Introduction to Transaction Processing Concepts and Theory

Dr. Dang Tran Khanh

Department of Information Systems

Faculty of Computer Science and Engineering

khanh@cse.hcmut.edu.vn

Outline

- Introduction to Transaction Processing
- Transaction and System Concepts
- Desirable Properties of Transactions
- Characterizing Schedules based on Recoverability
- Characterizing Schedules based on Serializability
- Transaction Support in SQL

Introduction to Transaction Processing

- **Multiuser vs. Single-User System**:
  - Many users can access the system concurrently vs. at most one user at a time can use the system
- **Concurrency**
  - Interleaved processing:
    - Concurrent execution of processes is interleaved in a single CPU
  - Parallel processing:
    - Processes are concurrently executed in multiple CPUs

Introduction to Transaction Processing

- A **Transaction**: Logical unit of database processing that includes one or more access operations (read - retrieval, write - insert or update, delete)
- A transaction (set of operations) may be stand-alone specified in a high level language like SQL submitted interactively, or may be embedded within a program
- **Transaction boundaries**:
  - Begin and End transaction
  - SQL ?

Introduction to Transaction Processing

- **Granularity** of data in a DB: a field, a record , or a whole disk block (Concepts are independent of granularity)
- Basic operations are **read** and **write**
  - **read_item(X**): Reads a database item named X into a program variable **X**
  - **write_item(X**): Writes the value of program variable X into the database item named X

Introduction to Transaction Processing

- READ & WRITE:
  - Basic unit of data transfer from the disk to the computer main memory is one block
  - read_item(X) command includes the following steps:
    - Find the address of the disk block that contains item X
    - Copy that disk block into a buffer in main memory (if not already in there))
    - Copy item X from the buffer to the program variable named X
  - write_item(X) command includes the following steps:
    - Find the address of the disk block that contains item X
    - Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer)
    - Copy item X from the program variable named X into its correct location in the buffer
    - Store the updated block from the buffer back to disk (either immediately or at some later point in time)

  - Discussion: Tree-based data structures' node size, disk block size and the querying performance in DBs

Two sample transactions

(a) T_1

read_item (X);
$X := X - N$;
write_item (X);
read_item (Y);
$Y := Y + N$;
write_item (Y);

(b) T_2

read_item (X);
$X := X + M$;
write_item (X);

Introduction to Transaction Processing
Why is concurrency control needed?

- **The Lost Update Problem**
  - This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect
- **The Temporary Update (or Dirty Read) Problem**
  - This occurs when one transaction updates a database item and then the transaction fails for some reason
  - The updated item is then accessed by another transaction before it is changed back to its original value
- **The Incorrect Summary Problem**
  - If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated

The lost update problem

Figure 17.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

(a)

Time

T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;
write_item(X); read_item(Y);	
	write_item(X);
$Y := Y + N$; write_item(Y);	

Item X has an incorrect value because its update by T_1 is *lost* (overwritten).

The temporary update problem

Figure 17.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

(b)

Time

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$X := X + M$;
	write_item(X);

read_item(Y);	

Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the *temporary* incorrect value of X.

The incorrect summary problem

Figure 17.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

(c)

T_1	T_3

	sum := 0;
	read_item(A);
	sum := *sum* + A;
	⋮

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	sum := *sum* + X;
	read_item(Y);
	sum := *sum* + Y;

read_item(Y);
$Y := Y + N$;
write_item(Y);	

T_3 reads X after N is subtracted and reads Y before N is added; a wrong summary is the result (off by N).

Introduction to Transaction Processing

- Why **recovery** is needed (What causes a Transaction to fail):
  - **A computer failure (system crash)**: A hardware or software error and the contents of the computer's internal memory may be lost
  - **A transaction or system error**:
    - Some operation in the transaction may cause it to fail, such as integer overflow or division by zero
    - Transaction failure may also occur because of erroneous parameter values or because of a logical programming error
    - The user may interrupt the transaction during its execution
  - **Local errors or exception conditions detected by the transaction:**
    - Certain conditions necessitate cancellation of the transaction: data for the transaction may not be found; a condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal from that account, to be canceled
    - A programmed abort in the transaction causes it to fail
  - **Concurrency control enforcement:**
    - The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock (cf. Chapter 18)

