**Лабораторная работа № 12 Транзакции, блокировки**

 Настройка SQL\*Plus. Необходимо включить режим ECHO и вывести протокол лаб. работы в файл *<Фамилия студента>.txt*. Этот файл является отчетом о проделанной лаб. работе.

SQL> set echo on

SQL> spool c:\spool. txt

……………………………………..

Завершить протокол лаб. работы (команда spool off) и направить результаты преподавателю. SQL> spool off

Модель транзакций реляционной СУБД Oracle основана на «единице работы» (unit of work) и поддерживается большую часть операций над транзакциями (нельзя использовать ROLLBACK FORSE). Транзакция неявно начинается с началом сеанса или при выполнении первой команды SQL после последней команды COMMIT или ROLLBACK (фактически при первом изменении данных). Транзакция заканчивается при выполнении команды COMMIT или ROLLBACK.

Транзакция не зависит от блоков PL/SQL. Транзакции могут охватывать несколько системных блоков, или в одном блоке может быть несколько транзакций (автономных). PL/SQL поддерживает следующие команды для транзакций: COMMIT, ROLLBACK, SAVEPOINT, TRANSACTION и LOCK TABLE.

COMMIT делает изменения в БД постоянными и видимыми для других сеансов в соответствии со следующим синтаксисом:

 COMMIT [WORK] [COMMENT *текст*];

 где WORK – необязательное ключевое слово, применяется для улучшения читаемости и соответствия стандарту. Необязательный комментарий COMMENT текст может иметь длину до 50 символов.

ROLLBACK отменяет не зафиксированные изменения, сделанные в текущей транзакции, влияет до начала транзакции или до указанной точки сохранения.

ROLLBACK [WORK] [ TO [SAVEPOINT] *имя\_точки\_сохранения*];

 SAVEPOINT устанавливает точку сохранения (именованную точку обработки) для текущей транзакции. Установка точки транзакции делает возможным выполнение частичного отката.

SAVEPOINT *имя\_точки\_сохранения;*

 где *имя\_точки\_сохранения*– это необъявленный идентификатор. Внутри транзакции может быть установлено несколько точек сохранения. Если повторно использовать имя точки сохранения то точка передвинется на новую позицию и откат к исходной позиции данной точки сохранения будет невозможен.

 SET TRANSACTION управляет типом транзакции

SET TRANSACTION *тип\_транзакции* NAME *имя*

*где имя*– имя транзакции, доступное на протяжении ее выполнения; а *тип\_транзакции м*ожет принимать следующие значения:

- READ ONLY – означает начало транзакции только для чтения. Указывает СУБД, что следует обеспечить целостность по чтению БД в пределах транзакции (по умолчанию для команды). Транзакцию заканчивают команды COMMIT или ROLLBACK. Внутри такой транзакции разрешены только команды LOCK TABLE, SELECT, SELECT INTO, OPEN, FETCH, CLOSE, COMMIT и ROLLBACK. Попытка выполнения в транзакции только для чтения других команд, такие как INSERT или UPDATE, приводит к появлению ошибки ORA-1456.

- READ WRITE – обозначает начало транзакции READ WRITE; это тип по умолчанию.

В следующем примере выполняется инструкция UPDATE, чтобы начать транзакцию, и запрашивается V $ TRANSACTION для получения подробной информации о транзакции:

**Задание 1.**

SQL> UPDATE hr.employees SET salary=salary;

SQL> SELECT XID AS "txn id", XIDUSN AS "undo seg", XIDSLOT AS "slot", XIDSQN AS "seq", STATUS AS "txn status"

FROM V$TRANSACTION;

SQL> UPDATE hr.employees SET salary=salary;

SQL> SELECT XID, STATUS FROM V$TRANSACTION;

SQL>SELECT dbms\_transaction.local\_transaction\_id FROM dual;

SQL> ROLLBACK;

SQL> SELECT XID FROM V$TRANSACTION;

