www.tunisie-etudes.info

Ce document a été téléchargé depuis
www.tunisie-etudes.info

Des documents gratuits, devoirs, examens, cours, exercices, corrigés... Ainsi que toute une rubrique pour vous aider à trouver un emploi sans oublier les avis de concours en direct

Notre page Twitter :
http://www.twitter.com/TunisieEtudes
Notre page FaceBook :
http://www.facebook.com/pages/wwwtunisie-etudesinfo/257388010106



Merci d'avoir choisi www.tunisie-etudes.info
Bonne lecture et bon travail

Gérer les données et la simultanéité d'accès aux données



ORACLE

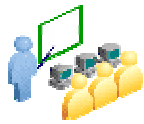
Riadh ZAAFRANI

2ème années SI - 2ème années E-services

Avril 2010

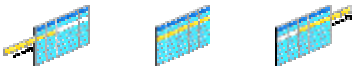
Objectifs du cours

- A la fin de ce cours, vous pourrez :
  - gérer les données en utilisant le langage SQL
  - identifier et administrer des objets PL/SQL
  - décrire les déclencheurs et les événements déclencheurs
  - surveiller et résoudre les conflits de verrouillage



Manipuler les données par l'intermédiaire du langage SQL

> SQL
> PL/SQL
> Verrous externes



```
SQL> INSERT INTO employees VALUES
  2  (9999,'Bob','Builder','bob@abc.net',NULL,SYSDATE,
  3  'IT_PROG',NULL,NULL,100,90);

1 row created.

SQL> UPDATE employees SET SALARY=6000
  2  WHERE EMPLOYEE_ID = 9999;

1 row updated.

SQL> DELETE from employees
  2  WHERE EMPLOYEE_ID = 9999;

1 row deleted.
```

Commande INSERT

- L'instruction **INSERT** de base crée une ligne à la fois.
- Via l'utilisation d'une sous-interrogation **SELECT**, vous pouvez faire en sorte que la commande **INSERT** copie des lignes d'une table vers une autre. Cette méthode est également appelée instruction **INSERT SELECT**.
- Par exemple : insert into dept_80 (
  select * from employees
  where department_id = 80);



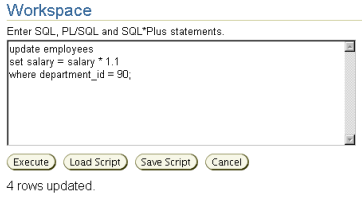
- Dans cet exemple, la table **dept_80** présente exactement la même structure que la table **employees**.

Commande INSERT

- Si tel n'est pas le cas, vous pouvez nommer les colonnes de chaque table. Les valeurs sélectionnées dans l'instruction **SELECT** sont associées respectivement aux colonnes de la table dans laquelle elles sont insérées. La correspondance des valeurs des colonnes suit l'ordre dans lequel les colonnes sont nommées dans les instructions **INSERT** et **SELECT**. Il suffit que les types de données correspondent. Par exemple :
- **insert into just_names (first, last)**
  (select first_name, last_name from employees);
- Dans cet exemple, la table **just_names** ne comporte que deux colonnes qui présentent le même type de données que les colonnes **first_name** et **last_name** de la table **employees**.
- La méthode **INSERT SELECT** permet de charger en masse des données d'une ou de plusieurs tables vers une autre table.

Commande UPDATE

- La commande **UPDATE** est utilisée pour modifier les lignes existantes d'une table. Le nombre de lignes modifiées par la commande **UPDATE** dépend de la condition **WHERE**. Si la clause **WHERE** est omise, toutes les lignes sont modifiées. Si aucune ligne ne satisfait à la condition **WHERE**, aucune ligne n'est modifiée.





Commande DELETE

- La commande **DELETE** est utilisée pour supprimer des lignes existantes d'une table. Le nombre de lignes supprimées dépend de la condition **WHERE**. Si la clause **WHERE** est omise, toutes les lignes sont supprimées. Si aucune ligne ne satisfait à la condition **WHERE**, aucune ligne n'est supprimée. Comme le montre l'exemple, lorsqu'aucune ligne n'est supprimée, aucune erreur n'est générée.

Workspace

Enter SQL, PL/SQL and SQL*Plus statements.

```
delete from employees
where department_id = 200
```

Execute | Load Script | Save Script | Cancel

0 rows deleted.



