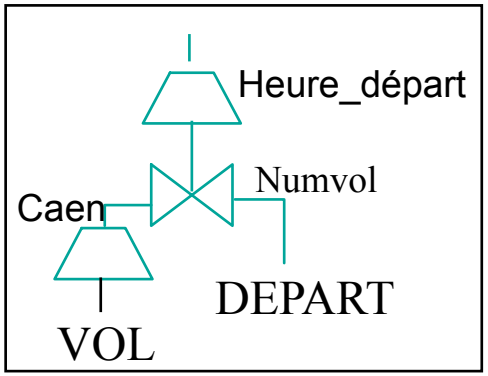
SQL Avancé

SQL (Structured Query Language)

Division

Les jointures : internes & externes

Généalogie du langage SQL



{V.Heure_Départ/ Vol (V)
et V.Ville_arrivée = 'Caen'
et ∃ D / Départ (D)
et D.Numvol = V.Numvol
et D.Date = '19-12-95'}

Algèbre relationnelle

Autre

Calcul de tuples

SQL

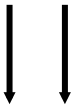
Exemple de référence

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700



EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
...		
102	90	Executive
205	110	Accounting
206	110	Accounting

LA DIVISION SQL

Quels sont les employés qui ont travaillé dans tous les départements ?
<=> Quels sont les employés tels que, quel que soit le département, ils y ont travaillé?

<=> Quels sont les employés tels que, quel que soit le département, il existe un emploi assuré par cet employé dans ce département?

<=> Quels sont les employés tels qu' il n'existe pas de département tel qu' il n'existe pas d'emploi assuré par cet employé dans ce département ?

```
SELECT DISTINCT E1.employee_id
FROM EMPLOYEES E1
WHERE NOT EXISTS (  SELECT *
                    FROM DEPARTMENTS D
                    WHERE NOT EXISTS (  SELECT *
                                        FROM EMPLOYEES E2
                                        WHERE  E1.employee_id = E2.employee_id
                                        AND E2.department_id = D.department_id));
```

LA JOINTURE

- 2 principales méthodes:

  Quels sont les noms des employés qui travaillent au département Marketing ?

- Prédicative

```
SELECT E.employee_id, E.last_name
FROM EMPLOYEES E, DEPARTMENTS D
WHERE E.department_id = D.department_id
AND  D.department_name = ' Marketing';
```

- Imbrication

```
SELECT E.employee_id, E.last_name
FROM EMPLOYEES E
WHERE E.department_id  IN (   SELECT D.department_id
                              FROM DEPARTMENTS D
                              WHERE D.department_name = ' Marketing');
```

Remarque

Il y a de nombreuses autres façons d'exprimer une jointure.

```
SELECT E.employee_id, E.last_name
FROM EMPLOYEES E
WHERE EXISTS  (  SELECT *
                 FROM DEPARTMENTS D
                 WHERE E.department_id = D.department_id
                 AND D.department_name = ’Marketing');
```

Les jointures

- Les jointures conformes au standard SQL 1999 comprennent :
  - Le produit cartésien (**CROSS JOIN**)
  - Les jointures internes :
    - La jointure naturelle (**NATURAL JOIN**)
    - La jointure avec la clause (**JOIN..USING, JOIN..ON**)
  - Les jointures externes : gauche, droite, totale (**LEFT|RIGHT|FULL OUTER JOIN**)

```
SELECT     table1.column, table2.column
FROM       table1
[CROSS JOIN table2] |
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2 ON (table1.column_name = table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2
           ON (table1.column_name = table2.column_name)] ;
```

Produit Cartésien

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
⇔
SELECT last_name, department_name
FROM   employees, departments ;
```

LAST_NAME	DEPARTMENT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
Hunold	Administration

...

160 rows selected.

La jointure naturelle : equi-jointure sur tous les attributs communs

```
SELECT department_id, department_name,location_id,city
FROM   departments
NATURAL JOIN locations ;
```

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

8 rows selected.

Jointure sur un seul attribut : equi-jointure

```
SELECT employees.employee_id, employees.last_name,
       departments.location_id, department_id
FROM   employees JOIN departments USING (department id);
⇔
SELECT e.employee_id, e.last_name,
       d.location_id, d.department_id
FROM   employees e, departments d
WHERE  e.department_id= d.department_id;
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
200	Whalen	1700	10
201	Hartstein	1800	20
202	Fay	1800	20
124	Mourgos	1500	50
141	Rajs	1500	50
142	Davies	1500	50
144	Vargas	1500	50
143	Matos	1500	50

...

