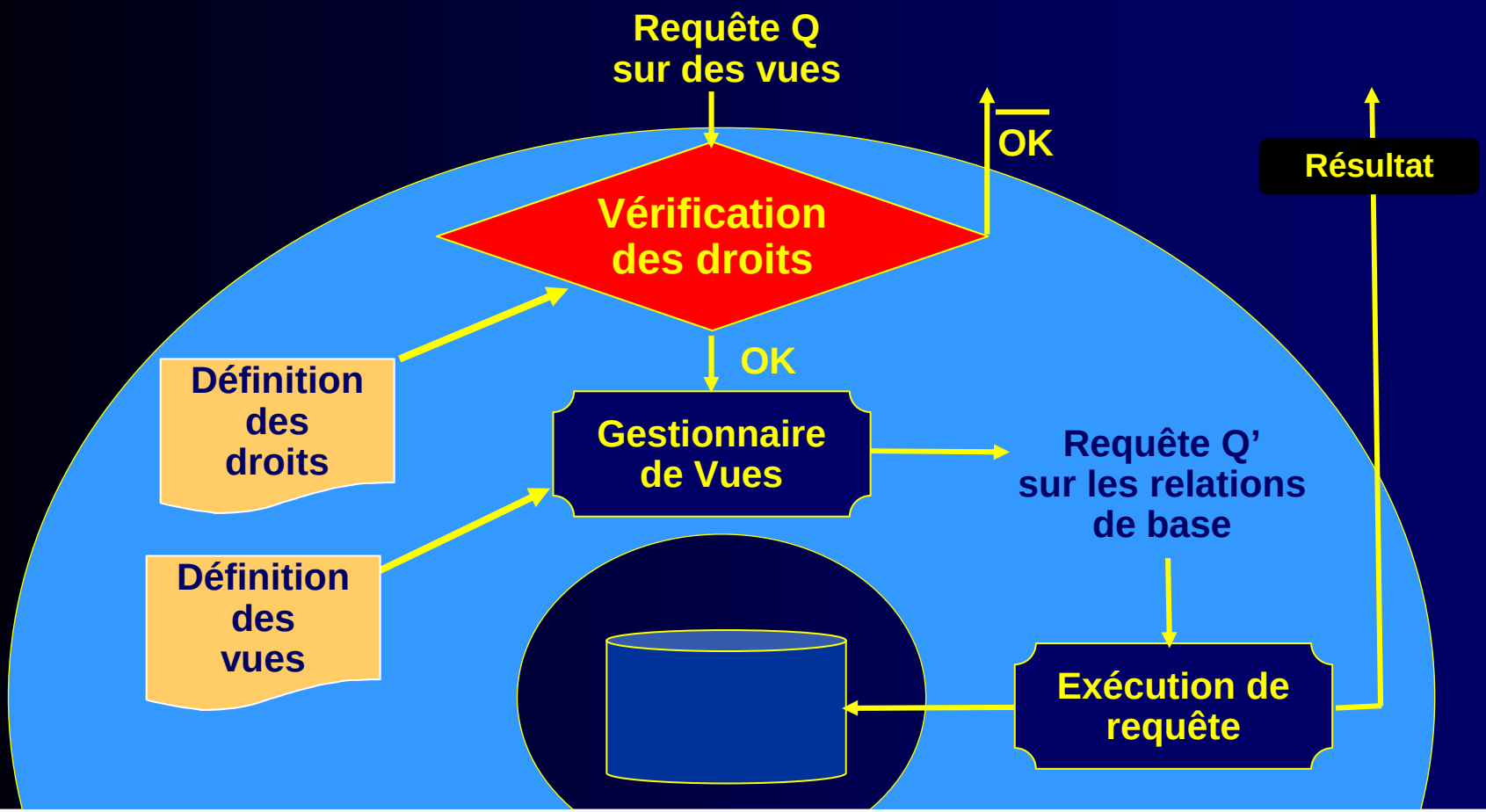
Gestion des vues

Le SGBD transforme **la question sur les vues en question sur les relations de base**