Commande MERGE

- La commande **MERGE** réunit en une commande unique les opérations de mise à jour (**UPDATE**) et d'insertion (**INSERT**). Elle permet de fusionner les données d'une source avec celles d'une autre source, et éventuellement d'insérer de nouvelles lignes ou de mettre à jour certaines colonnes d'une ligne existante.

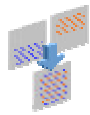
Workspace

Enter SQL, PL/SQL and SQL*Plus statements.

```
MERGE INTO jobs j
 USING (SELECT * FROM jobs_acquisition) a
 ON (j.job_id = a.job_id)
 WHEN MATCHED THEN UPDATE SET j.job_title = a.job_title
 WHEN NOT MATCHED THEN INSERT
  (j.job_id, j.job_title, j.min_salary, j.max_salary)
 VALUES (a.job_id, a.job_title, a.min_salary, a.max_salary)
```

Execute | Load Script | Save Script | Cancel

5 rows merged.



Commande MERGE

- Considérons l'exemple précédent. Certaines des données de la table **JOBS** sont semblables à ce qui suit :

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
----------	----------	----------	----------
AD_PRES	President	20000	40000
FI_ACCOUNT	Accountant	4200	9000
ST_CLERK	Stock Clerk	2000	5000
IT_PROG	Programmer	4000	10000

- La table **JOBS_ACQUISITION** contient les données suivantes :

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
----------	----------	----------	----------
AD_PRES	VP	20000	40000
DBA	DB Admin	4200	9000
SA	Sys Admin	2000	5000

- La commande **MERGE** insère dans la table **JOBS** toute ligne qui comporte une nouvelle valeur **JOB_ID**, et met à jour la ligne **JOBS** existante avec les données **JOB_TITLE** si l'**ID JOB_ID** existe déjà. Ainsi, le poste "President" est remplacé par "VP", et les nouveaux postes "SA" et "DBA" sont ajoutés.

Commandes COMMIT et ROLLBACK

- Par défaut, une commande LMD exécutée n'est pas validée (**commit**). Divers outils (notamment *i*SQL*Plus) disposent d'options pouvant être configurées pour valider chaque commande ou un groupe de commandes.
- Avant l'exécution d'une commande **COMMIT** ou **ROLLBACK**, les modifications sont en attente. Seul l'utilisateur ayant apporté la modification est autorisé à voir les données modifiées. Les autres utilisateurs peuvent sélectionner ces mêmes données, mais elles leur apparaissent telles qu'elles étaient avant la modification. Les autres utilisateurs ne peuvent pas exécuter d'opération LMD sur les données qu'un autre utilisateur a modifiées.
- Cette opération est contrôlée automatiquement par le mécanisme de verrouillage de la base de données Oracle.

Langage PL/ SQL

SQL
> **PL/SQL**
Verrous externes

- Le langage PL/SQL est un langage de programmation Oracle de quatrième génération, qui fournit des extensions procédurales du langage SQL. Il offre un environnement de programmation commun pour les bases de données et les applications Oracle, quel que soit le système d'exploitation ou la plate-forme matérielle.
- Grâce au langage PL/SQL, vous pouvez manipuler les données à l'aide d'instructions SQL et contrôler le flux des programmes par le biais de structures procédurales telles que `IF-THEN`, `CASE` et `LOOP`. Vous pouvez également déclarer des constantes et des variables, définir des procédures et des fonctions, utiliser des ensembles et des types d'objet, ou encore intercepter les erreurs lors de l'exécution. Un programme PL/SQL peut également appeler des programmes écrits dans d'autres langages, tels que C, C++ et Java.

Langage PL/ SQL

- Le langage PL/SQL permet par ailleurs de protéger les données. Pour pouvoir effectuer un appel, l'appelant n'a pas besoin de connaître les structures de données lues ou manipulées. Il n'est pas non plus obligé de disposer de droits d'accès pour ces objets. Il lui suffit d'être autorisé à exécuter le programme PL/SQL.
- Etant donné qu'il s'exécute dans la base de données, le code PL/SQL est très efficace pour les opérations impliquant des volumes importants de données et limite le trafic réseau des applications.



Administrer les objets PL/ SQL

- Les administrateurs de base de données doivent pouvoir :
  - identifier les objets PL/SQL qui posent problème
  - recommander l'utilisation appropriée du langage PL/SQL
  - charger des objets PL/SQL dans la base de données
  - assister les développeurs PL/SQL dans la résolution des problèmes



- Database Control vous permet d'accéder aux objets PL/SQL (cliquez sur l'onglet **Administration** sous **Schema**). Lorsque vous cliquez sur le type d'objet, vous pouvez afficher, modifier et créer le type d'objet PL/SQL sélectionné.

