

# Contents

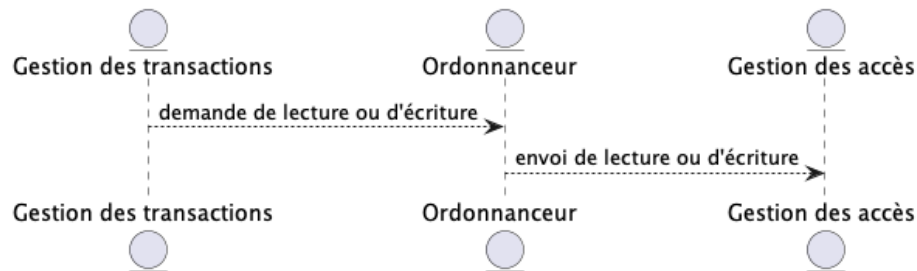
<b>1 Les concurrences</b>	<b>1</b>
1.1 Gestion de la concurrence . . . . .	1

## 1 Les concurrences

### 1.1 Gestion de la concurrence

Quand on exécute une requête, on récupère les données de la mémoire permanente, et sont stockées dans la mémoire vive.

Le gestionnaire de transactions est chargé de gérer les accès concurrents aux données.



Nous allons étudier :

- La sériabilisation d'une transaction
- La sériabilisation par accès concurrent
- Un système de verrous pour assurer la sériabilisation

#### 1.1.1 Executions sérialisables

##### 1.1.1.1 Executions concurrentes

**1.1.1.1.1 Définition** Une execution concurrente est une suite d'opération prises par une ou plusieurs transactions qui sont exécutées en même temps. on considère deux types d'opérations :

- **read(x, t)** : la transaction **t** lit la valeur de la variable **x**
- **write(x, t)** : la transaction **t** écrit la valeur de la variable **x**

##### 1.1.1.1.2 Exemple

$T_1$	$T_2$
$read(A, t)$	$read(A, s)$
$t = t + 100$	$s = 2 * s$

$T_1$	$T_2$
$write(A, t)$	$write(A, s)$
$read(B, t)$	$read(B, s)$
$t = t + 100$	$s = 2 * s$
$write(B, t)$	$write(B, s)$

**1.1.1.1.3 Définition** On dit qu'une execution est en série si toutes les opérations de chaque transaction sont exécutées de manière consécutive.

**1.1.1.1.4 Exemple**

$T_1$	$T_2$	$A$	$B$
$read(A, t)$	-	25	25
$t = t + 100$	-	...	...
$write(A, t)$	-	125	...
$read(B, t)$	-	...	...
$t = t + 100$	-	...	...
$write(B, t)$	-	...	125
-	$read(A, s)$	...	...
-	$s = 2 * s$	...	...
-	$write(A, s)$	250	...
-	$read(B, s)$	...	...
-	$s = 2 * s$	...	...
-	$write(B, s)$	...	250

On vient de faire l'exécution en scène de  $T_1$  et  $T_2$ , on la note  $(T_1, T_2)$ .

**1.1.1.1.5 Exercice** Exécutez  $(T_2, T_1)$  (la réciproque de  $(T_1, T_2)$ ).

$T_2$	$T_1$	$A$	$B$
$read(A, s)$	-	25	25
$s = 2 * s$	-	...	...
$write(A, s)$	-	50	...
$read(B, s)$	-	...	...
$s = 2 * s$	-	...	...
$write(B, s)$	-	...	50
-	$read(A, t)$	...	...
-	$t = t + 100$	...	...
-	$write(A, t)$	150	...
-	$read(B, t)$	...	...
-	$t = t + 100$	...	...
-	$write(B, t)$	...	150

$T_2$	$T_1$	$A$	$B$
-------	-------	-----	-----

On observe que l'état de la base de données est différent entre les deux exécutions.

**1.1.1.2 Executions sérialisables** Quelles sont les exécutions qui garantissent la préservation de la cohérence des données ?

**1.1.1.2.1 Définition** Une execution  $E$  est sérialisable s'il existe une exécution en série  $E'$  qui est équivalente à  $E$  i.e. qu'elles ont le même effet sur toutes les BDD.