SQL>SELECT dbms\_transaction.local\_transaction\_id FROM dual;

SQL> UPDATE hr.employees SET last\_name=last\_name;

SQL> SELECT XID, STATUS FROM V$TRANSACTION;

**Задание 2.**

|  |  |  |
| --- | --- | --- |
|  | Сессия | Пояснение |
| 1 | COMMIT; | Этот оператор завершает любую существующую транзакцию в сеансе. |
| 2 | SET TRANSACTION NAME 'sal\_update'; | Этот оператор начинает транзакцию и называет ее sal\_update. |
| 3 | SELECT last\_name, salary from HR.employees WHERE last\_name = 'Banda'; | Посмотреть з.п. у работника |
| 4 | UPDATE HR.employees  SET salary = 7000  WHERE last\_name = 'Banda'; | Изменить з.п. у работника |
| 5 | SAVEPOINT after\_banda\_sal; | Этот оператор создает точку сохранения с именем after\_banda\_sal, позволяя откатить изменения транзакции до этой точки. |
| 6 | SELECT last\_name, salary from HR.employees WHERE last\_name = 'Greene'; | Посмотреть з.п. у работника |
| 7 | UPDATE HR.employees  SET salary = 12000  WHERE last\_name = 'Greene'; | Изменить з.п. у работника |
| 8 | SELECT last\_name, salary from HR.employees WHERE last\_name = 'Greene'; | Посмотреть з.п. у работника |
| 9 | SAVEPOINT after\_greene\_sal; | создает точку сохранения с именем after\_greene\_sal |
| 10 | ROLLBACK TO SAVEPOINT  after\_banda\_sal; | Откатиться до точки сохранения с именем after\_banda\_sal |
| 11 | SELECT last\_name, salary from HR.employees WHERE last\_name in ('Greene', 'Banda'); | Посмотреть з.п. у работников |
| 12 | UPDATE HR.employees  SET salary = 11000  WHERE last\_name = 'Greene'; | ! точка сохранения для этой транзакции с именем sal\_update ! |
|  |  |  |
| 13 | SELECT last\_name, salary from HR.employees WHERE last\_name in ('Greene', 'Banda'); | Посмотреть з.п. у работников |
| 14 | SET TRANSACTION NAME 'sal\_update2';  UPDATE HR.employees  SET salary = 11999  WHERE last\_name = 'Greene';  UPDATE HR.employees  SET salary = 7999  WHERE last\_name = 'Banda';  COMMIT; | COMMIT; - гарантирует фиксацию изменений в таблице |
| 15 | ROLLBACK TO SAVEPOINT  after\_banda\_sal;  SELECT last\_name, salary from HR.employees WHERE last\_name in ('Greene', 'Banda'); | ! Проверьте ! |

**Блокировки**

- Препятствуют, чтобы несколько сеансов изменили те же самые данные одновременно

- Создаются автоматически на самом низком возможном уровне для данного оператора

- Не нарастают

**Задание 3 смоделируйте два параллельных сеанса работы Run SQL**

|  |  |
| --- | --- |
| Сессия 1  Транзакция 1 | Сессия 2  Транзакция 2 |
| SELECT employee\_id, last\_name, salary from HR.employees WHERE employee\_id in (100, 101); | |
| UPDATE HR.employees  SET salary = salary+100  WHERE employee\_id = 100; | UPDATE HR.employees  SET salary = salary\*1.1  WHERE employee\_id = 101; |
| SELECT employee\_id, last\_name, salary from HR.employees WHERE employee\_id in (100, 101); | |
| UPDATE HR.employees  SET salary = salary\*1.1  WHERE employee\_id = 101; | UPDATE hr.employees  SET salary = salary+100  WHERE employee\_id=100  AND salary>24150; |
| SELECT employee\_id, last\_name, salary from HR.employees WHERE employee\_id in (100, 101); | |

Прежде, чем база данных позволит сеансу изменять данные, сеанс должен сначала заблокировать данные, которые будут изменяться. Блокировка дает сеансу монопольный контроль над данными так, чтобы никакая другая транзакция не могла изменить заблокированные данные, пока блокировка не будет снята.