Objets PL/ SQL

- Il existe de nombreux types d'objet de base de données PL/SQL :
  - **Package :** Un package est un ensemble de procédures et de fonctions liées de façon logique. Cette partie d'un package est également appelée spécification et constitue l'interface avec les applications. Elle déclare les types, variables, constantes, exceptions, curseurs et sous-programmes pouvant être utilisés.
  - **Corps du package :** Le corps définit de manière complète les curseurs et les sous-programmes, implémentant ainsi la spécification. Il contient les détails de l'implémentation et les déclarations privées, lesquels sont masqués pour l'appelant.



Objets PL/ SQL

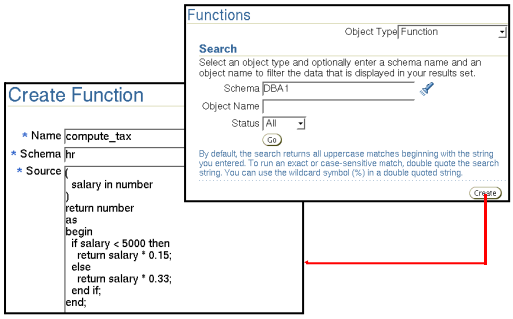
- **Corps du type :** Il s'agit de l'ensemble de méthodes (procédures et fonctions) associées à des types de données définis par l'utilisateur.
- **Procédure :** Une procédure est un bloc PL/SQL qui effectue une opération spécifique.
- **Fonction :** Une fonction est un bloc PL/SQL qui renvoie une valeur unique à l'aide de la commande PL/SQL **RETURN**. Il s'agit d'une procédure qui comporte une valeur renvoyée.
- **Déclencheur :** Un déclencheur (trigger) est un bloc PL/SQL exécuté lorsqu'un événement particulier se produit dans la base de données. Ces événements peuvent être basés sur une table, par exemple lorsqu'une ligne est insérée dans la table. Il peut également s'agir d'événements de base de données (lorsqu'un utilisateur se connecte à la base de données, par exemple).

Fonctions

- Les fonctions PL/SQL sont généralement utilisées pour calculer une valeur. Il existe de nombreuses fonctions intégrées, telles que **SYSDATE**, **SUM**, **AVG** et **TO_DATE**. Les développeurs créent également leurs propres fonctions lors de l'écriture d'applications. Le code d'une fonction PL/SQL *doit* contenir une instruction **RETURN**. Pour créer des **fonctions PL/SQL**, vous devez entrer un **nom**, un **schéma** et un **code source**, comme l'indique la diapositive suivante. La fonction `compute_tax` présentée dans la diapositive est créée à l'aide de la commande SQL suivante :

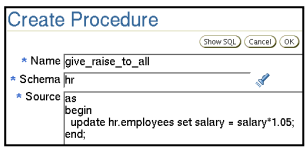
```
CREATE OR REPLACE FUNCTION compute_tax (salary in NUMBER)
RETURN NUMBER AS
BEGIN
  IF salary< 5000 THEN
    RETURN salary*.15;
  ELSE
    RETURN salary*.33;
  END IF;
END;
```

Fonctions



Procédures

- Les procédures sont utilisées pour effectuer une action spécifique. Elles présentent les caractéristiques suivantes :
  - Elles transmettent et reçoivent des valeurs par l'intermédiaire d'une liste d'arguments.
  - Elles sont appelées à l'aide de la commande **CALL**.

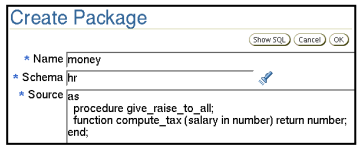


Packages

- Les packages sont des regroupements de fonctions et de procédures. Le regroupement de fonctions et de procédures dans un même package présente des avantages en termes de performances et de maintenance. Chaque package doit être constitué de deux objets de base de données compilés séparément :
  - **Spécification du package :** Cet objet (parfois appelé en-tête du package) est de type **PACKAGE** et contient uniquement la définition des procédures, des fonctions et des variables du package.
  - **Corps du package :** Cet objet est de type **PACKAGE BODY** et contient le code proprement dit des sous-programmes définis dans la spécification du package.