Introduction to Transaction Processing

- Why **recovery** is needed (What causes a Transaction to fail):
  - **Disk failure:**
    - Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction
  - **Physical problems and catastrophes:**
    - This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator

Outline

- Introduction to Transaction Processing
- Transaction and System Concepts
- Desirable Properties of Transactions
- Characterizing Schedules based on Recoverability
- Characterizing Schedules based on Serializability
- Transaction Support in SQL

Transaction and System Concepts

- A **transaction** is an atomic unit of work that is either completed in its entirety or not done at all
- Recovery manager keeps track of the following operations:
  - **begin_transaction**: Marks the beginning of transaction execution
  - **read** or **write**: Read or write operations on the database items that are executed as part of a transaction
  - **end_transaction**: Read and write transaction operations have ended and marks the end limit of transaction execution
  - **commit_transaction**: Any changes can be safely committed to the database and will not be undone
  - **rollback** (or **abort**): The transaction has ended unsuccessfully, so that any changes or effects must be undone

Transaction and System Concepts

- Recovery techniques use the following operators:
  - **undo**: Similar to rollback except that it applies to a single operation rather than to a whole transaction
  - **redo**: This specifies that certain *transaction operations* must be *redone* to ensure that all the operations of a committed transaction have been applied successfully to the database