Транзакции могут заблокировать отдельные строки данных, несколько строк или даже целые таблицы. База данных Oracle поддерживает и ручные и автоматическые блокировки. Автоматически полученные блокировки всегда выбирают самый низкий возможный уровень блокировки, чтобы минимизировать потенциальные конфликты с другими транзакциями.

Есть много типов блокировок, используемых экземпляром Oracle, чтобы поддерживать внутреннюю непротиворечивость.

**Замки (locks, иначе — "блокировки")** в Oracle являются средством предотвращения нежелательного одновременного доступа к данным и внутренним структурам СУБД путем либо выстраивания процессов СУБД в очередь, либо прерывания операции с возвращением в программу ошибки доступа.

Замки имеют *тип* (type) и *режим наложения* (mode).

Oracle использует несколько десятков различных типов замков, однако для регулирования изменений данных в таблицах первоочередную важность имеют замки всего двух типов, необходимость наличия которых на объекте доступа и обусловливает возможность выполнения изменяющей операции DML:

* **TM**: блокировка транзакцией целой таблицы ("DML-блокировка");
* **TX**: блокировка транзакцией отдельных строк таблицы ("блокировка транзакции").

Режимов наложения замка на объект шесть:

|  |  |  |  |
| --- | --- | --- | --- |
| ***Режим блокировки*** | ***Код — краткое название*** | ***(Иногда) иное название*** | ***Старое название*** |
| NULL | 1 — NL |  |  |
| ROW SHARE | 2 — RS | SUB SHARED (SS) | SHARE UPDATE |
| ROW EXCLUSIVE | 3 — RX | SUB EXCLUSIVE (SX) |  |
| SHARE | 4 — S |  |  |
| SHARE ROW EXCLUSIVE | 5 — SRX | SHARED SUB EXCLUSIVE (SSX) |  |
| EXCLUSIVE | 6 — X |  |  |

Режимы с характеристикой EXCLUSIVE предполагают монопольное овладевание объектом, а режимы с характеристикой SHARE — долевое, допускающее одновременное нахождение на объекте нескольких однотипных замков (для режима SRX первенствует поведение EXCLUSIVE). Эти две характеристики ("виды блокировок") существуют в общем подходе к устройству транзакций. Слово ROW в названиях сообщает о действии замка на группу строк, а отсутствие этого слова — о действии на всю таблицу.

Замки типа **TX** могут налагаться СУБД в режимах RS и RX; типа **TM** — во всех.

Если транзакция пытается наложить на объект замок в режиме, несовместимом с режимом ранее наложенного на тот же объект замка, она либо (а) будет поставлена в очередь, либо (б) получит от СУБД сообщение об ошибке. Правила совместимости режимов наложения замков:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Режим претендующей блокировки*** | | | | | |
| *Режим имеющейся блокировки* | **NL** | **RS** | **RX** | **S** | **SRX** | **X** |
| **NL** | OK[1] | OK | OK | OK | OK | OK |
| **RS** | OK | (OK)[2] | (OK) | (OK) | (OK) | Несовм.[3] |
| **RX** | OK | (OK) | (OK) | Несовм. | Несовм. | Несовм. |
| **S** | OK | OK | Несовм. | OK | Несовм. | Несовм. |
| **SRX** | OK | OK | Несовм. | Несовм. | Несовм. | Несовм. |
| **X** | OK | Несовм. | Несовм. | Несовм. | Несовм. | Несовм. |

**Явное наложение замка типа TM на таблицу командой LOCK TABLE**

Используемая в Oracle техника замков позволяет предотвратить искажение данных изменяющими их транзакциями. Однако с точки зрения конкретной программы она может приводить к неожиданным "подвисаниям", смысл которых не всегда удается осознать конечному пользователю (для него программа просто "перестает работать"). Чтобы не терять контроль над работой программы, разработчик может побеспокоиться заранее и до действий по изменению строк явочным порядком наложить на объект требуемый замок. После этого транзакция сможет свободно изменять необходимые данные вплоть до своего завершения, не опасаясь быть заблокированной.