Packages

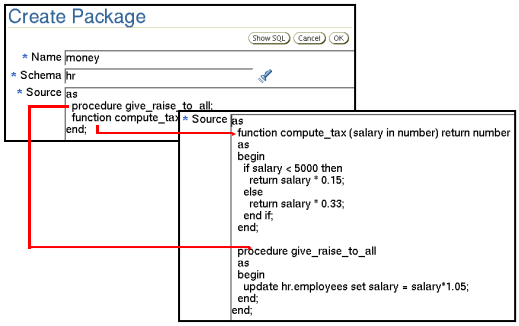
- Les procédures et fonctions appelées à partir d'un package sont appelées par l'intermédiaire d'une notation à points :
  - *nom_package.nom_procédure ou nom_function*
- Pour le package présenté, les sous-programmes peuvent être appelés de la manière suivante :
  - SQL> SELECT money.compute_tax(salary) FROM hr.employees WHERE employee_id=107;
  - SQL> EXECUTE money.give_raise_to_all;



Spécification du package



Spécification et corps d'un package

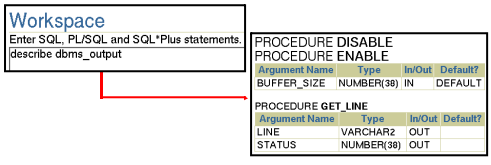


Spécification et corps d'un package

- Le corps du package présente les caractéristiques suivantes :
  - Il est distinct de la spécification du package. Par conséquent, le code du corps peut être modifié et recompilé sans que les autres objets qui dépendent de la spécification soient marqués comme non valides.
  - Il contient le code des sous-programmes définis dans la spécification du package. La spécification indique comment appeler les sous-programmes dans le package. Le corps constitue la section de code.
  - Il ne peut pas être compilé tant que la spécification du package n'a pas été compilée. Vous pouvez créer une spécification sans corps, mais vous ne pouvez pas créer un corps sans spécification.
  - Il peut être encapsulé afin de masquer les détails du code. **Wrap** est un programme autonome qui crypte le code source PL/SQL afin que vous puissiez fournir des applications PL/SQL sans exposer le code source.

Packages intégrés

- La base de données Oracle est fournie avec plus de 350 packages PL/SQL intégrés, qui offrent :
  - des utilitaires d'administration et de maintenance
  - des fonctionnalités étendues
- Utilisez la commande **DESCRIBE** pour afficher les sous-programmes.



Packages intégrés

- Les packages utilisés par un administrateur dépendent du type d'application que la base de données exécute. Voici quelques-uns des packages d'administration et de maintenance les plus courants :
- **DBMS_STATS :** Collecte, affichage et modification des statistiques destinées à l'optimiseur.
- **DBMS_OUTPUT :** Génération d'une sortie à partir d'un programme PL/SQL.
- **DBMS_SESSION :** Accès PL/SQL aux instructions **ALTER SESSION** et **SET ROLE**.
- **DBMS_RANDOM :** Génération de nombres aléatoires.

Packages intégrés

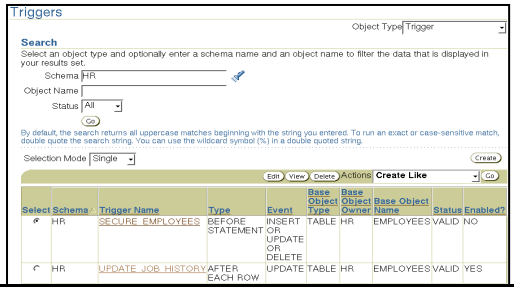
- **DBMS_UTILITY :** Obtention d'informations relatives à l'heure, au temps CPU et à la version. Calcul d'une valeur de hachage (hash value) et réalisation de nombreuses autres fonctionnalités.
- **DBMS_SCHEDULER :** Fonctions et procédures de planification pouvant être appelées à partir de n'importe quel programme PL/SQL.
- **DBMS_CRYPTO :** Cryptage et décryptage des données de la base.
- **UTL_FILE :** Opérations de lecture et d'écriture dans les fichiers du système d'exploitation à partir d'un programme PL/SQL.