**1.1.1.2.2 Exemple** Une exemple de transaction sérialisable est :

$T_1$	$T_2$	$A$	$B$
$read(A, t)$	-	25	25
$t = t + 100$	-	...	...
$write(A, t)$	-	125	...
...	$read(A, s)$	...	...
...	$s = 2 * s$	...	...
...	$write(A, s)$	250	...
$read(B, t)$	-	...	...
$t = t + 100$	-	...	...
$write(B, t)$	-	...	125
...	$read(B, s)$	...	...
...	$s = 2 * s$	...	...
...	$write(B, s)$	...	250

Cette exécution n'est pas en série mais elle est équivalente à l'exécution en série  $(T_1, T_2)$ . En effet,  $A$  est transformé en  $2 \times (A + 100)$  et pareil pour  $B$ .

**1.1.1.2.3 Exemple** Un exemple de transaction non sérialisable est :

$T_1$	$T_2$	$A$	$B$
$read(A, t)$	-	25	25
$t = t + 100$	-	...	...
$write(A, t)$	-	125	...
...	$read(A, s)$	...	...
...	$s = 2 * s$	...	...
...	$write(A, s)$	250	...
...	$read(B, s)$	...	...

$T_1$	$T_2$	$A$	$B$
...	$s = 2 * s$	...	...
...	$write(B, s)$	...	50
$read(B, t)$	-	...	...
$t = t + 100$	-	...	...
$write(B, t)$	-	...	150

**1.1.1.2.4 Exemple** Il est compliqué de savoir si une exécution est sérialisable ou non. Par exemple, l'exécution suivante est-elle sérialisable ?

$T_1$	$T_2$	$A$	$B$
$read(A, t)$	-	25	25
$t = t + 100$	-	...	...
$write(A, t)$	-	125	...
...	$read(A, s)$	...	...
...	$s = s + 200$	...	...
...	$write(A, s)$	325	...
...	$read(B, s)$	...	...
...	$s = s + 200$	...	...
...	$write(B, s)$	...	225
$read(B, t)$	-	...	...
$t = t + 100$	-	...	...
$write(B, t)$	-	...	325

Ici on est sérialisable et cela vient du fait que l'on effectue uniquement des additions dont l'ordre ne compte pas.

Par la suite, on considère toujours le cas où l'ordre compte et on fera abstraction de la modification des valeurs.

## 1.1.2 Sériabilité par conflits

On utilise les notations suivantes :

- $r_i(x)$  : la transaction  $T_i$  lit la valeur  $x$  de la BDD
- $w_i(x)$  : la transaction  $T_i$  écrit la valeur  $x$  de la BDD

La transaction sérialisable précédente peut s'écrire :

$$E = \{r_1(A), w_1(A), r_2(A), w_2(A), r_1(B), w_1(B), r_2(B), w_2(B)\}$$

### 1.1.2.1 Conflit

**1.1.2.1.1 Définition** On parle de conflit entre deux opérations consécutives d'une exécution lorsque leur ordre ne peut pas être inversé sans changer le résultat de l'exécution.

On sait qu'il n'y a pas de conflit entre les paires d'opérations suivantes issues des transactions  $T_i$  et  $T_j$  :

1.  $r_i(A)$  et  $r_j(A)$
2.  $r_i(A)$  et  $w_j(A)$  avec  $i \neq j$
3.  $w_i(A)$  et  $r_j(A)$  avec  $i \neq j$

Par contre, il y a un conflit entre les paires d'opérations suivantes issues des transactions  $T_i$  et  $T_j$  :

1. Deux opérations qui sont dans la même transaction  $T_i$
2.  $w_i(A)$  et  $w_j(A)$  avec  $i \neq j$
3.  $r_i(A)$  et  $w_j(A)$  ou  $w_i(A)$  et  $r_j(A)$

Pour résumer, on peut exécuter deux opérations consécutives issues de deux transactions différentes si :

- Elles sont sur des éléments différents
- Elles sont deux lectures

**1.1.2.1.2 Exemple**  $E = \{r_1(A), w_1(A), r_2(B), w_2(B)\}$  est une exécution de  $T_1$  et  $T_2$ .

On a une exécution équivalente :

- $E = \{r_1(A), r_2(B), w_1(A), w_2(B)\}$

**1.1.2.1.3 Définition** On dit que deux exécutions sont équivalentes par conflit si on peut passer de l'une à l'autre en inversant des opérations consécutives sans conflit.

**1.1.2.2 Propriété** Si une exécution est sérialisable par conflit alors elle est sérialisable.