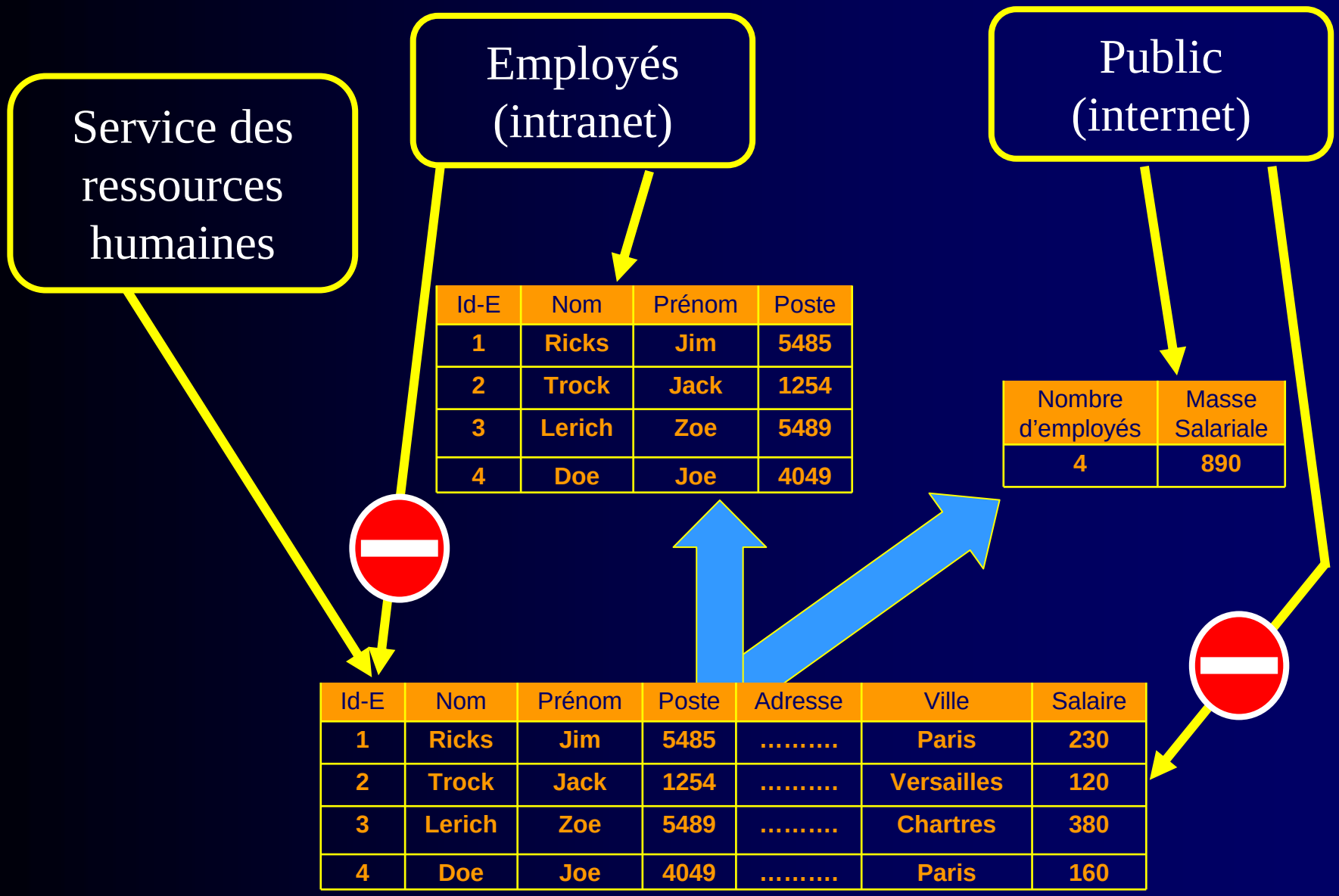


Confidentialité via les vues

Principe : Restreindre l'accès à la BD en distribuant les droits via des vues :



Confidentialité via les vues



Expression des règles (exemples)

- **R1 : La secrétaire médicale a la permission de gérer le « Dossier_Admin » des patients du groupe médical**

```
GRANT ALL PRIVILEGES
    ON dossier_admin
    TO Nadine ;
```

- **R2 : Le médecin a la permission de consulter l'intégralité du dossier de ses propres patients**

```
CREATE VIEW dossier_patient_du_medecin AS
        SELECT *
        FROM dossier_patient
        WHERE dossier_patient.medecin_traitant = CURRENT_USER ;
```

(CURRENT_USER : opérateur prédéfini SQL)

```
GRANT SELECT
    ON dossier_patient_du_medecin
    TO Jean, Jeanne ;
```

Expression des règles (exemples)

- **R4 : En l'absence de la secrétaire médicale, le médecin a la permission de modifier le « dossier_admin » d'un nouveau patient**
  - Deux tables
    - user_status (nom, status) => à créer
    - user_role (nom, role) => existante dans la métabase (dba_roles_privs)
  - Définition d'une vue

```
CREATE VIEW dossier_admin_medecin AS
   SELECT * FROM dossier_admin
   WHERE NOT EXIST (SELECT *
                    FROM    user_status, user_role
                    WHERE   user_status.nom = user_role.nom
                    AND     user_role.role = "secretaire_medicale"
                    AND     user_status.status = "present" ) ;

GRANT UPDATE ON dossier_admin_medecin TO Jean, Jeanne ;
```

  - Puissance d'expression élevée mais définition complexe

Expression des règles (exemples)

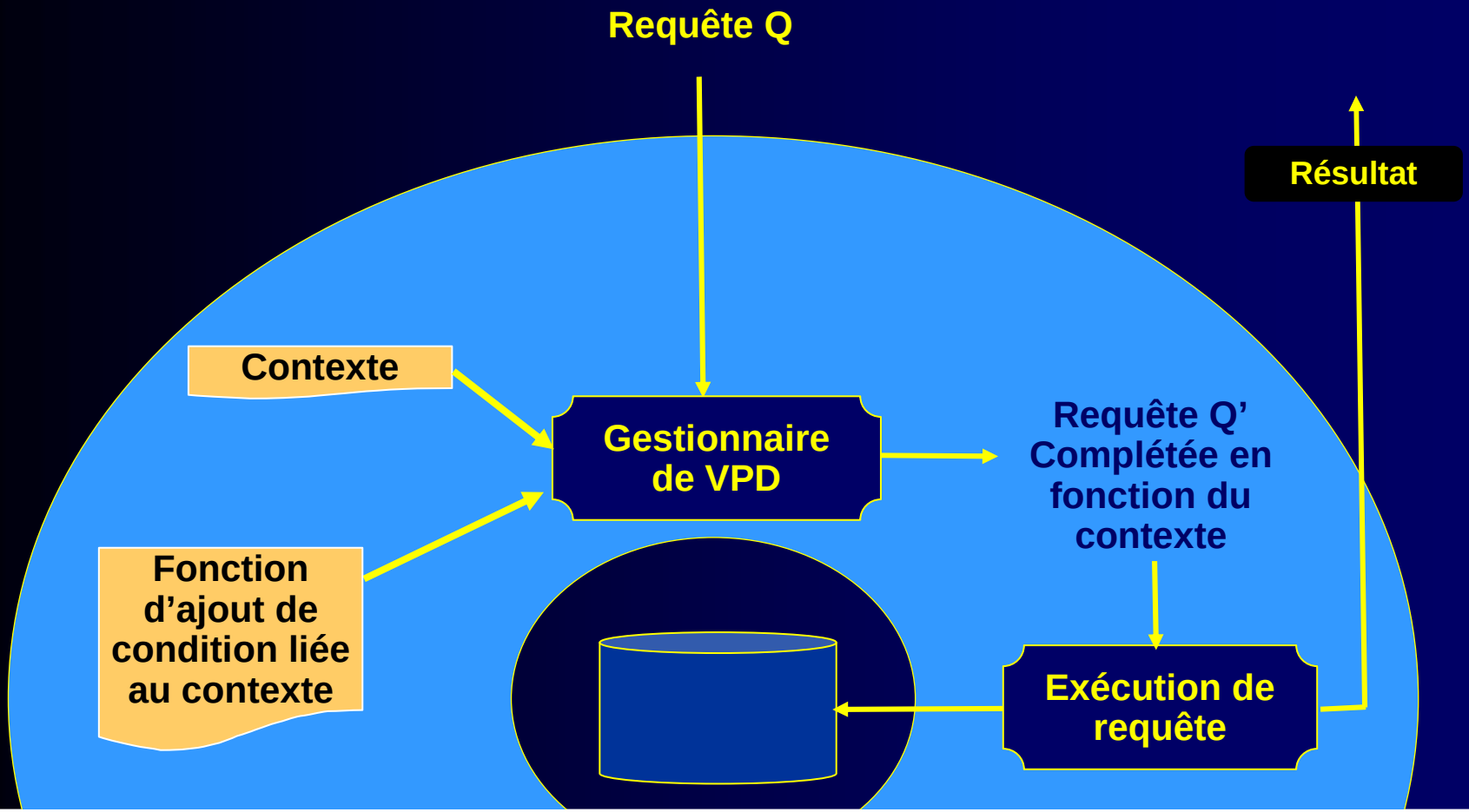
- **R5 : les dossiers administratifs ne sont accessibles que pendant les heures ouvrables**
  - CREATE VIEW dossier_ouvrable
    AS SELECT * FROM dossier_admin
    WHERE TO_CHAR(SYSDATE,'HH') BETWEEN '08' AND '17'