Déclencheurs

- Les **déclencheurs (triggers)** sont des objets de code PL/SQL stockés dans la base de données, qui s'exécutent automatiquement lorsqu'un événement se produit.
- La base de données Oracle permet d'utiliser de nombreuses actions comme événement déclencheur, par exemple l'insertion dans une table, la connexion d'un utilisateur à la base, ou bien encore la tentative de suppression d'une table ou de modification des paramètres d'audit. Les déclencheurs peuvent appeler d'autres procédures ou fonctions. Il est préférable que le code d'un déclencheur soit très court.
- Les administrateurs de base de données utilisent des déclencheurs pour faciliter l'audit basé sur les données, pour appliquer des contraintes complexes et pour automatiser de nombreuses tâches.

Déclencheurs

- Par exemple, le déclencheur **SECURE_EMPLOYEES** illustré dans la diapositive ci-dessous enregistre toutes les instructions LMD exécutées sur une table.



Evénements déclencheurs

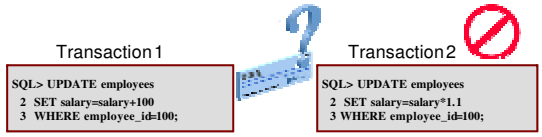
- De nombreux événements peuvent servir de déclencheur (trigger). Ils sont répartis en trois catégories.
  - Les **déclencheurs de type événement LMD** s'exécutent lorsque les instructions modifient des données.
  - Les **déclencheurs de type événement LDD** s'exécutent lorsque les instructions créent un objet ou modifient un objet de quelque manière que ce soit.
  - Les **déclencheurs de type événement de base de données** s'exécutent lorsque certaines actions se produisent dans la base de données.
- Pour la plupart des déclencheurs, vous pouvez indiquer si l'exécution doit avoir lieu avant ou après l'événement. Un déclencheur LMD peut être conçu de manière à s'exécuter une seule fois pour l'instruction, ou pour chaque ligne modifiée.

Evénements déclencheurs

Type d'événement	Exemples d'événements
LMD	INSERT, UPDATE, DELETE
LDD	CREATE, DROP, ALTER, GRANT, REVOKE, RENAME
Base de données	LOGON, LOGOFF, STARTUP, SHUTDOWN, SERVERERROR

Verrous externes

- Avant que la base de données n'autorise une session à modifier des données, la session doit d'abord verrouiller ces données. Un **verrou externe (lock)** accorde à la session le contrôle exclusif des données, de sorte qu'aucune autre transaction ne puisse modifier les données verrouillées jusqu'à la libération du verrou.
- Oracle prend en charge le verrouillage **manuel** et le verrouillage **automatique**. Les verrous obtenus automatiquement choisissent toujours le niveau de verrouillage le plus bas possible, afin de limiter les conflits potentiels avec d'autres transactions.

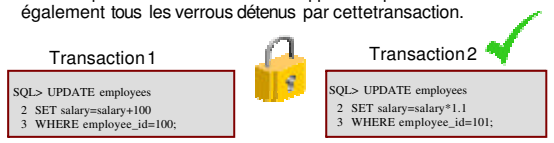


Mécanisme de verrouillage

- Le mécanisme de verrouillage est conçu pour fournir le degré maximal de simultanéité d'accès aux données dans la base.
- Les transactions qui modifient des données acquièrent des **verrous externes (locks)** au niveau ligne plutôt qu'au niveau bloc ou table. Les modifications apportées aux objets (telles que les déplacements de tables) obtiennent des verrous externes au niveau objet, plutôt qu'au niveau de la base de données complète ou du schéma.
- Les **interrogations de données** ne nécessitent pas de verrous externes et une interrogation réussit même si quelqu'un a verrouillé les données (en effet, c'est toujours la valeur originale avant verrouillage qui est affichée, reconstruite à partir des informations d'annulation).

Mécanisme de verrouillage

- Lorsque plusieurs transactions doivent verrouiller la même ressource, la première transaction qui demande le verrou externe l'obtient. Les autres transactions attendent jusqu'à la fin de la première transaction. Le mécanisme de mise en **file d'attente** est automatique et ne nécessite aucune intervention de l'administrateur.
- Tous les verrous externes sont libérés à la fin d'une transaction. Les transactions sont terminées lorsqu'une opération de validation (**COMMIT**) ou d'annulation (**ROLLBACK**) est exécutée. En cas d'échec d'une transaction, le processus en arrière-plan qui annule automatiquement les modifications apportées par la transaction libère également tous les verrous détenus par cettetransaction.



Simultanéité d'accès aux données

- Le mode de verrouillage par défaut est un **verrouillage détaillé au niveau ligne**. Différentes transactions peuvent mettre à jour différentes lignes de la même table sans interférer les unes avec les autres.