19 rows selected.

Equi or Inequi Jointure

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department id, d.location id
FROM   employees e JOIN departments d
                        ON(e.department_id = d.department_id);
⇔
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500

...

19 rows selected.

Sélection-Projection-Jointure

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
                        ON (e.department_id = d.department_id)
                        AND    e.manager_id = 149 ;
⇔
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e, departments d
WHERE  e.department_id = d.department_id
AND    e.manager_id = 149 ;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
174	Abel	80	80	2500
176	Taylor	80	80	2500

Chemin de jointure : jointure n-aire

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
       ON d.department_id = e.department_id
JOIN   locations l
       ON d.location_id = l.location_id;
⇔
SELECT employee_id, city, department_name
FROM   employees e, departments d, locations l
WHERE   d.department_id = e.department_id
AND     d.location_id = l.location_id;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
124	South San Francisco	Shipping
141	South San Francisco	Shipping
142	South San Francisco	Shipping
143	South San Francisco	Shipping
144	South San Francisco	Shipping

...

19 rows selected.

Inequi-jointure (1)

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

...

20 rows selected.

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

Le salaire des employés doit être compris entre le salaire minimum et le plus haut salaire dans la table Job_Grades

Inequi-jointure (2)

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary BETWEEN j.lowest_sal AND j.highest_sal;
⇔
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

...

20 rows selected.

La jointure externe & valeurs nulles

- Dans SQL1999, la jointure entre 2 tables ne retourne que des n-uplets qui satisfont le prédicat de jointure
  - Que se passe-t-il si l'on désire conserver tous les départements même ceux qui n'ont pas encore d'employés?

- Les jointures externes sont utilisées
  - pour exprimer des requêtes OLAP,
  - pour calculer des requêtes imbriquées (Not Exists, Not In, All,…)

- **LEFT ou RIGHT OUTER JOIN**
  - Une jointure entre 2 tables, qui retournent en résultat à la fois les n-uplets qui satisfont le prédicat de joitnure, mais également les tuples de la table de gauche ou de droite qui ne satisfont pas le prédicat
- **FULL OUTER JOIN**
  - Une jointure de 2 tables qui retournent à la fois les n-uplets qui satisfont le prédicat de jointure et également ceux de la table de gauche et de droite. On appelle cette jointure :

Jointure Externe

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

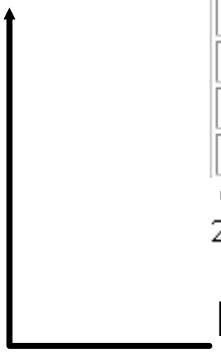
8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

...

20 rows selected.



Il n'y a pas d'employé dans le département 190

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
                   ON (e.department_id = d.department_id) ;
⇔
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id = d.department_id (+) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		

20 rows selected.

RIGHT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
                   ON(e.department_id = d.department_id) ;
⇔
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id (+) = d.department_id ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
Davies	50	Shipping
...		
Kochhar	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
	190	Contracting

20 rows selected.

FULL OUTER JOIN

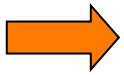
```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
                   ON(e.department_id = d.department_id) ;
⇔
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id = d.department_id (+) ;
UNION
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id (+) = d.department_id ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		
	190	Contracting

21 rows selected.

LEFT OUTER JOIN & NOT EXISTS

```
SELECT d.department_name
FROM Departments d
WHERE NOT EXISTS
      (SELECT * FROM Employees e
       WHERE d.deparment_id=e.department_id)
```