    en dehors de la période 8H – 18h le predicat est faux!
- **R6 : les dossiers administratifs ne sont accessibles qu'à partir des terminaux du secrétariat**
  - CREATE VIEW dossier_ouvrable
    AS SELECT * FROM dossier_admin
    WHERE sys_context('USERENV', 'IP_ADDRESS') IN ('T1', 'T2')

    le prédicat est faux pour tous les postes clients autres que T1 et T2 !

Base de données privée virtuelle (VPD)

- Principe : Rajouter dynamiquement des conditions aux requêtes utilisateur en fonction d'un contexte quelconque (lié à cet utilisateur et/ou à l'application)
- A partir d'Oracle 9i, étendu dans Oracle 10g
- Egalement appelé 'Fine-grained Access Control' (FGAC)



VPD : Contexte d'application

- **Oracle a prévu un contexte par défaut**
  - USERENV : contient des informations système relatives à la session courante (équiv. sys_context)
  - Exemple : CURRENT_USER, HOST, ISDBA …

- **Possibilité de créer un nouveau contexte et d'y associer des attributs**
  - Exemple :
    - create context CTX_SEC_MEDICALE using SCHEMA_MED.SEC_MEDICALE

    - CTX_SEC_MEDICALE sera un contexte associé au package PL/SQL nommé SEC_MEDICALE et stocké dans le schéma SCHEMA_MED

VPD : Définition des règles de sécurité

- **Les règles de sécurité sont écrites en PL/SQL**

```
create package body SEC_MEDICALE as
        function DOSSIER_SEC return varchar2 is
           MY_PREDICATE varchar2(2000) ;
           begin
           MY_PREDICATE := 'id_patient in
              (SELECT id_patient
               FROM dossier_medical
               WHERE medecin_traitant = sys_context('USERENV', 'CURRENT_USER'))' ;
           return MY_PREDICATE ;
        end DOSSIER_SEC ;
end SEC_MEDICALE ;
```

- **Puis associées à un objet**

```
DBMS_RLS.add_policy            // RLS = Row Level Security
        (object_schema      => 'SCHEMA_MED',
            object_name      => 'dossier_medical',
            policy_name      => 'mesdossiers',
            policy_function  => ' DOSSIER_SEC');
        END;
```

Exemple de transformation de requêtes

- **Supposons que Jean formule la requête suivante :**

```
SELECT *
FROM dossier_medical
WHERE id_patient = 'Paul'
```

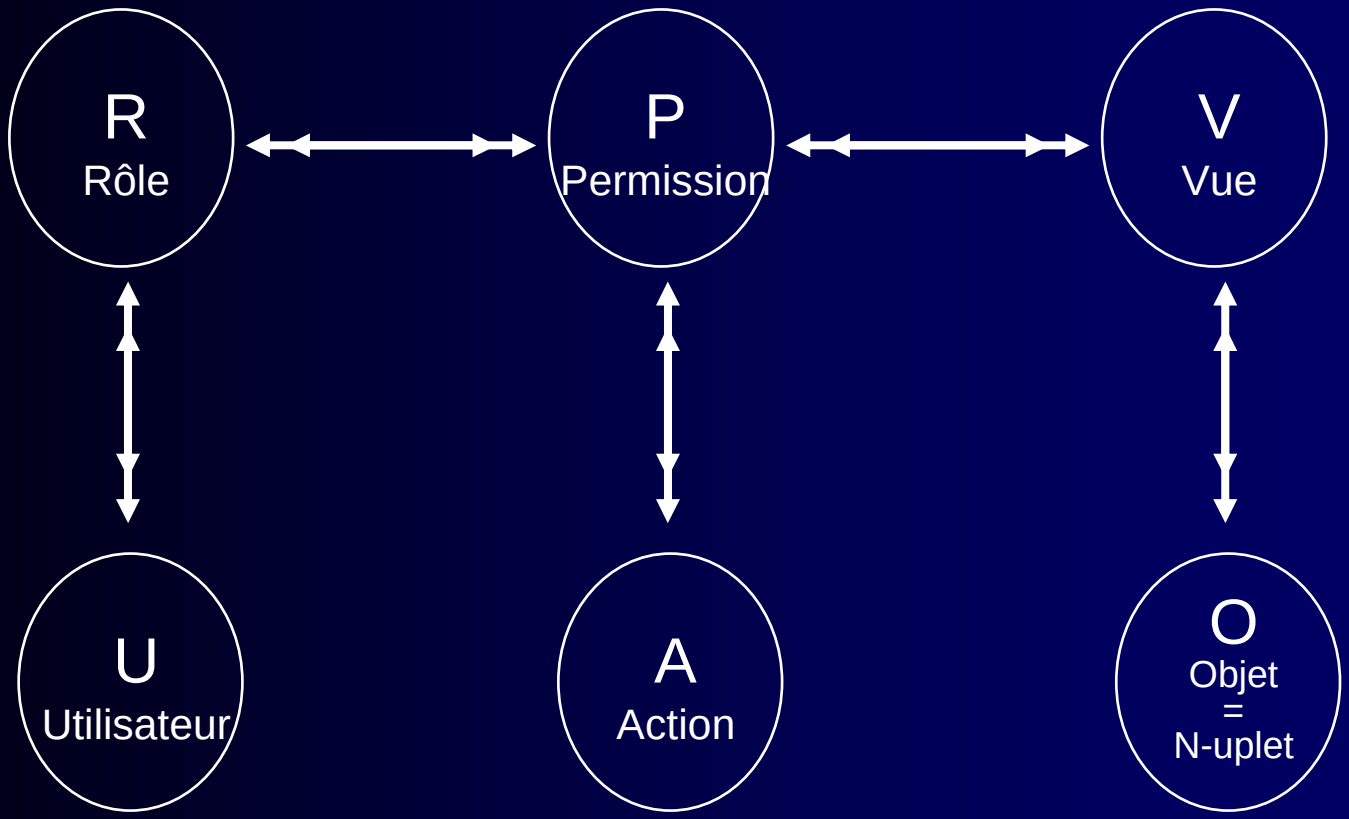
- **L'application de la règle *mesdossiers* va automatiquement transformer cette requête en la requête suivante :**

```
SELECT *
FROM dossier_medical
WHERE id_patient = 'Paul'
AND     id_patient in       (SELECT id_patient
                              FROM dossier_medical
                              WHERE medecin_traitant = 'Jean' ) ;
```