Heure :	Transaction 1	`UPDATE hr.employees SET salary=salary+100 WHERE employee_id=100;`
	Transaction 2	`UPDATE hr.employees SET salary=salary+100 WHERE employee_id=101;`
09:00:00	Transaction 3	`UPDATE hr.employees SET salary=salary+100 WHERE employee_id=102;`
	...	...
	Transaction *x*	`UPDATE hr.employees SET salary=salary+100 WHERE employee_id=xxx;`

Simultanéité d'accès aux données

- Le modèle par défaut est le verrouillage au niveau ligne, mais la base de données Oracle prend également en charge le verrouillage manuel à des niveaux supérieurs, si nécessaire :
  - **SQL> LOCK TABLE employees IN EXCLUSIVE MODE;**
  - **Table(s) Locked.**
- Avec l'instruction ci-dessus, toute autre transaction qui tente de mettre à jour une ligne dans la table verrouillée doit attendre que la transaction ayant généré la demande de verrouillage soit terminée. **EXCLUSIVE** est le mode de verrouillage le plus strict.
- Vous disposez également des modes de verrouillage suivants :
  - **ROW SHARE :** Permet des accès simultanés à la table verrouillée, mais empêche les sessions de verrouiller l'ensemble de la table pour un accès exclusif.

Simultanéité d'accès aux données

- **ROW EXCLUSIVE :** Identique au mode **ROW SHARE**, mais empêche également le verrouillage en mode partagé (**SHARE**). Les verrous externes **ROW EXCLUSIVE** sont obtenus automatiquement lors de la mise à jour, de l'insertion ou de la suppression de données.
- **SHARE :** Permet les interrogations simultanées, mais empêche les mises à jour de la table verrouillée. Un verrou **SHARE** est nécessaire (et automatiquement demandé) pour la création d'un index sur une table.
- **SHARE ROW EXCLUSIVE :** Utilisé pour l'interrogation d'une table entière et pour autoriser d'autres utilisateurs à interroger des lignes de la table, mais empêche les autres utilisateurs de verrouiller la table en mode **SHARE** ou de mettre à jour des lignes.
- **EXCLUSIVE :** Autorise les interrogations sur la table verrouillée, mais empêche toute autre activité sur cette table. Un verrou externe **EXCLUSIVE** est nécessaire pour supprimer une table.

Simultanéité d'accès aux données

- Comme pour n'importe quelle demande de verrou externe, les instructions de verrouillage manuel attendent que toutes les sessions détenant déjà des verrous (ou ayant précédemment demandé des verrous) aient libéré ces verrous. La commande **LOCK** accepte un argument spécial qui contrôle le comportement d'attente, à savoir **NOWAIT**.
- L'argument **NOWAIT** vous redonne immédiatement le contrôle si la table désignée est déjà verrouillée par une autre session.

```
SQL> LOCK TABLE hr.employees IN SHARE MODE NOWAIT;
LOCK TABLE hr.employees IN SHARE MODE NOWAIT
           *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified
```

- Il est généralement inutile de verrouiller manuellement des objets. Le mécanisme de verrouillage automatique fournit la simultanéité d'accès aux données nécessaire pour la plupart des applications.

Verrous LMD

- Transaction 1 / Transaction 2

```
SQL> UPDATE employees
  2  SET salary=salary*1.1
  3  WHERE employee_id= 107;
1 row updated.
```

```
SQL> UPDATE employees
  2  SET salary=salary*1.1
  3  WHERE employee_id= 106;
1 row updated.
```

- Chaque transaction LMD acquiert deux verrous :
  - Un verrou sur ligne de type **EXCLUSIVE** pour les lignes mises à jour.
  - Un verrou sur table de type **ROW EXCLUSIVE** pour la table mise à jour. Ce verrou permet d'empêcher une autre session de verrouiller la table entière (par exemple pour la supprimer ou la vider) pendant que la modification est entrain d'être apportée.

Mécanisme de mise en file d'attente

- Les demandes de verrous externes (locks) sont automatiquement mises en **file d'attente**. Dès que la transaction détenant un verrou est terminée, la session suivante de la file reçoit le verrou.
- Le mécanisme de mise en file d'attente effectue le suivi de l'ordre dans lequel les verrous sont demandés, ainsi que le suivi du mode de verrouillage demandé.