State transition diagram illustrating the states for transaction execution

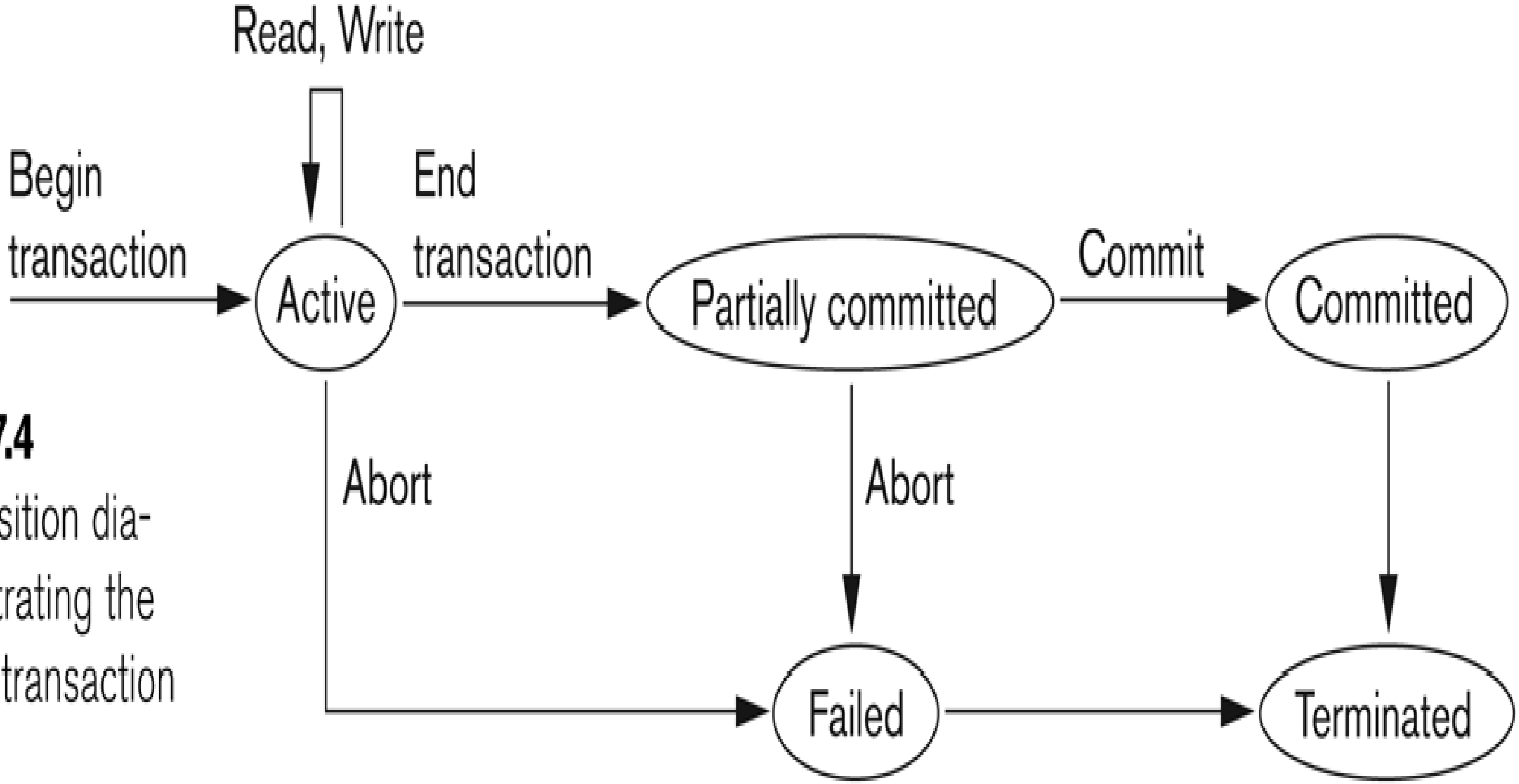


Figure 17.4
State transition diagram illustrating the states for transaction execution.

Transaction and System Concepts

- The System Log
  - **Log** or **Journal**: The log keeps track of all transaction operations that affect the values of database items
    - This information may be needed to permit recovery from transaction failures
    - The log is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure
    - The log is periodically backed up to archival storage to guard against such catastrophic failures

Transaction and System Concepts

- Recovery using log records:
- If the system crashes, we can recover to a consistent database state by examining the log and using one of the techniques described in Chapter 19

Transaction and System Concepts

- **Definition a Commit Point:**
  - A transaction T reaches its **commit point** when all its operations that access the database have been executed successfully *and* the effect of all the transaction operations on the database has been recorded in the log
  - Beyond the commit point, the transaction is said to be committed, and its effect is assumed to be permanently recorded in the database
  - The transaction then writes an entry [commit,T] into the log
- **Roll Back of transactions:**
  - Needed for transactions that have a [start_transaction,T] entry into the log but no commit entry [commit,T] into the log

Outline

- Introduction to Transaction Processing
- Transaction and System Concepts
- Desirable Properties of Transactions
- Characterizing Schedules based on Recoverability
- Characterizing Schedules based on Serializability
- Transaction Support in SQL

Desirable Properties of Transactions

- ACID properties:
  - **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all
  - **Consistency preservation**: A correct execution of the transaction must take the database from one consistent state to another
  - **Isolation**: A transaction should not make its updates visible to other transactions until it is committed; this property, when enforced strictly, solves the temporary update problem and makes cascading rollbacks of transactions unnecessary
  - **Durability or permanency**: Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure

Outline

- Introduction to Transaction Processing
- Transaction and System Concepts
- Desirable Properties of Transactions
- Characterizing Schedules based on Recoverability
- Characterizing Schedules based on Serializability
- Transaction Support in SQL

Characterizing Schedules based on Recoverability

- **Transaction schedule or history**:
  - When transactions are executing concurrently in an interleaved fashion, the order of execution of operations from the various transactions forms is called **a transaction schedule** (or history)
- A **schedule** (or **history**) S of n transactions T1, T2, …, Tn:
  - It is an ordering of the operations of the transactions subject to the constraint that, for each transaction Ti that participates in S, the operations of Ti in S must appear in the same order in which they occur in Ti
  - Note, however, that operations from other transactions can be interleaved with the operations of Ti in S

Characterizing Schedules based on Recoverability

Schedules classified on recoverability:

- **Recoverable schedule**:
  - One where no transaction needs to be rolled back
  - A schedule S is recoverable if no transaction T in S commits until all transactions T' that have written an item that T reads have committed
- **Cascadeless schedule**:
  - One where every transaction reads only the items that are written by committed transactions.

Characterizing Schedules based on Recoverability

- **Schedules requiring cascaded rollback**:
  - A schedule in which uncommitted transactions that read an item from a failed transaction must be rolled back
- **Strict Schedules**:
  - A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed

Outline

- Introduction to Transaction Processing
- Transaction and System Concepts
- Desirable Properties of Transactions
- Characterizing Schedules based on Recoverability
- Characterizing Schedules based on Serializability
- Transaction Support in SQL

Characterizing Schedules based on Serializability

- Serial schedule:
  - A schedule S is serial if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule
  - Otherwise, the schedule is called nonserial schedule
- Serializable schedule:
  - A schedule S is serializable if it is equivalent to some serial schedule of the same n transactions