```
SELECT d.department_name
FROM Departments d
     LEFT OUTER JOIN
     Employees e
     ON (d.department_id = e.department_id)
WHERE e.department_id IS NULL
```

Opérateurs logiques avec valeurs nulles

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

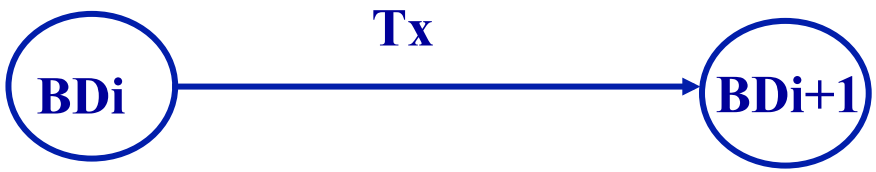
OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Transactions

Transactions

- Unité de traitement faisant passer la base de données d'un état cohérent à un nouvel état cohérent

- Changement d'état atomique de la base de données
  - non observable de l'extérieur



Atomicité

- Une transaction est un atome selon 4 points de vue :
  - Sécurité
  - Intégrité
  - Concurrence
  - Fiabilité

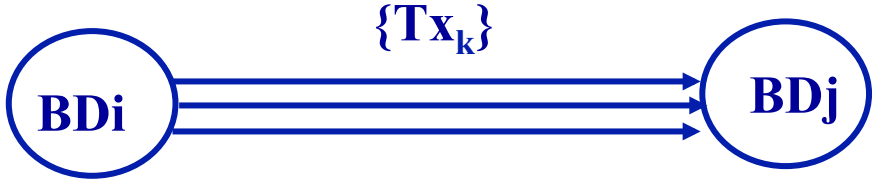
Sécurité / Intégrité

- Sécurité
  - Si une violation de sécurité est détectée sur une donnée quelconque, la transaction entière doit être abandonnée

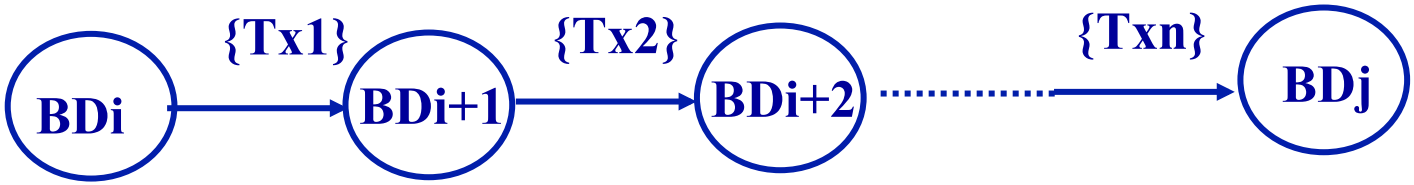
- Intégrité
  - Pour être acceptée, une transaction doit vérifier les contraintes d'intégrité sur toutes les données concernées.