Mécanisme de mise en file d'attente

- Les sessions qui détiennent déjà un verrou peuvent demander la *conversion* de ce verrou sans avoir à accéder à la fin de la file d'attente.
- Par exemple, supposons qu'une session détienne un verrou sur table de type **SHARE**. La session peut demander la conversion de ce verrou en un verrou de type **EXCLUSIVE**.
- Dès lors que personne d'autre ne détient de verrou de type **EXCLUSIVE** ou **SHARE** sur la table, un verrou de type **EXCLUSIVE** est accordé à la session qui détient le verrou de type **SHARE** sans que celle-ci ait à patienter de nouveau dans la file d'attente.

Conflits de verrouillage

- Les **conflits de verrouillage** sont fréquents, mais ils sont généralement résolus avec le temps ou via le mécanisme de mise en file d'attente.

Transaction 1	Heure	Transaction 2
`UPDATE employees SET salary=salary+100 WHERE employee_id=100;` `1 row updated.`	9:00:00	`UPDATE employees SET salary=salary+100 WHERE employee_id=101;` `1 row updated.`
`UPDATE employees SET COMMISION_PCT=2 WHERE employee_id=101;` La session patiente en file d'attente en raison d'un conflit de verrouillage.	9:00:05	`SELECT sum(salary) FROM employees;` `SUM(SALARY)` `----------` `692634`
La session est toujours en attente.	16:30:00	Nombreuses opérations de sélection, d'insertion, de mise à jour et de suppression au cours des dernières 7,5 heures, mais pas de validation ni d'annulation.
`1 row updated.` La session se poursuit.	16:30:01	`commit;`

Conflits de verrouillage

- Dans de rares cas, un conflit de verrouillage peut nécessiter l'intervention de l'administrateur.
- Dans l'exemple, la transaction 2 obtient un verrou externe (lock) sur une ligne unique à 9:00:00 et l'utilisateur oublie de valider (commit) la transaction, laissant ainsi le verrou en place.
- A 9:00:05, la transaction 1 tente de mettre à jour la table entière, opération qui nécessite un verrou sur toutes les lignes. La transaction 1 est bloquée par la transaction 2 jusqu'à la validation de cette dernière, à 16:30:01.
- L'utilisateur qui tente d'effectuer la transaction 1 est amené à demander l'aide de l'administrateur, qui doit détecter et résoudre le conflit.

Causes possibles des conflits de verrouillage

- Le plus souvent, les conflits de verrouillage sont dus à une modification non validée, mais il existe d'autres causes possibles :
- **Transactions longues :** De nombreuses applications utilisent le traitement batch pour effectuer des mises à jour en masse. Ces traitements batch sont généralement planifiés pendant les périodes de faible activité (ou d'absence d'activité) des utilisateurs, mais il peut arriver qu'ils ne soient pas terminés ou que leur exécution soit trop longue pour tenir dans la période de faible activité. Les conflits de verrouillage se produisent fréquemment lorsque le traitement des transactions et le traitement batch sont effectués simultanément.

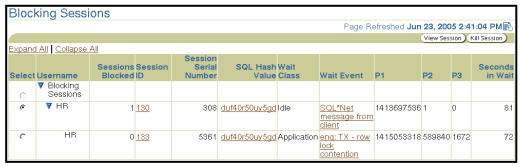


Causes possibles des conflits de verrouillage

- **Niveaux de verrouillage inutilement élevés :** Les bases de données n'admettent pas toutes le verrouillage au niveau ligne (Oracle a inclu la prise en charge des verrous au niveau ligne en 1988, dans la version 6).
- Certaines bases de données nécessitent donc toujours un verrouillage au niveau page ou au niveau table. Les développeurs qui écrivent des applications devant s'exécuter sur de nombreuses bases de données différentes appliquent souvent des niveaux de verrouillage très élevés, de sorte que la base de données Oracle se comporte de la même façon que les systèmes de base de données les moins performants. En outre, les développeurs qui débutent sur Oracle écrivent parfois du code avec des niveaux de verrouillage plus élevés que nécessaires pour la base de données Oracle.