  - Jean ne pourra ainsi accéder qu'aux dossiers médicaux de ses patients

RBAC : Role-Based Access Control

- **Rôle = ensemble de privilèges**
- **Les accès des utilisateurs sont gérés en fonction de leur rôle organisationnel**
- **Objectif = faciliter l'administration des droits**



RBAC : Gestion des rôles dans SQL

- **Le concept de rôle a été introduit dans SQL3 (1999)**

MySQL Q&A :

A.9.5: Does MySQL 5.7 include support for Roles Based Access Control (RBAC)?

Not at this time.

- **Instructions de SQL3**
  - CREATE ROLE <nom_role> ;
    - Création d'un nouveau rôle nom_role
  - DROP ROLE <nom_role> ;
    - Suppression du rôle nom_role
  - SET ROLE <liste_roles> ;
    - Permet à un utilisateur d 'activer un ensemble de rôles pendant la durée d'une session SQL

Adaptation de l'instruction GRANT

- **Affectation des privilèges aux rôles**

  GRANT <liste privileges>
  ON <table ou vue>
  TO <liste roles>
  [WITH GRANT OPTION] ;

- **Affectation des rôles aux utilisateurs**

  GRANT <liste roles>
  TO <liste utilisateurs>

- **Rôle junior et rôle senior**

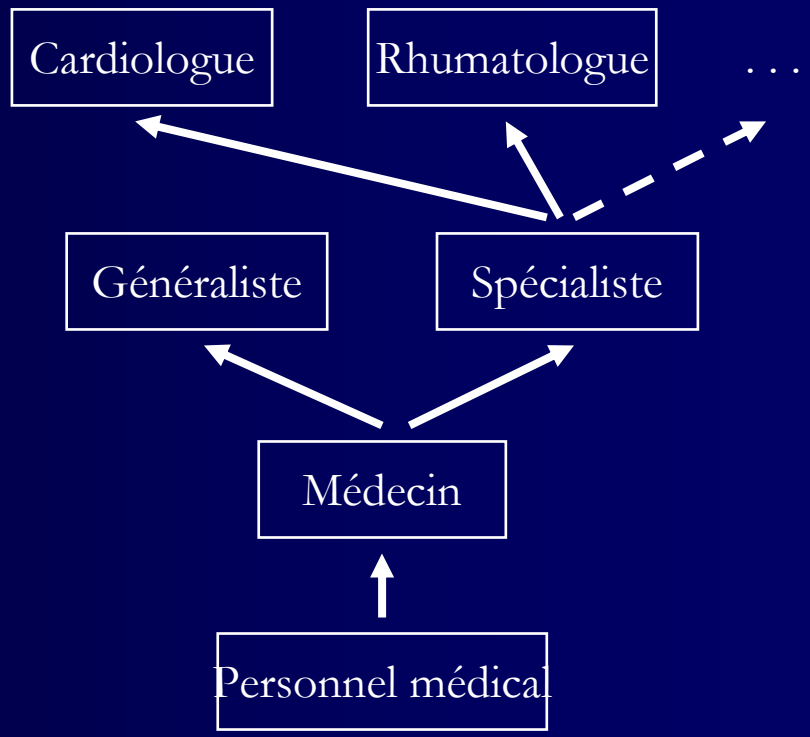
  GRANT <role1> TO <role2>

  Le rôle role2 reçoit tous les privilèges du rôle role1

Hiérarchie de rôles (1)

- **Spécialisation/Généralisation**
  - R1 est un rôle senior de R2

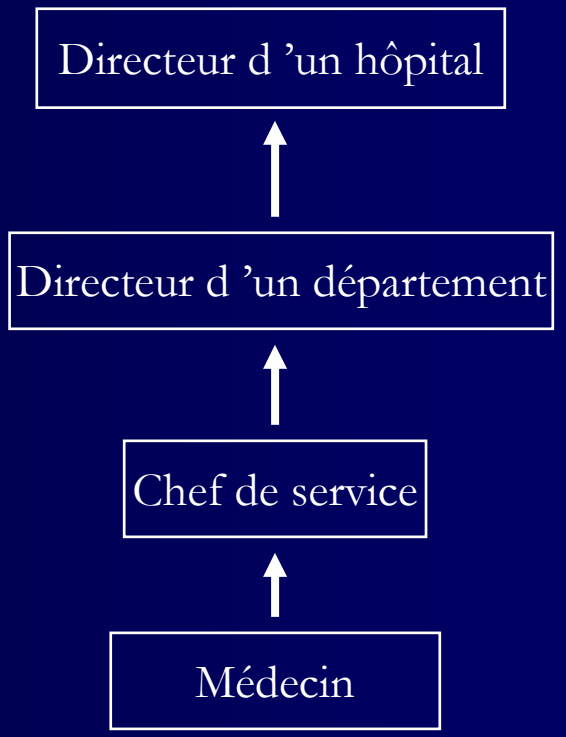
    si chaque fois qu'un utilisateur joue le rôle R1, cet utilisateur joue aussi le rôle R2
  - Les feuilles de la hiérarchie ont plus de privilèges que la racine