Concurrence

- A un instant donné, un ensemble de transactions s'exécute en parallèle



- L'exécution concurrente doit être équivalente à celle d'une suite de transactions atomiques sans concurrence (sérialisabilité)



- Le même ordre d'exécution apparent doit être observé pour tous les éléments de données partagées

Fiabilité

- Une transaction doit réussir en totalité ou être abandonnée en totalité
- Un état intermédiaire d'une transaction ne peut subsister après une panne
- Des sauvegardes/restauration doivent permettre de reconstruire la base en cas de perte du support

ACID-ité

- Atomicité
  - transaction entièrement exécutée ou pas du tout
- Cohérence
  - respect des règles d'intégrité
- Isolation
  - mises à jour non validées non vues
- Durabilité
  - après validation, les mises à jour ne peuvent être perdues

⇒ Les SGBD actuels assurent l'ACID-ité des transactions

Modèles de transactions

- Modèle séquentiel simple

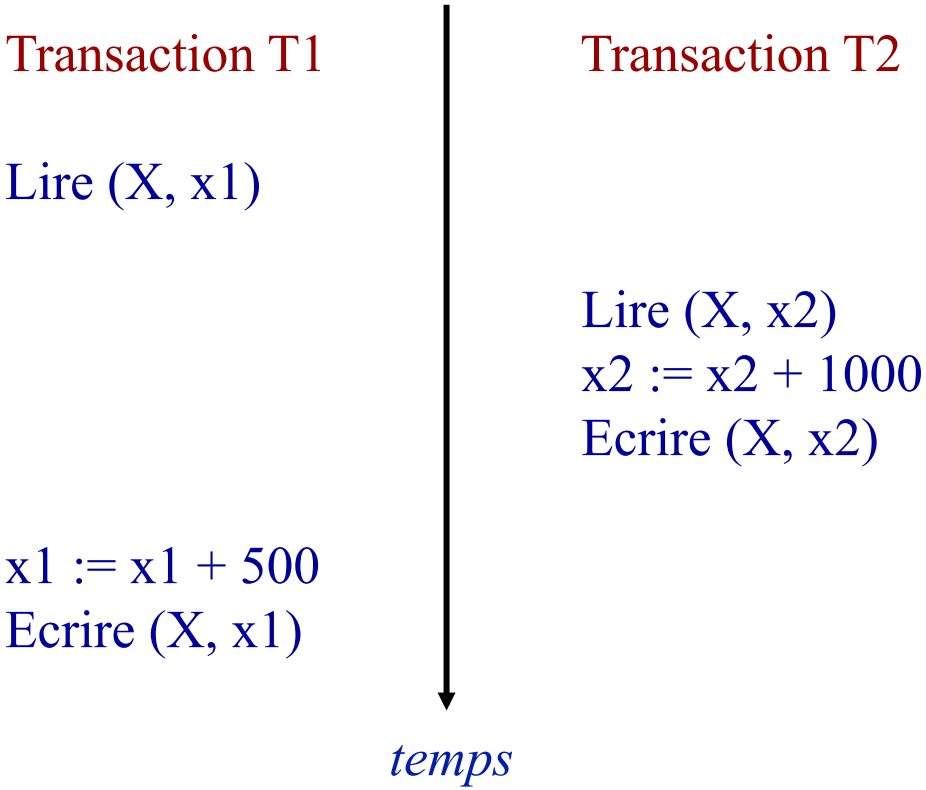
Début de transaction
    requête 1 sur la BD
    requête 2 sur la BD
    si erreur alors abandon