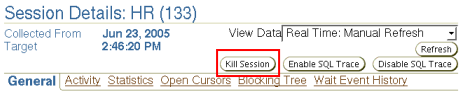
Détecter les conflits de verrouillage

- Utilisez la page **Blocking Sessions** d'Enterprise Manager pour localiser les conflits de verrouillage. Les demandes de verrous externes (locks) qui provoquent des conflits sont indiquées de façon hiérarchique. La session détenant le verrou figure en haut. Elle est suivie des sessions en attente de verrou.
- Pour chaque session impliquée dans le conflit, le nom utilisateur, l'ID de session et la durée d'attente de la session (en secondes) sont indiqués.



Résoudre les conflits de verrouillage

- La résolution d'un conflit de verrouillage passe par la libération du verrou détenu par la session. Le meilleur moyen d'y parvenir consiste à contacter l'utilisateur et à lui demander de mettre fin à la transaction.
- En cas d'urgence, l'administrateur peut mettre fin à la session détenant le verrou en cliquant sur le bouton **Kill Session**.

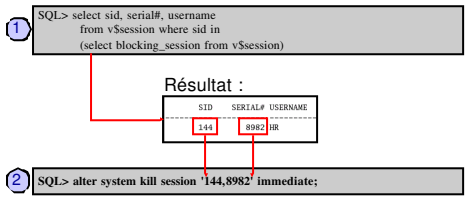


Résoudre les conflits de verrouillage

- N'oubliez pas que lorsqu'une session est fermée, tout le travail de la transaction en cours est perdu (la transaction est annulée (roll back)).
- Un utilisateur dont la session est fermée doit se reconnecter et effectuer de nouveau tout le travail réalisé depuis la dernière validation (commit) de la session.
- Les utilisateurs dont les sessions ont été fermées reçoivent l'erreur suivante lors de leur tentative suivante d'exécution d'une instruction SQL :
  - **ORA-03135: connection lost contact**

Résoudre les conflits de verrouillage à l'aide d'instructions SQL

- La table **v$session** contient des détails sur toutes les sessions connectées. **blocking_session** est l'ID de la session bloquante.
- Par conséquent, si vous exécutez une interrogation afin d'obtenir les valeurs **SID** et **SERIAL#**, où **SID** correspond à un ID de session bloquante, vous disposerez des informations nécessaires pour réaliser l'opération **kill session**.



"Verrous mortels"

- Un **"verrou mortel" (deadlock)** est un type particulier de conflit de verrouillage. Un "verrou mortel" apparaît lorsque plusieurs sessions attendent des données verrouillées par l'une d'elles. Etant donné que chacune attend l'autre, aucune d'entre elles ne peut terminer la transaction afin de résoudre le conflit.
- La base de données Oracle détecte automatiquement les **"verrous mortels"** et met fin à l'instruction en générant une erreur. En réponse à cette erreur, il convient de réaliser une opération de validation (**commit**) ou d'annulation (**rollback**), ce qui a pour effet de libérer touts les verrous de la session et de permettre à l'autre session de poursuivre sa transaction.

"Verrous mortels"

- Dans la **Transaction 1**, l'utilisateur doit effectuer une validation ou une annulation en réponse à l'erreur **"deadlock detected"**. S'il procède à une validation, il doit soumettre à nouveau la deuxième mise à jour pour terminer sa transaction. S'il effectue une annulation, il doit soumettre à nouveau les deux instructions pour réaliser la transaction.

Transaction 1		Transaction 2
UPDATE employees SET salary = salary x 1.1 WHERE employee_id = 1000;	9:00	UPDATE employees SET manager = 1342 WHERE employee_id = 2000;
UPDATE employees SET salary = salary x 1.1 WHERE employee_id = 2000;	9:15	UPDATE employees SET manager = 1342 WHERE employee_id = 1000;
ORA-00060: Deadlock detected while waiting for resource	9:16	

Synthèse

- Ce cours vous a permis d'apprendre à :
  - gérer les données en utilisant le langage SQL
  - identifier et administrer des objets PL/SQL
  - décrire les déclencheurs et les événements déclencheurs
  - surveiller et résoudre les conflits de verrouillage



www.tunisie-etudes.info

Ce document a été téléchargé depuis
www.tunisie-etudes.info

Des documents gratuits, devoirs, examens, cours, exercices, corrigés... Ainsi que toute une rubrique pour vous aider à trouver un emploi sans oublier les avis de concours en direct

Notre page Twitter :
http://www.twitter.com/TunisieEtudes
Notre page FaceBook :
http://www.facebook.com/pages/wwwtunisie-etudesinfo/257388010106



Merci d'avoir choisi www.tunisie-etudes.info
Bonne lecture et bon travail