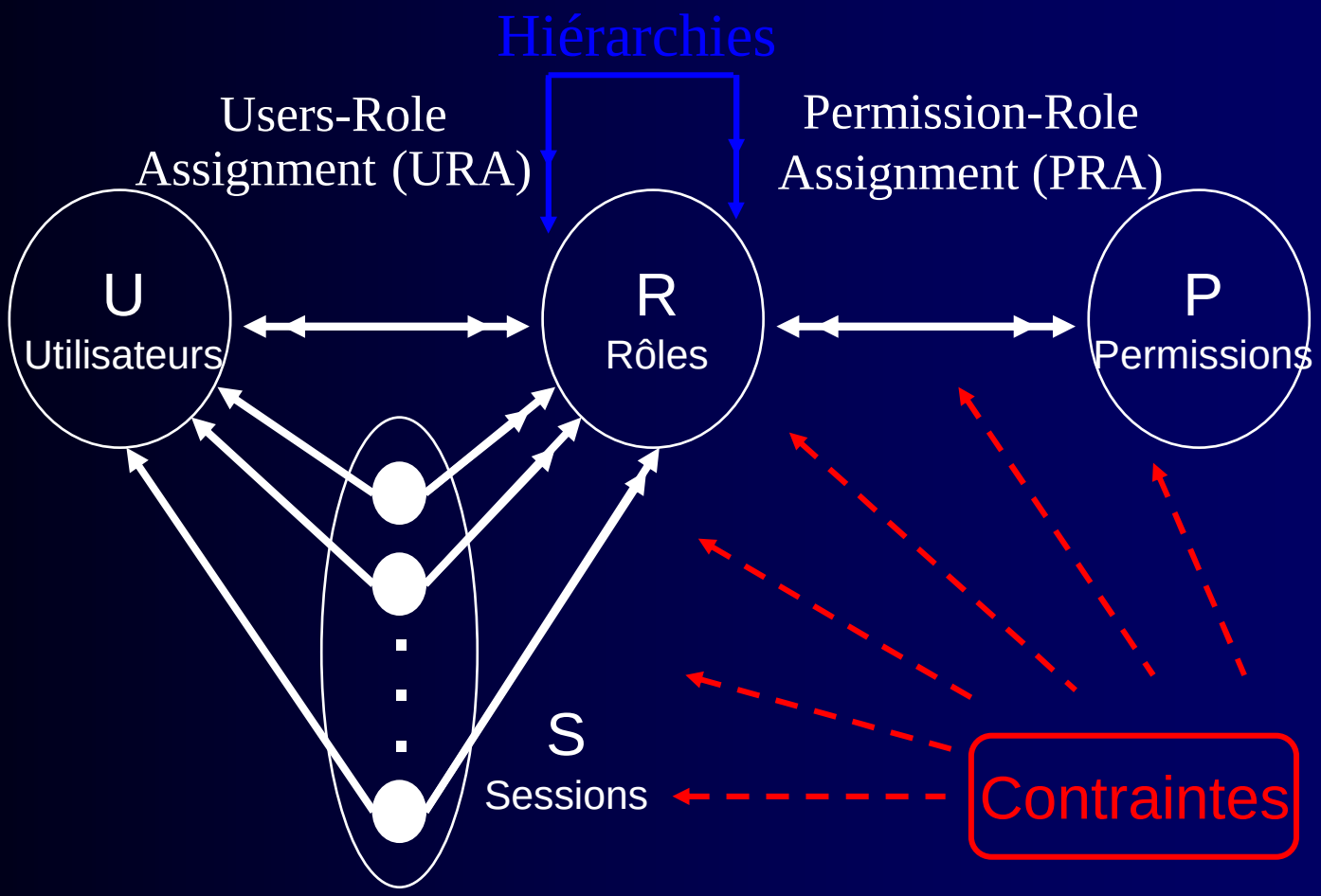
Hiérarchie de rôles (2)

- **Hiérarchie organisationnelle**
  - R1 est un rôle senior de R2

    si un utilisateur jouant le rôle R1 est un supérieur hiérarchique d'un utilisateur jouant le rôle R2
  - Les feuilles de la hiérarchie ont plus de privilèges que la racine