Предложение LOCK TABLE в Oracle позволяет наложить замок в требуемом режиме на таблицу:

LOCK TABLE имя\_таблицы

IN режим\_блокировки MODE

[NOWAIT | WAIT [ n ]]

**Задание 4 смоделируйте сеансы разрешения/запрещения блокировок**

*Примеры*:

LOCK TABLE dept IN SHARE MODE;

LOCK TABLE emp IN EXCLUSIVE MODE NOWAIT;

Блокировку таблицы в режимах SHARE и EXCLUSIVE можно специально запретить или же наоборот, разрешить.

*Пример*:

ALTER TABLE emp DISABLE TABLE LOCK;

LOCK TABLE emp IN EXCLUSIVE MODE NOWAIT;

ALTER TABLE emp ENABLE TABLE LOCK;

LOCK TABLE emp IN SHARE MODE;

**Явная блокировка групп строк в таблицах**

Недостатком блокирования данных таблицы командой LOCK TABLE может оказаться чересчур широкий охват строк, способный порождать по существу ненужные ожидания среди прочих транзакций. Зарезервировать для собственных нужд группы строк, а не все строки таблицы целиком, можно оператором SELECT, завершив его специальной фразой FOR UPDATE.

Предложение

SELECT ...

FOR UPDATE [список\_имен\_столбцов]

[NOWAIT | WAIT [ n ]]

не только вернет в программу запрашиваемые данные, но и автоматически наложит два замка, связывающие с действиями в этой транзакции группы строк, участвующих в формировании результата запроса:

* типа **TM** в режиме ROW SHARE на эту таблицу;
* типа **TX** в режиме EXCLUSIVE.

**Задание 5**

*Примеры*:

SELECT \* FROM emp WHERE deptno = 10 FOR UPDATE;

Здесь программа получает сведения о сотрудниках 10-го отдела и тут же резервирует за собой право свободно их изменять до конца транзакции.

Примечательно, что ограничений на однотабличность предложения SELECT не накладывается. Это позволяет одним предложением зарезервировать для последующих изменений группы строк сразу из нескольких таблиц, что важно, когда данные разных таблиц содержательно связаны:

SELECT empno, sal, comm

FROM emp INNER JOIN dept

USING ( deptno )

WHERE job = 'CLERK'

AND loc = 'NEW YORK'

FOR UPDATE

;

Здесь программа получает сведения о сотрудниках-клерках из Нью-Йорка и тут же блокирует строки об этих сотрудниках в таблице EMP и строкe об отделе этих сотрудников из таблицы DEPT. В подобных случаях снова может возникать проблема чересчур широкого блокирования.

**Групповая блокировка исключительно свободных строк**

С версии 8.1 существует разновидность групповой блокировки с обходом уже кем-то блокированных в данный момент строк. Запрос ниже выдаст в программу и одновременно пометит для текущей транзакции замками только те строки о сотрудниках отдела 10, которые свободны в настоящее время от несовместимых замков, принадлежащих другим транзакциям:

SELECT \* FROM emp WHERE deptno = 10 FOR UPDATE SKIP LOCKED;

Строки с существующими несовместимыми замками "наша" транзакция попросту проигнорирует, обойдет.

Такая возможность позволяет сократить и упростить программный код. Долгое время она была не документирована, но с версии 11 стала официальной.

**Задание 6**

Вывести список блокированных объектов с указанием идентификатора сессии, пользователя, блокирующего объект, схему-владельца и объект, типа объекта и типа блокировки.