    requête 3
    .....
    Validation
Fin transaction

Contrôle de concurrence

- Objectif :
  - rendre invisible aux utilisateurs le partage simultané des données
- Nécessité de protocoles permettant de synchroniser les mises à jour pour éviter
  - les pertes d'opérations
  - l'observation d'incohérences

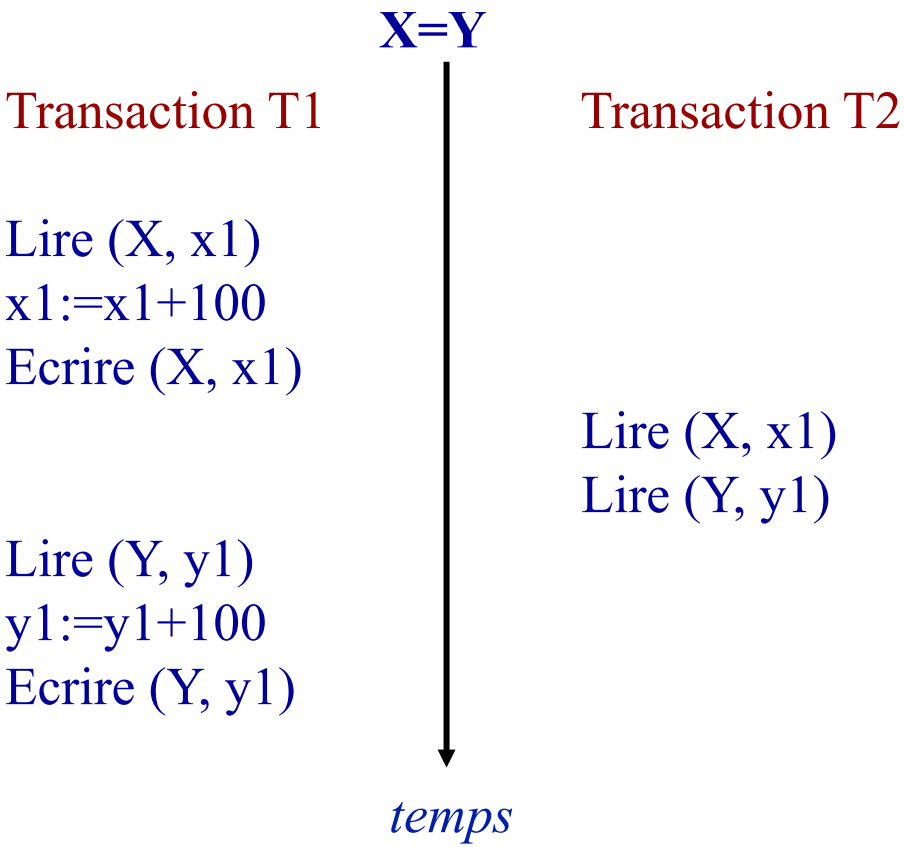
EXEMPLE

- **Perte d'opérations**



EXEMPLE

- **Observation d'incohérences**



Définitions

- Granule
  - une base de données est constituée d'un ensemble d'objets élémentaires qui sont les granules de l'algorithme de contrôle de concurrence
    - relations, tuples ou parties de tuples...

- Action atomique
  - Une action d'une transaction sur un granule est dite atomique si le système garantit qu'elle ne sera pas interrompue par une action d'une autre transaction
    - une transaction est une séquence d'actions
      - Lecture
      - Ecriture

....Définitions

- Exécution en série
  - si chaque transaction est exécutée après la fin de la précédente, l'exécution globale est dite en série