Le modèle RBAC complet

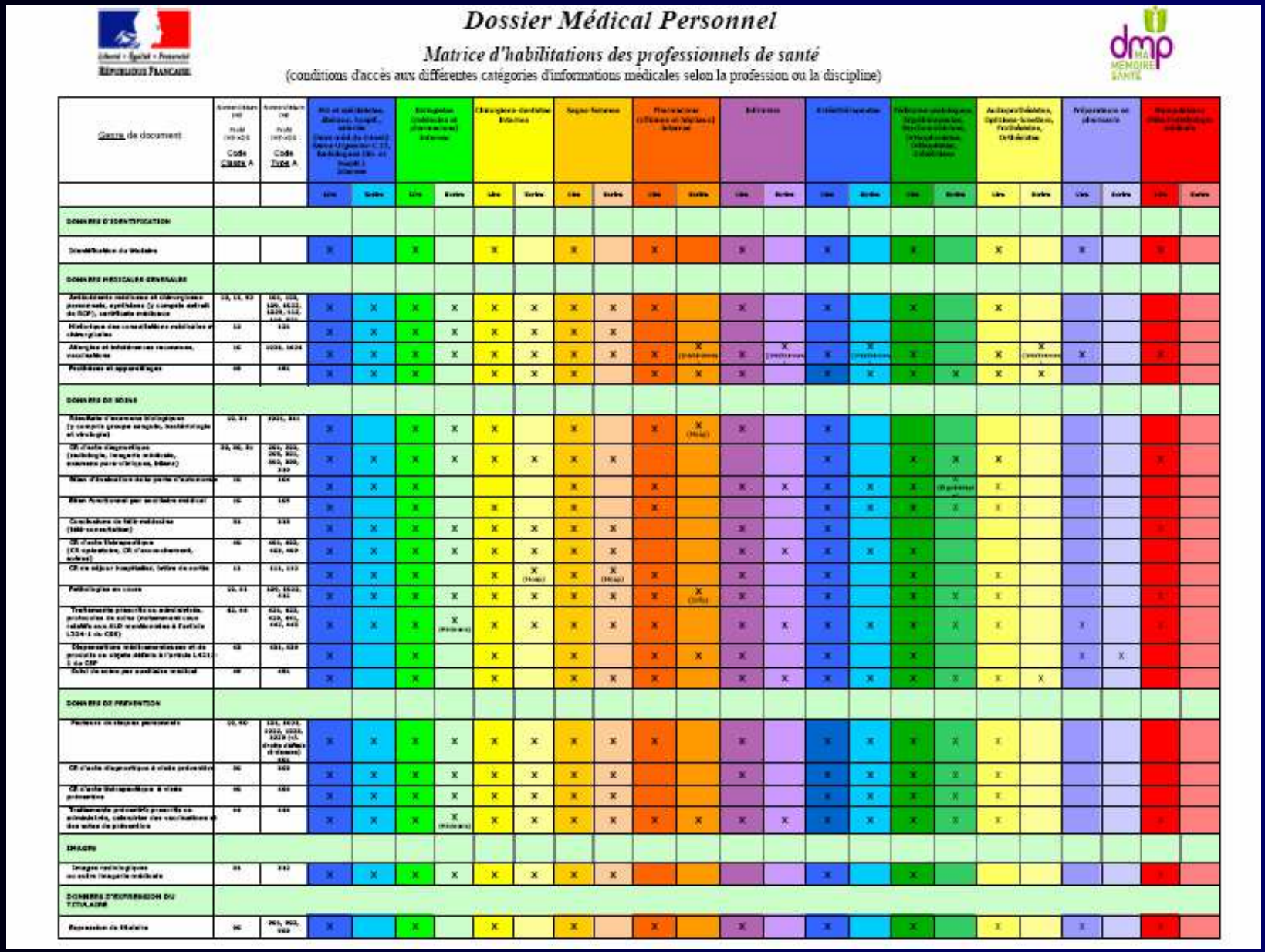


$RBAC_0$: le noyau (URA + PRA)
$RBAC_1$: Les hiérarchies
$RBAC_2$: les contraintes
$RBAC_3$: hiérarchies + contraintes

Les contraintes dans RBAC

- **Contrainte sur Utilisateur $\leftrightarrow$ Rôle**
  - Contrainte de type « Séparation des pouvoirs »
    - Exemple : rôles anesthésiste et chirurgien sont exclusifs
    - $\forall$ u, $\neg$ (URA(u,Anesthésiste) $\wedge$ URA(u,Chirurgien))
- **Contrainte sur Session $\leftrightarrow$ Rôle**
  - Un utilisateur peut cumuler plusieurs rôles mais pas les activer dans une même session
    - Contrainte de type « Séparation des tâches »
- **Contrainte sur Rôle $\leftrightarrow$ Permission**
  - Un rôle ne doit pas pouvoir cumuler certaines permissions
    - Autre contrainte de type « Séparation des tâches »

Exemple de politique RBAC



Dossier Médical Personnel

Matrice d'habilitations des professionnels de santé

(conditions d'accès aux différentes catégories d'informations médicales selon la profession ou la discipline)

Limites des modèles DAC et RBAC

- **Avec DAC, l'application s'exécutant pour le compte d'un utilisateur hérite des droits de ce dernier**
- **Avec RBAC, l'application s'exécutant pour le compte d'un utilisateur hérite des droits associés aux rôles activés dans la session ouverte par ce dernier**
- **Risque de programmes malveillants**
  - Cheval de Troie : programme qui a une fonctionnalité apparente mais qui contient des fonctions cachées
  - Objectif : transmission illégale d'informations vers le bénéficiaire du piège