SET LINESIZE 60

select lo.session\_id

, lo.oracle\_username locker

, lo.os\_user\_name,o.owner||'.'||o.object\_name object

, o.object\_type

, DECODE(lo.locked\_mode,

1, 'No Lock',

2, 'Row Share',

3, 'Row Exclusive',

4, 'Shared Table',

5, 'Shared Row Exclusive',

6, 'Exclusive') locked\_mode

from v$locked\_object lo, dba\_objects o

where lo.object\_id=o.object\_id;

или

SELECT B.Owner, B.Object\_Name, A.Oracle\_Username, A.OS\_User\_Name

FROM V$Locked\_Object A, All\_Objects B

WHERE A.Object\_ID = B.Object\_ID

или

select

c.owner,

c.object\_name,

c.object\_type,

b.sid,

b.serial#,

b.status,

b.osuser,

b.machine

from

v$locked\_object a ,

v$session b,

dba\_objects c

where

b.sid = a.session\_id

and

a.object\_id = c.object\_id;

Информация о заблокированных таблицах

select a.session\_id,a.oracle\_username, a.os\_user\_name, b.owner "OBJECT OWNER", b.object\_name,b.object\_type,a.locked\_mode from

(select object\_id, SESSION\_ID, ORACLE\_USERNAME, OS\_USER\_NAME, LOCKED\_MODE from v$locked\_object) a,

(select object\_id, owner, object\_name,object\_type from dba\_objects) b

where a.object\_id=b.object\_id;

### Задание 7 Узнать кто кого блокирует

select l1.sid, ' IS BLOCKING ', l2.sid

from v$lock l1, v$lock l2

where l1.block =1 and l2.request > 0

and l1.id1=l2.id1

and l1.id2=l2.id2;

или

select s1.username || '@' || s1.machine

|| ' ( SID=' || s1.sid || ' ) is blocking '

|| s2.username || '@' || s2.machine || ' ( SID=' || s2.sid || ' ) ' AS blocking\_status

from v$lock l1, v$session s1, v$lock l2, v$session s2

where s1.sid=l1.sid and s2.sid=l2.sid

and l1.BLOCK=1 and l2.request > 0

and l1.id1 = l2.id1

and l2.id2 = l2.id2 ;

1)Лок освобождается, только когда завершается транзакция.

2)Транзакция завершается, когда выполняются явные фиксация (commit) или откат (rollback) изменений, а также когда завершается открытая сессия (что по факту означает неявный откат).

**Задание 8**

**Конкурентный доступ к ресурсам и DBMS\_LOCK**

Сеансы работы с СУБД Oracle есть параллельно выполняющиеся процессы, работающие как с собственными, так и с общими ресурсами. Объекты БД, такие как таблицы, индексы, являются общими ресурсами. CУБД Oracle делает все возможное для того, чтобы конкурентный доступ к табличным данным был эффективным и максимально незаметным - прозрачным - для сеансов. Однако, в ряде случаев от программиста требуется явная блокировка ресурса на время работы с ним, и освобождение ресурса по окончании работы.

Таким ресурсом может быть, например, экземпляр некоторой сущности предметной области (сотрудник, элемент орг. структуры, и т.д.) Строки одной таблицы сеанс может зарезервировать для исключительного использования при помощи SELECT ... FOR UPDATE. Но в случае, когда необходимо заблокировать данные, распределенные по нескольким таблицам, лучшим решением будет специальное соглашение между конкурирующими процессами о доступе к ресурсу. А реализовать такое соглашение поможет пакет DBMS\_LOCK.

Ниже продемонстрирована секция кода, которая может выполняться одновременно не более, чем в одном сеансе. Вход в такую секцию охраняет исключительная блокировка - мьютекс (mutex, от **mut**ual **ex**clusive), и код в этой секции гарантированно защищен от параллельного выполнения в разных сеансах. Если работать с некоторым ресурсом только внутри таких секций, то одновременный доступ к ресурсу будет исключен.

Пакет DBMS\_LOCK предоставляет программисту API для работы с пользовательскими блокировками, которые обладают всеми возможностями системных блокировок СУБД Oracle. Так же, как и системные блокировки, пользовательские блокировки отображаются в динамическом вью v$lock.

