Chapter 4

Concurrency Control Techniques Part III

Outline

Databases Concurrency Control

1. Purpose of Concurrency Control
2. Two-Phase locking technique
3. Timestamp Ordering technique
4. Multiversion technique
5. Optimistic or Validation technique
6. Multiple Granularity Locking
7. Phantom problem
8. Transaction Support in SQL2
9. Oracle concurrency control

VI- Multiple Granularity Locking

Granularity of data items

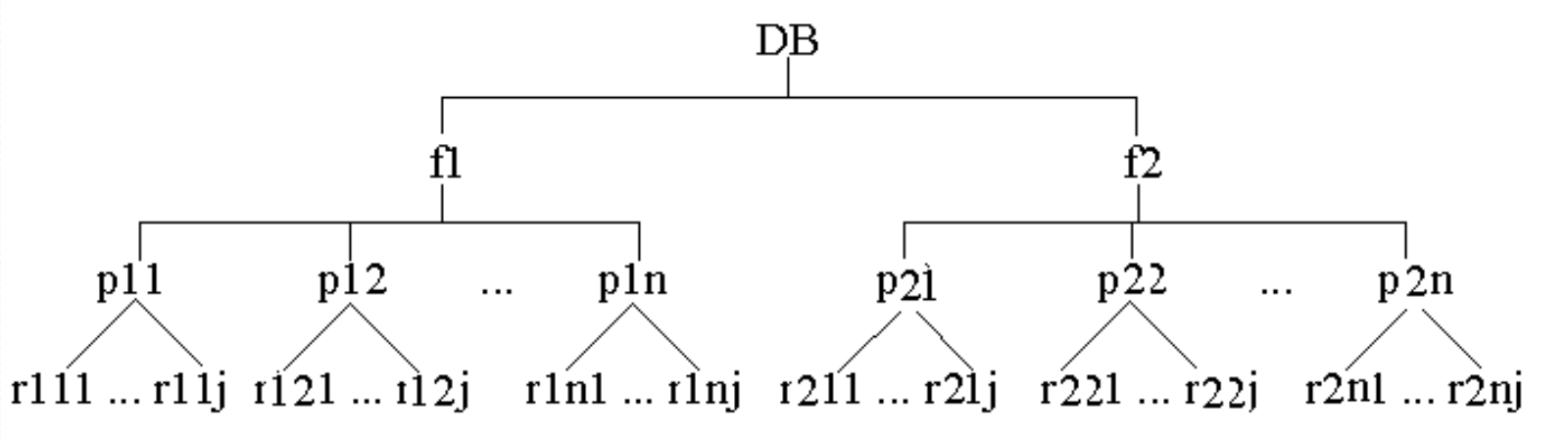
The lockable unit of data defines its granularity. Granularity can be **coarse** (entire database) or it can be **fine** (a tuple or an attribute of a relation). Thus, the degree of concurrency is low for coarse granularity and high for fine granularity. Example of data item granularity:

1. A field of a database record (an attribute of a tuple).
2. A database record (a tuple of a relation).
3. A disk block.
4. An entire file.
5. The entire database.

Multiple Granularity Locking

Granularity of data items

The following diagram illustrates a hierarchy of granularity from coarse (database) to fine (record).



Multiple Granularity Locking

Granularity of data items

- **Large data item size ➔ lower degree of concurrency**
  - For example, if the data item size is a disk block, a transaction T that needs to lock a record *B* must lock the whole disk block *X* that contains *B* because a lock is associated with the whole data item (block). Now, if another transaction S wants to lock a different record *C* that happens to reside in the same block *X* in a conflicting lock mode, it is forced to wait.
- **Small data item size ➔ more active locks or timestamps to be handled**
  - The smaller the data item size is, the system will have a larger number of active locks to be handled by the lock manager. More lock and unlock operations will be performed, causing a higher overhead. In addition, more storage space will be required for the lock table.

Multiple Granularity Locking

Granularity of data items

The best item size should depend on the type of the transaction involved ➔ multiple granularity

- If a typical transaction accesses a small number of records, it is advantageous to have the data item granularity be **one record.**
- On the other hand, if a transaction typically accesses many records in the same file, it may be better to have **block or file** granularity so that the transaction will consider all those records as one.