- But d'un algorithme de contrôle de concurrence :
  - faire apparaître l'exécution des transactions comme une exécution en série sans concurrence

Actions permutables

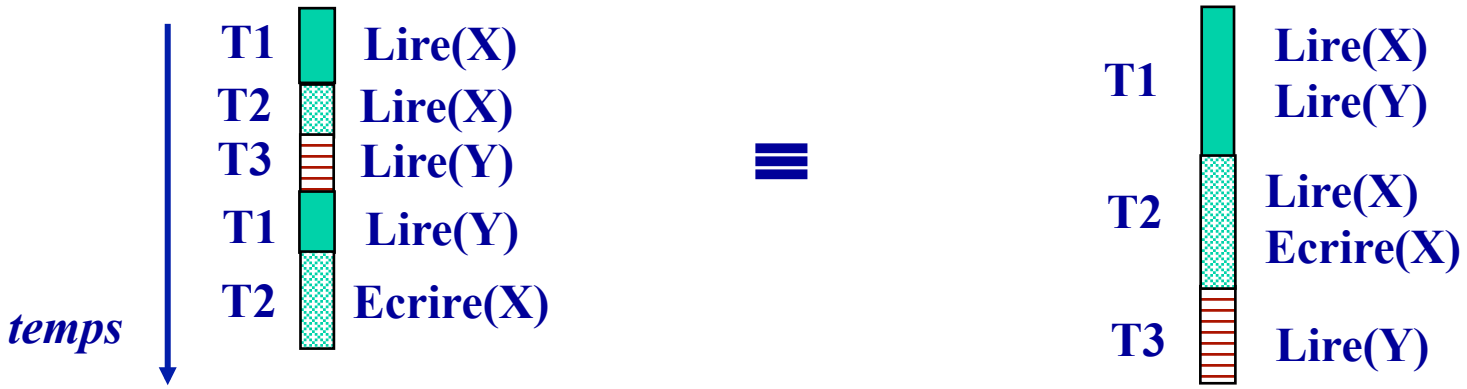
- Deux actions atomiques de deux transactions différentes sont permutables dans une exécution si leur résultat ne dépend pas de leur ordre d'exécution
  - deux actions concernant deux granules différents sont toujours permutables

- Matrice de compatibilité :

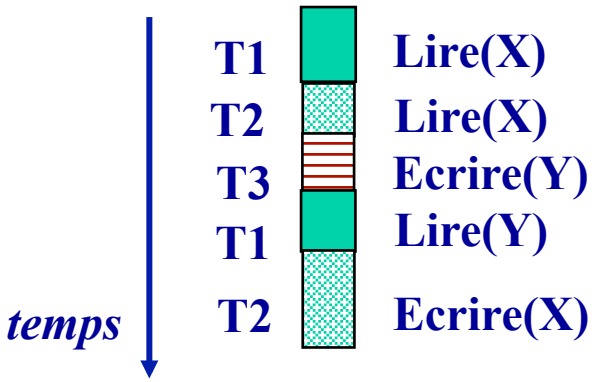
	Lecture	Ecriture
Lecture	OUI	NON
Ecriture	NON	NON

Exécution sérialisable

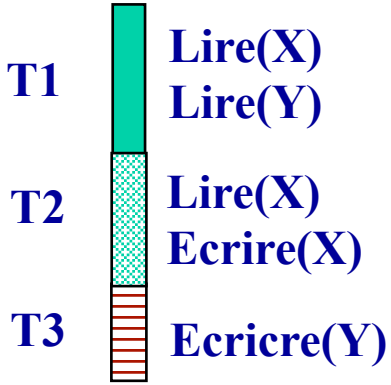
- Une exécution parallèle des transactions T1, T2...., Tn est dite sérialisable si son résultat est équivalent à une exécution en série des mêmes transactions.
  - Si une exécution de transactions est transformable par commutations successives en une exécution en série, alors elle est sérialisable



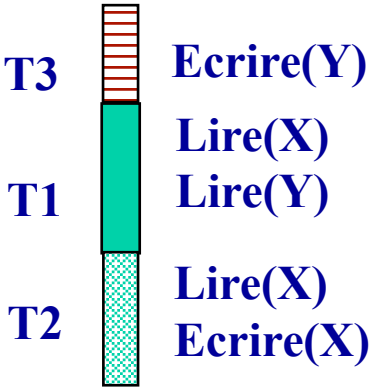
Autre exemple d'exécution sérialisable



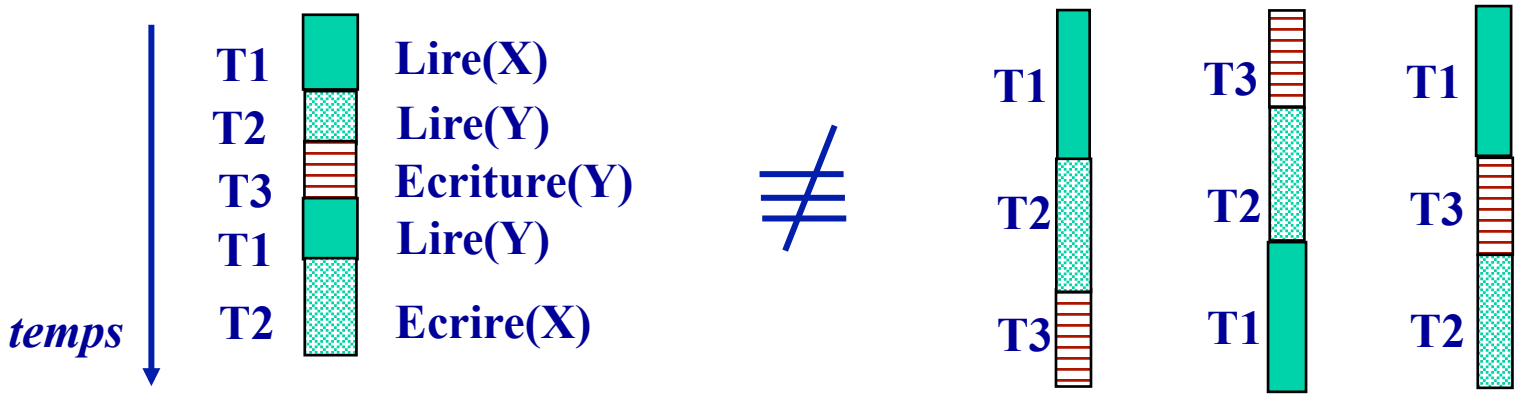
≠



≡