Следующий блок PL/SQL создает исключительную блокировку при помощи пакета DBMS\_LOCK и выводит ее идентификатор:

SQL> SET SERVEROUTPUT ON

SQL> DECLARE

l\_lock VARCHAR2(30);

l\_status NUMBER;

BEGIN

dbms\_lock.allocate\_unique('myexclusivelock', l\_lock);

l\_status := dbms\_lock.request(l\_lock, dbms\_lock.x\_mode);

dbms\_output.put\_line(l\_status || ' : ' || l\_lock);

END;

/

Продублируйте (повторная попытка)

SQL> /

Процедура DBMS\_LOCK.ALLOCATE\_UNIQUE в строке 5 связывает придуманное программистом имя блокировки (первый агрумент) с числовым идентификатором (второй аргумент), назначенным системой. Далее для работы с блокировкой используется полученный числовой идентификатор.

Функция DBMS\_LOCK.REQUEST в строке 6 запрашивает у СУБД блокировку с данным идентификатором в исключительном режиме и получает ее, о чем говорит возвращаемый код 0. Как видим, повторная попытка получить ту же самую блокировку, выполнив PL/SQL блок еще раз, не увенчалась успехом: код возврата 4 означает, что у блокировки уже есть владелец. (Мьютексы в DBMS\_LOCK нереентерабельны.)

См. описание [функций и процедур пакета DBMS\_LOCK](http://docs.oracle.com/cd/E11882_01/appdev.112/e25788/d_lock.htm#i1002478) в официальной документации по СУБД Oracle.

Полученную нами блокировку можно увидеть во вью v$lock, где пользовательские блокировки имеют тип 'UL':

SQL> select \* from v$lock where sid = userenv('sid');

Снимем блокировку и убедимся, что она больше не отражается в v$lock:

SQL> DECLARE

l\_lock VARCHAR2(30);

l\_status NUMBER;

BEGIN

dbms\_lock.allocate\_unique('myexclusivelock', l\_lock);

l\_status := dbms\_lock.release(l\_lock);

dbms\_output.put\_line(l\_status || ' : ' || l\_lock);

END;

/

0 : 10737491871073749187101

PL/SQL procedure successfully completed

SQL> select \* from v$lock where sid = userenv('sid');

Процедура DBMS\_LOCK.ALLOCATE\_UNIQUE в строке 5 возвращает числовой идентификатор уже существующей блокировки по ее имени. Далее, функция DBMS\_LOCK.RELEASE в строке 6 снимает данную блокировку.

Теперь вернемся к сценарию, в котором некоторый код может одновременно выполняться только в одном сеансе. Этот сценарий реализует процедура critical:

CONNECT sys/masterkey.. AS SYSDBA;

GRANT execute ON sys.dbms\_lock TO system;

SQL> CREATE OR REPLACE PROCEDURE critical

IS

l\_lock VARCHAR2(30);

l\_status NUMBER;

BEGIN

dbms\_lock.allocate\_unique('myspeciallock', l\_lock);

l\_status := dbms\_lock.request(l\_lock, dbms\_lock.x\_mode);

IF l\_status = 0 THEN

dbms\_lock.sleep(5);

END IF;

l\_status := dbms\_lock.release(l\_lock);

END;

/

Procedure created

Чтобы создать процедуру critical, текущему пользователю должна быть явно (не через роль) предоставлена привилегия EXECUTE на пакет SYS.DBMS\_LOCK.