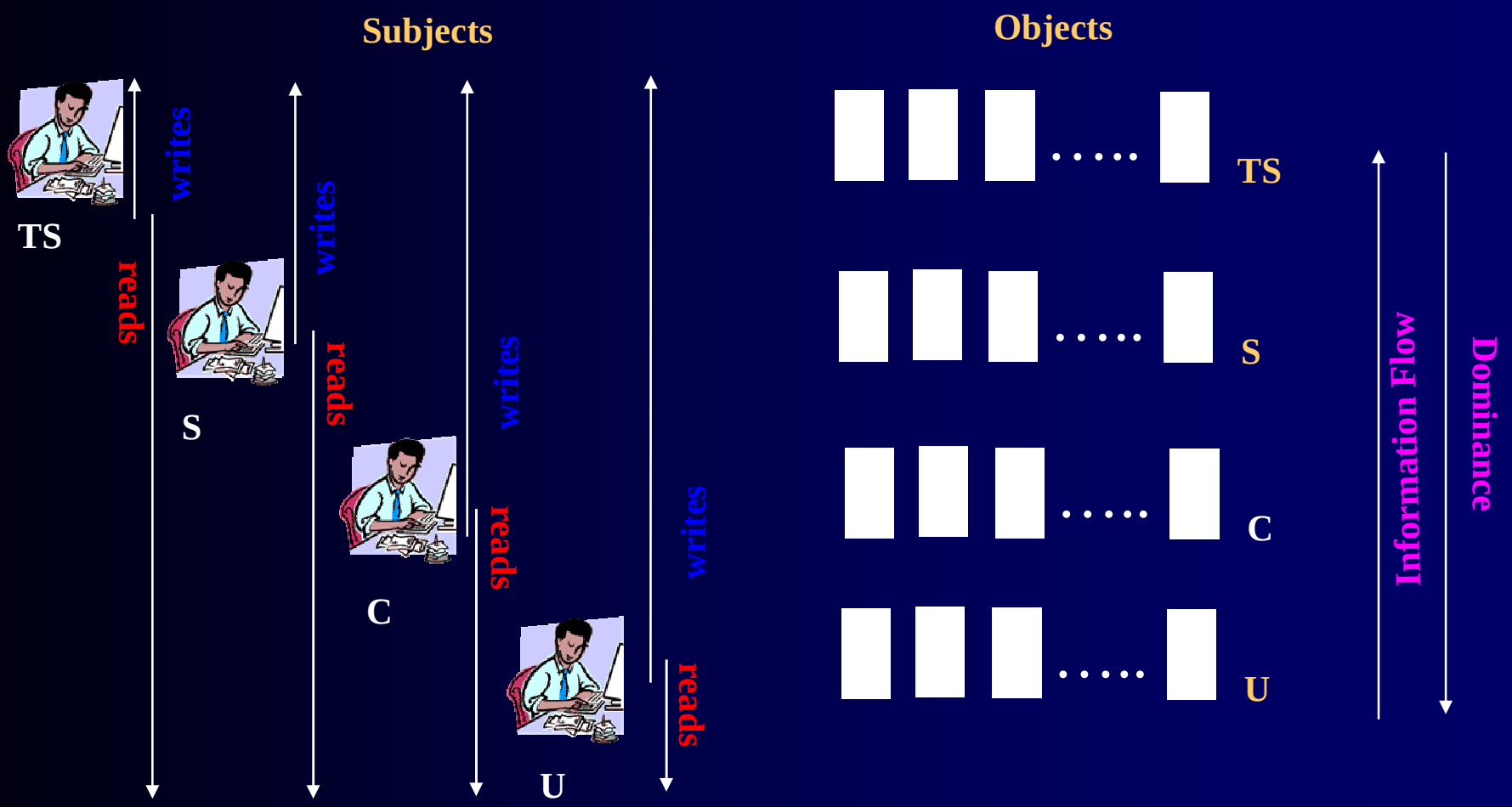
MAC : Mandatory Access Control

- **Politique de sécurité multi-niveaux**
  - Niveaux de sécurité hiérarchiques
    - Cloisonnement vertical : Unclassified < Confidentiel < Secret < Très Secret …
  - Catégories
    - Cloisonnement horizontal : cardiologie, pédiatrie, rhumatologie, ...
  - La combinaison d'un niveau de sécurité et d'une catégorie forme une classe d'accès
    - Le niveau de sécurité d'une classe d'accès associée à un objet est appelé niveau de classification
    - Le niveau de sécurité d'une classe d'accès associée à un utilisateur est appelé niveau de clearance (ou d'accréditation)
- **Exemple de systèmes supportant un modèle mandataire**
  - Oracle Label Security, Label-Based Access Control DB2, Label Security SQL-Server

Mandatory Access Control (2)

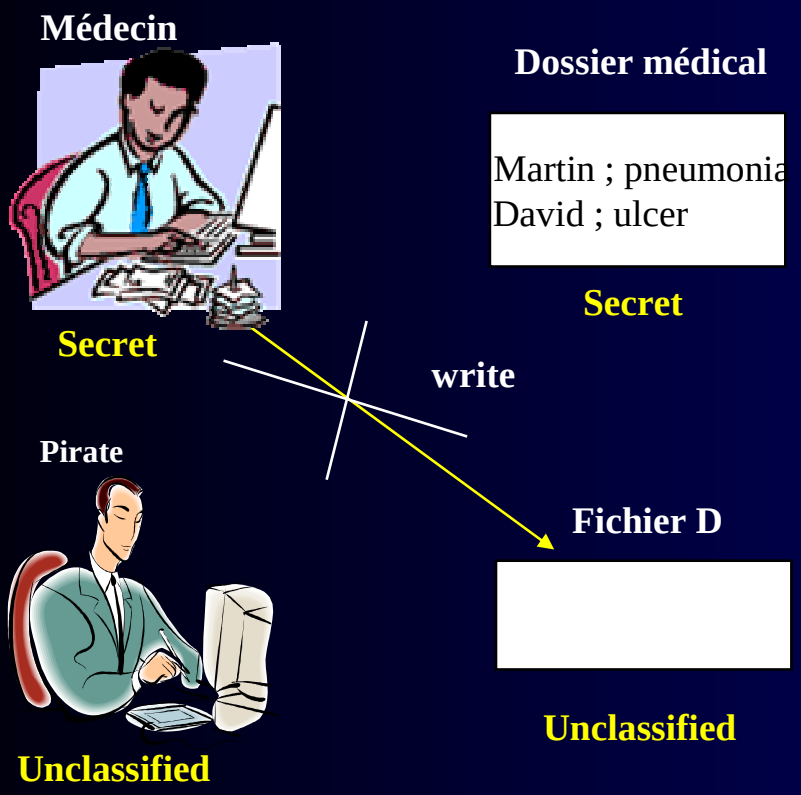
- **La décision d'accès est prise en comparant les deux classes d'accès de l'objet et du sujet**
  - No read up : un sujet est autorisé à lire un objet seulement si sa classe d'accès domine la classe d'accès de l'objet
  - No write down : un sujet est autorisé à écrire un objet seulement si sa classe d'accès est dominée par la classe d'accès de l'objet
- **Une classe d'accès c1 domine (≥) c2 ssi :**
  - Le niveau de sécurité de c1 >= niveau de sécurité de c2
  - Les catégories de c1 ⊇ c2
- **Les deux classes c1 et c2 sont dites incomparables ssi c1 ≥ c2 ni c2 ≥ c1 ne sont vérifiées**

Mandatory Access Control (3)

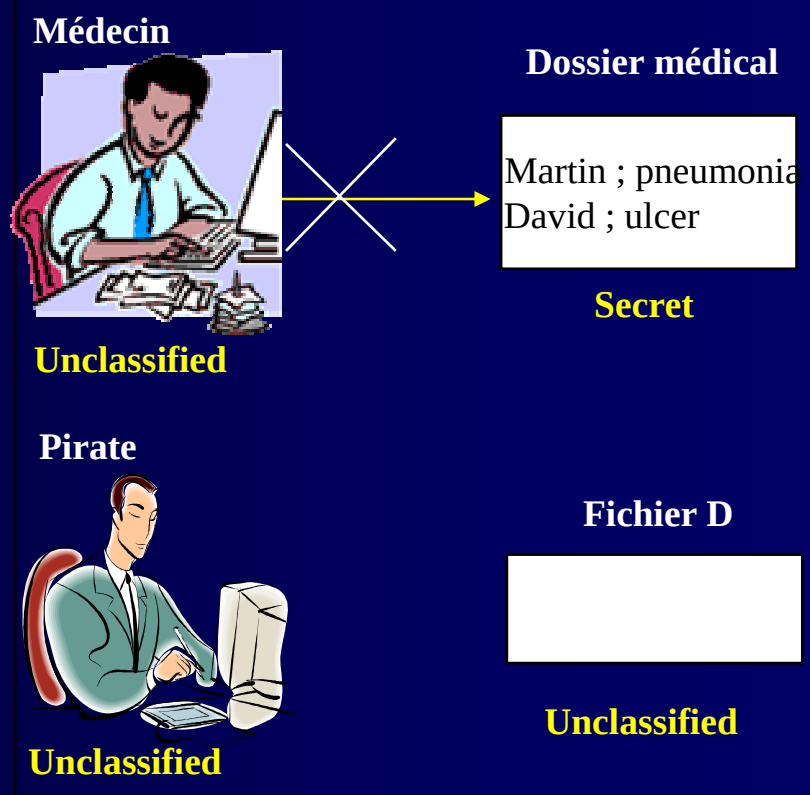


Flot d'information pour la confidentialité

Mandatory Access Control (4)