Problem : T1 read locks a row in a page, T2 wants to write lock the complete table. The lock manager will have to check the lock status of all the rows of the table ➔ inefficiency

➔ Intention locks

Multiple Granularity Locking

So, to manage such hierarchy, in addition to read and write, three locking modes, called intention lock modes are defined:

- **Intention-shared (IS):** indicates that a shared lock(s) will be requested on some descendent node(s) : the query will read some descendent objects.
- **Intention-exclusive (IX):** indicates that an exclusive lock(s) will be requested on some descendent node(s) : the query will modify some descendent objects.
- **Shared-intention-exclusive (SIX):** indicates that the current node is locked in shared mode but an exclusive lock(s) will be requested on some descendent node(s): the query will read/modify some descendent objects.

Locking in SIX mode

- Soit la relation :
  - livre(isbn, titre, catégorie, prix)

  et la transaction *T* :
  - UPDATE livre
    SET prix := 1.25 * prix
    WHERE catégorie = 'roman';
- Pour réaliser cette mise à jour, la transaction *T* doit demander un accès exclusif à la table livre pour empêcher toute modification de cette table durant l'opération de mise à jour.
- Si *T* verrouille la table livre en mode exclusif alors elle ne pourra pas lire les n-uplets de livre pour sélectionner ceux qui doivent être mis à jour.
- Par contre, si *T* verrouille la table livre en mode SIX (accès partagé exclusif), alors elle et elle seule pourra verrouiller les lignes de cette table en mode S pour les lire afin de sélectionner ceux qui sont des romans, qui eux seront verrouillés en mode X afin d'être modifiés.

Summary

- **Two-phase locking**:
  - Use for simple applications where a single granularity is acceptable.
  - If there are large read-only transactions, multiversion protocols would do better.
- **Two-phase locking with multiple granularity locking**:
  - Use for an application mix where some applications access individual records and others access whole relations or substantial parts thereof.
- **Timestamp ordering**:
  - Use if the application demands a concurrent execution that is equivalent to a particular serial ordering. But conflicts are handled by roll-back of transactions rather than waiting.
  - Not suitable if there are long read-only transactions, since they will starve. Deadlocks are absent.

Summary

- **Validation/optimistic**:
  - If the probability that two concurrently executing transactions conflict is low, this protocol can be used advantageously to get better concurrency and good response times with low overheads.
- **Multiversion timestamp ordering**:
  - Use if timestamp ordering is appropriate but it is desirable for read requests to never wait.
  - Shares the other disadvantages of the timestamp ordering protocol.
- **Multiversion two-phase locking**:
  - This protocol allows read-only transactions to always commit without ever waiting. Update transactions follow 2PL, thus allowing recoverable schedules with conflicts solved by waiting rather than roll-back.
  - But the problem of deadlocks comes back.
  - Keeping multiple versions adds space and time overheads though; therefore plain 2PL may be preferable in low conflict situations.

What do real systems do?

Oracle, Microsoft SQL Server etc use Strict 2PL or variants.

All these systems support multiple granularity locking with support for table, page and row level locks. They all deal with deadlock using waits-for graphs.

IX- Oracle Transaction Types

Type	Description
Data manipulation language (DML)	Consists of any number of DML statements that the Oracle server treats as a single entity or a logical unit of work. Like Insert, Update, Delete.
Data definition language (DDL)	Consists of only one DDL statement> Like create, alter, drop.
Data control language (DCL)	Consists of only one DCL statement Like grant/Revoke access.

Transaction boundaries

A transaction begins with the first executable SQL statement.

- A transaction ends with one of the following events:
  - A COMMIT or ROLLBACK statement is issued
  - A DDL or DCL statement executes (automatic commit)
  - The user exits *i*SQL*Plus
  - The system crashes

Advantages of COMMIT and ROLLBACK

With COMMIT and ROLLBACK statements, you can:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically related operations

State of the Data Before COMMIT or ROLLBACK

- The previous state of the data ***can be recovered***.
- The current user ***can review*** the results of the DML operations by using the SELECT statement.
- Other users ***can not view*** the results of the DML statements by the current user.
- The affected rows ***are locked***
- Other users ***cannot change*** the data within the affected rows.