Для запуска процедуры critical в нескольких сеансах параллельно воспользуемся возможностями пакета DBMS\_SCHEDULER:

SQL> BEGIN

FOR i IN 1..3 LOOP

dbms\_scheduler.create\_job(

job\_name => 'critical\_' || i,

job\_type => 'STORED\_PROCEDURE',

job\_action => 'critical',

ENABLED => TRUE

);

END LOOP;

END;

/

PL/SQL procedure successfully completed

Дав процедуре время выполниться, проверю результат выполнения:

SQL> SELECT job\_name, status, actual\_start\_date, run\_duration

FROM all\_scheduler\_job\_run\_details

WHERE job\_name LIKE 'CRITICAL%';

Как видим, три задачи отработали, причем каждой следующей понадобилось на 5 секунд больше, чтобы выполниться. Это - время, в течение которого функция DBMS\_LOCK.REQUEST ожидала получения блокировки во втором и третьем сеансах.

Мы рассмотрели работу с исключительной (exclusive) блокировкой, но не менее важны разделяемые (shared) блокировк. Они позволяют параллельным процессам более эффективно работать с общим ресурсом. Исключительной блокировкой может владеть только один процесс, и, пока он ею владеет, ни один другой процесс не может получить доступ к заблокированному ресурсу. Разделяемой блокировкой могут владеть многие процессы, и, пока хотя бы один процесс ею владеет, ни один другой процесс не может получить исключительную блокировку данного ресурса.

Как это используется? Например, процессы, которым нужно только читать (но не изменять) некоторый ресрус, блокируют его в разделяемом режиме, что позволяет другим процессам также получать доступ к нему в разделяемом режиме, не дожидаясь его полного освобождения. Однако, процесс, которому необходимо изменить ресурс, для получения исключительной блокировки должен дождаться, когда с ресурса будут сняты все блокировки. И пока этот процесс будет изменять ресурс, ни один другой процесс не сможет получить доступ к ресурсу ни для чтения, ни для изменения.

Проиллюстрирую совместную работу разделяемой и исключительной блокировок. Для этого создам процедуру sharedmode, подобную critical, но, в отличие от нее, sharedmode запрашивает блокировку в разделяемом режиме:

SQL> CREATE OR REPLACE PROCEDURE sharedmode

IS

l\_lock VARCHAR2(30);

l\_status NUMBER;

BEGIN

dbms\_lock.allocate\_unique('myspeciallock', l\_lock);

l\_status := dbms\_lock.request(l\_lock, dbms\_lock.s\_mode);

IF l\_status = 0 THEN

dbms\_lock.sleep(5);

END IF;

l\_status := dbms\_lock.release(l\_lock);

END;

/

Procedure created

Теперь в трех различных сеансах параллельно запущу процедуры sharedmode, а в четвертом сеансе - процедуру critical. Три процесса, выполняющие sharedmode, получат разделяемые блокировки сразу, как только за ними обратятся, и выполнятся приблизительно за 5 секунд, перед завершением сняв блокировки. Процесс, выполняющий процедуру critical, получит исключительную блокировку только после того, как будут сняты разделяемые блокировки, и будет выполняться еще в течение 5 секунд, итого, около 10 секунд.

SQL> BEGIN

FOR i IN 1..3 LOOP

dbms\_scheduler.create\_job(

job\_name => 'special\_' || i,

job\_type => 'STORED\_PROCEDURE',

job\_action => 'sharedmode',

ENABLED => TRUE

);

END LOOP;

dbms\_scheduler.create\_job(

job\_name => 'special\_x',

job\_type => 'STORED\_PROCEDURE',

job\_action => 'critical',

ENABLED => TRUE

);

END;

/

PL/SQL procedure successfully completed

SQL> SELECT job\_name, status, actual\_start\_date, run\_duration

FROM all\_scheduler\_job\_run\_details

WHERE job\_name LIKE 'SPECIAL%'

ORDER BY job\_name;

Последний запрос полностью подтвердил ожидания.

На этом закончим эксперименты с DBMS\_LOCK:

SQL> DROP PROCEDURE critical;

Procedure dropped

SQL> DROP PROCEDURE sharedmode;

Procedure dropped

Помимо конкурентного доступа к общим ресурсам, для параллельных процессов всегда актуальна проблема коммуникации, то есть, возможность передачи сигналов и данных между процессами.