Exemple d'exécution non sérialisable



Relation de précédence

- Sur chaque granule, deux actions non permutables impliquent une relation de précédence entre les transactions correspondantes

- Définition
  - Sur un même granule, si une transaction T1 applique une action X avant qu'une transaction T2 applique une action Y et si X et Y ne sont pas permutables, alors une relation de précédence est établie de T1 vers T2.

Graphe de précédence

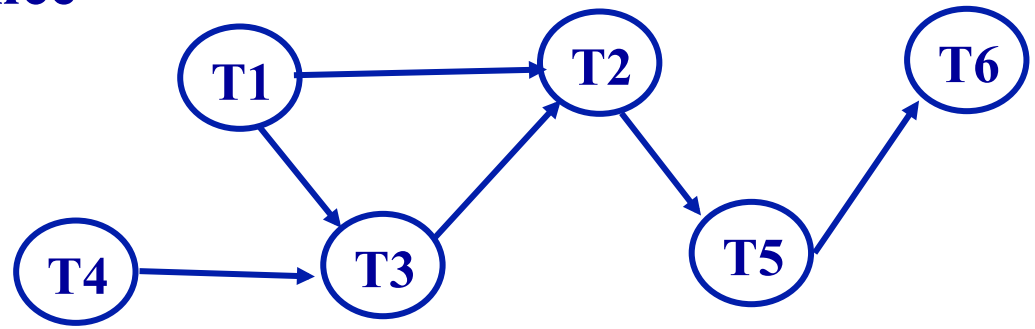
- La relation de précédence peut être représentée par son graphe
- Si le graphe de précédence d'une exécution est acyclique, alors l'exécution est sérialisable.

EXEMPLE

Ordre d'exécution

G1 : E1, E2, E5, L6
G2 : L4, E3, L2
G3 : L4, L1, L3, E3

Graphe de précédence



◊ On ne considère pas l'ordre des opérations à l'intérieur d'une même transaction

Méthodes de contrôle de concurrence

- Méthode par estampillage :
  - Approche curative consistant à contrôler l'ordre des accès en estampillant les objets et en annulant les transactions hors séquence (en retard)