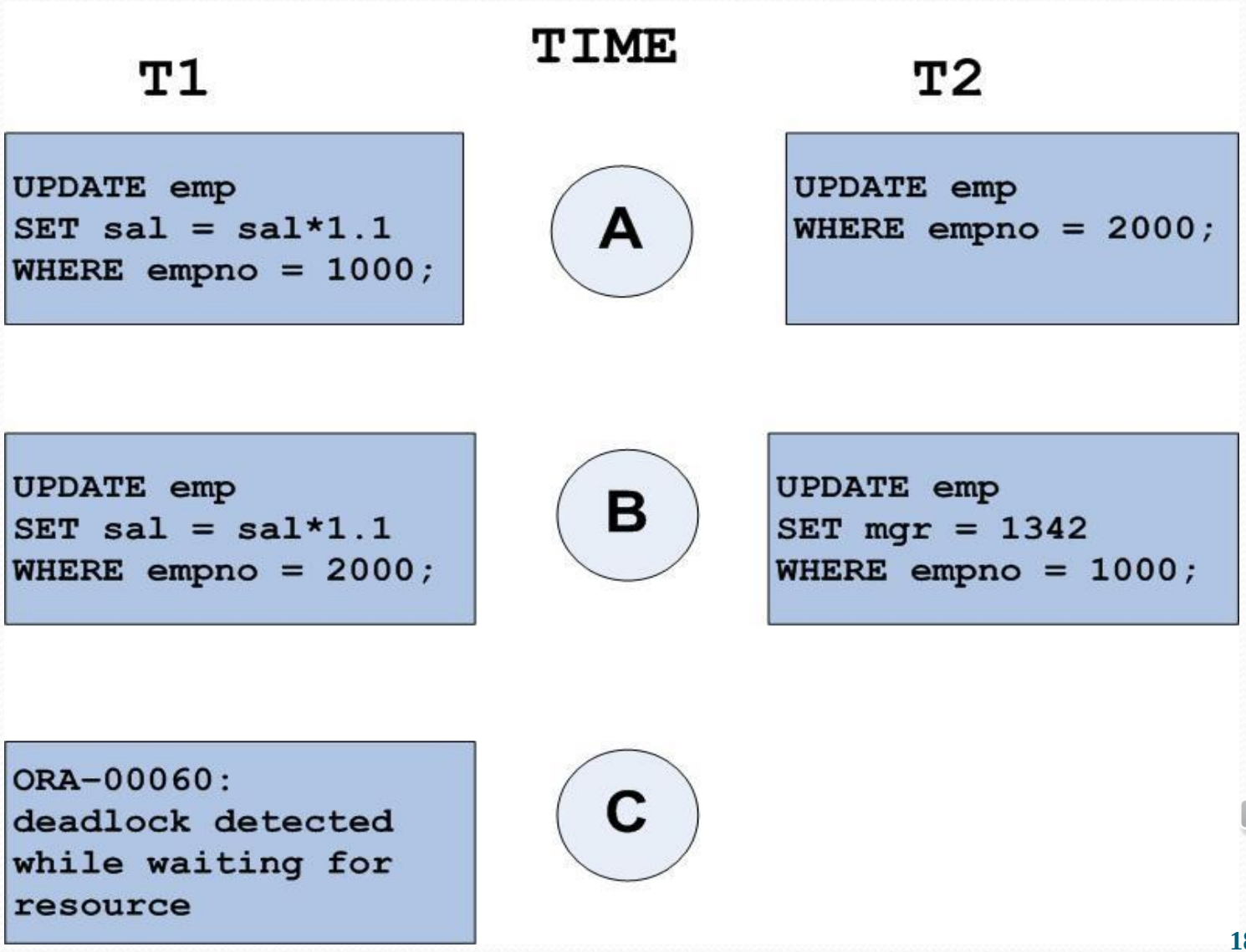
State of the Data after COMMIT

- Data changes are made *permanent* in the database.
- The previous state of the data *is permanently lost.*
- All users *can view* the results.
- Locks on the affected rows *are released*; those rows are available for other users to manipulate.
- All savepoints *are erased*.

Common recommendations

- Keep transactions as fast as possible
- Avoid executing long-running queries when transactions which update the table are also executing.

Deadlock


TIME
T1
T2
UPDATE emp
SET sal = sal*1.1
WHERE empno = 1000;
A
UPDATE emp
WHERE empno = 2000;
UPDATE emp
SET sal = sal*1.1
WHERE empno = 2000;
B
UPDATE emp
SET mgr = 1342
WHERE empno = 1000;
ORA-00060:
deadlock detected
while waiting for
resource
C

Types of Locks

Lock	Description
DML locks (data locks)	DML locks protect data For example, table locks lock entire tables, rowlocks lock selected rows.
DDL locks (dictionary locks)	DDL locks protect the structure of schema objects