Le médecin se connecte comme un sujet **Secret**



Le médecin se connecte comme un sujet **unclassified**

Hélas, l'écriture n'est pas le seul moyen de transmettre une information (ex: wait dans un programme)

Exemple : Oracle Label Security

- **OLS: une adaptation de MAC, construite au dessus de VPD et ne nécessitant pas de programmation**
- **Data label**
  - Constitué de 3 composants (Level, Compartment, Group)
  - Intégré aux tuples dans une colonne additionnelle (déclarée par le DSA)
  - Valeurs définies par le DSA
- **Level**
  - Obligatoire, unique, hiérarchique, dénotant la sensibilité de la donnée
  - Exemple: Confidential, Sensitive and Highly Sensitive
- **Compartment**
  - Optionnel, non unique, non hiérarchique, utilisé pour compartimenter les données
  - Exemple: types de données, liste de projets ou de secteur d'activité
- **Group**
  - Optionnel, non unique, potentiellement hiérarchique, utilisé pour isoler les données par organisation
  - Exemple: FBI, CIA

Oracle Label Security

Possibilité de définir jusqu'à 999 Levels et 9999 Compartments et Groups (souvent moins de 5 niveaux sont nécessaires)

Sample Policy and Label Component Matrix	HR Policy	Law Enforcement Policy	Government Policy
Levels	Confidential Sensitive Highly Sensitive	Level 1 Level 2 Level 3	Confidential Secret Top Secret
Compartments	PII Data Investigation	Internal Affairs Drug Enforcement	Desert Storm Border Protection
Groups	HR	Local Jurisdiction FBI Dept of Justice	NATO Homeland Security

Oracle Label Security

- **Un user label est associé à chaque utilisateur**
  - Mêmes composants: Level, Compartment, Group
- **Autorisations requises pour accéder aux données**
  - Les 3 conditions ci-dessous sont requises
  - UserLabel.level ≥ DataLabel.level
  - DataLabel.compartment ⊆ UserLabel.compartment
    - **valider tous les compartments**
  - UserLabel.group ⊆ DataLabel.group
    - **valider au moins 1 groupe**
- **Exemple**
  - Une donnée de label (L2:C1,C3:G1,G2) sera accessible avec un UserLabel (L2:C1,C2,C3:G1) mais pas avec un UserLabel (L3:C1:G1,G2)

Vue macroscopique : raffinement dans les slides suivants

Oracle Label Security

- **Détail des composants du UserLabel**

User Label Authorization	Authorization Description
Maximum Level	The maximum sensitivity level a user is authorized to access. For example this might be *Sensitive* or *Highly Sensitive*.
Minimum Level	The minimum sensitivity level a user is authorized to write data. For example, an administrator can prevent users from labeling data as *Confidential* by assigning a minimum level of *Sensitive*.
Default Level	The level used by default when a user connects to the database. For example, a user can set his or her default level to *Sensitive*. When he or she connects to the system, the default level will be initialized to *Sensitive*.
Row Level	The default level used to label data inserted into the database by the user through the application or directly through a tool such as SQL*Plus.

Oracle Label Security

- **Détail des composants du UserLabel (suite)**

Read Compartments	The set of compartments assigned to the user and used during READ access mediation. For example, if a user has compartments *A, B and C*, he could view data which has compartments *A and B* but not data which has compartments *A, B, C and D*.
Write Compartments	The set of compartments assigned to the user and used during WRITE access mediation. For example, a user could be given READ and WRITE access to compartments *A and B* but READ-ONLY access to compartment *C*. If an application record was labeled with compartments *A, B and C*, the user would not be allowed to update the record because he or she does not have WRITE access on compartment *C*.
Read Groups	The set of groups assigned to the user and used during READ access mediation. For example, if a user had the group *Manager*, he could view data that has the *Manager* group but not data that had only the *Senior VP* group.
Write Groups	The set of groups assigned to the user and used during WRITE access mediation. For example, a user could be

Oracle Label Security

- **Exercice**
  - C = Confidential, S = Secret

Table	User / Data	C	S	S:A:US	S:A,B:US,UK
Assets	C::UK				
	C::US	No Access	No Access	Access	Access
Projects	C	Access	Access	Access	Access
	S	No Access	Access	Access	Access
	S:A:US	No Access	No Access	Access	Access
	S:B:UK				
	S:A,B:US				

Oracle Label Security

- **Autorisations complémentaires pouvant être données à un utilisateur ou une procédure stockée**
  - READ : Oracle ne vérifie plus les labels lors des SELECT
    - les opérations update/delete/insert restent contrôlées
  - FULL: Oracle ne vérifie plus aucun label
    - mais les droits standard sur les objets continuent à s'appliquer (ex: un GRANT SELECT ON T est toujours requis pour interroger T)
  - WRITEUP – WRITEDOWN: donne le droit au user d'augmenter (resp. réduire) le DataLabel.Level d'un tuple dans la limite de ses propres capacités
  - WRITEACCROSS: donne le droit de modifier Groups et Compartment d'un DataLabel
- **Performance ?**
  - Un test supplémentaire par tuple (sauf si READ, FULL)
  - Création d'un index bitmap sur l'attribut Label est recommandé

Synthèse sur les modèles de contrôle d'accès

- **Principe fondateur**

- **DAC**
  - Permet de structurer les Objets
- **RBAC**
  - Permet de structurer les Sujets
- **MAC**
  - Lutte contre les programmes malveillants
  - Mais permet peu de souplesse dans la définition des politiques

➔ Mais tout cela suppose que l'utilisateur passe "par la porte d'entrée" !!