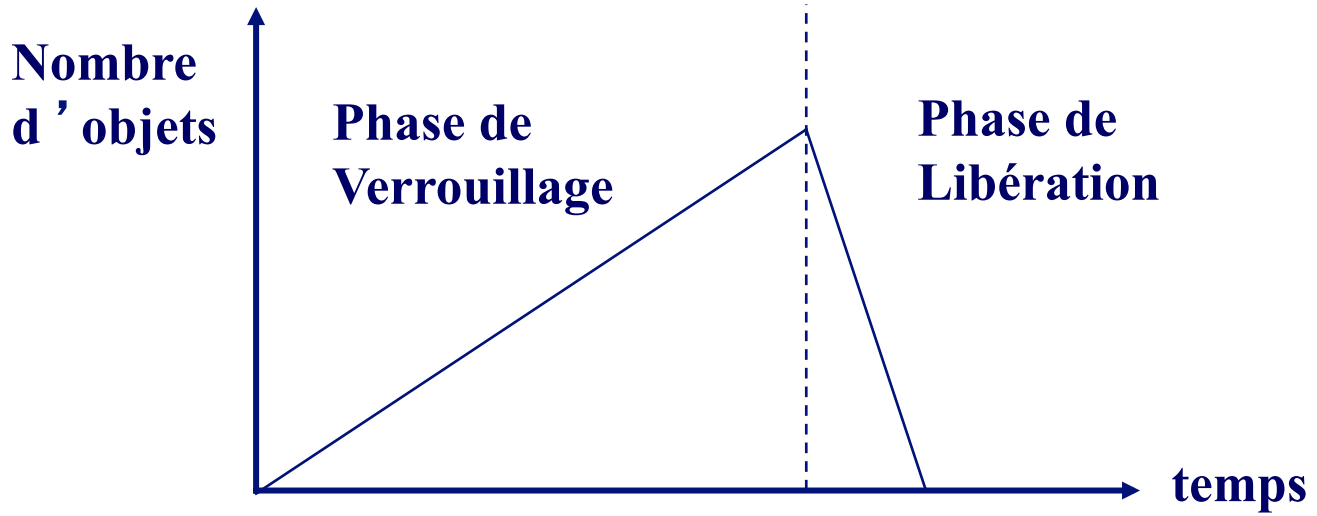
- Méthode par verrouillage
  - Approche préventive consistant à empêcher les conflits en bloquant les objets pendant la transaction (verrou en lecture ou en écriture)

Principes du verrouillage (1)

- But : laisser s'exécuter simultanément uniquement les actions permutables
- Lorsqu'une transaction T1 accède à un granule, elle le verrouille jusqu'à ce qu'elle termine toute ses actions
- Si une deuxième transaction T2 tente un accès conflictuel, elle est mise en attente

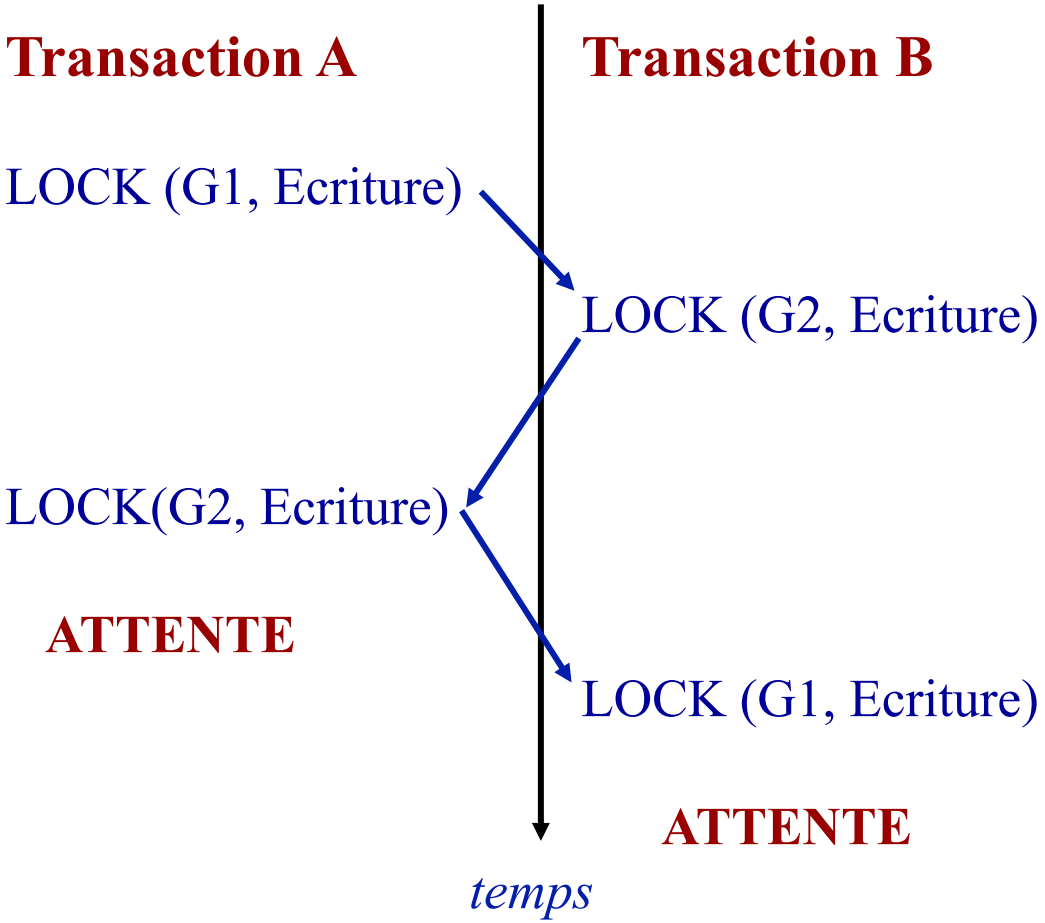
Principes du verrouillage (2)

- Une transaction est composée de deux phases
  - 2 PL (Two Phase Locking)



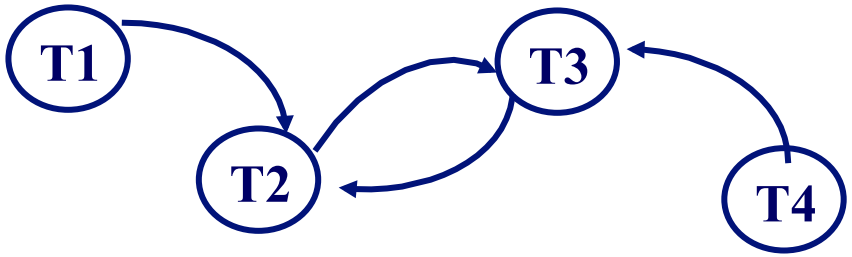
  - Le déverrouillage s' effectue en fin de transaction (commit) pour que les mises à jour non validées ne soient pas visibles

Problème du verrou mortel (inter-bloquage)



Graphe des attentes

- T_i attend T_j si T_i attend le verrouillage d'un granule déjà verrouillé par T_j



- Un ensemble de verrous ne provoque pas de verrou mortel si le graphe des attentes est acyclique