Characterizing Schedules based on Serializability

- Result equivalent:
  - Two schedules are called result equivalent if they produce the same final state of the database
- Conflict equivalent:
  - Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules
- Conflict serializable:
  - A schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule S’

Characterizing Schedules based on Serializability

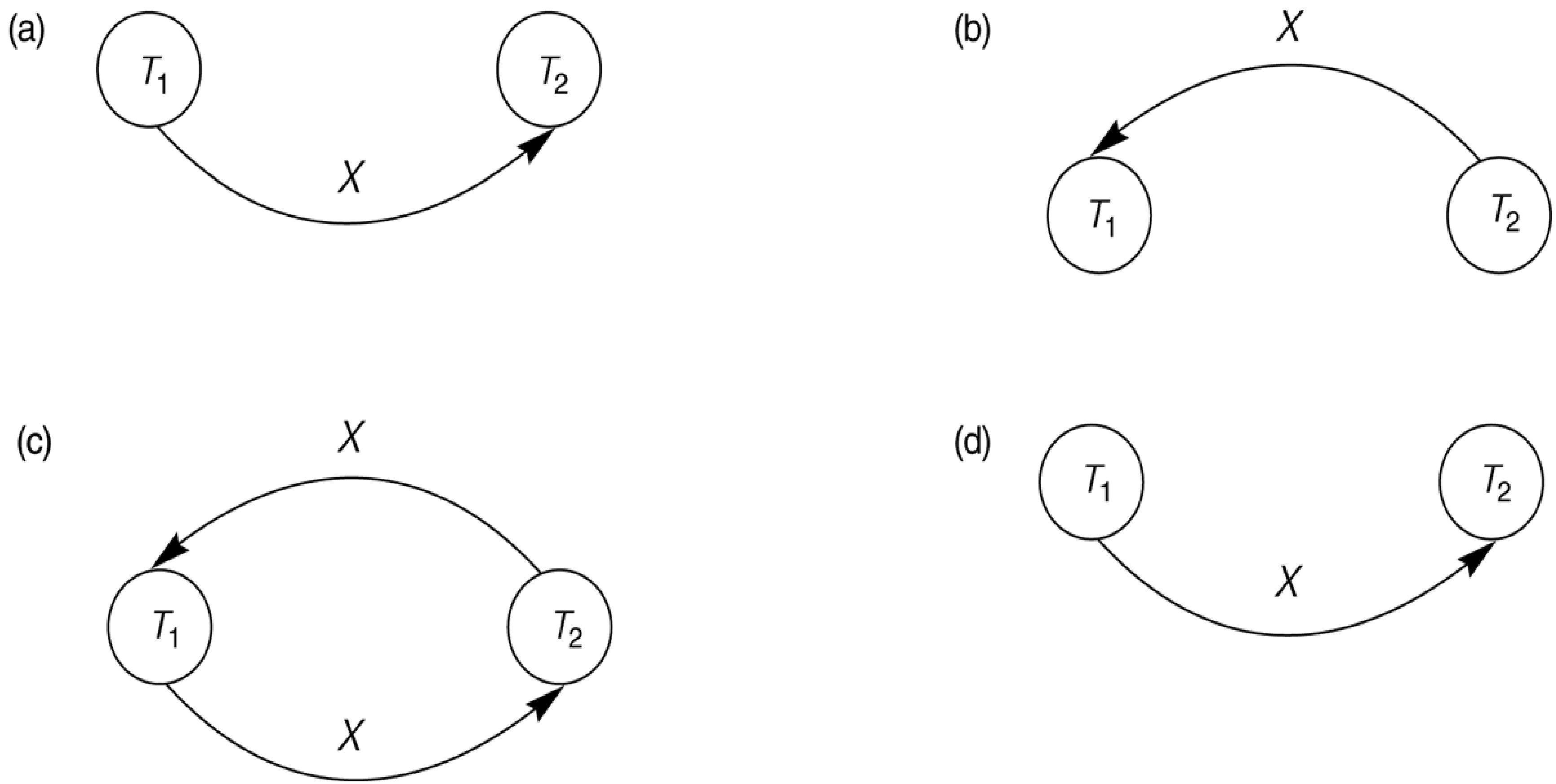
- Being serializable is <u>not</u> the same as being serial
- Being serializable implies that the schedule is a <u>correct</u> schedule
  - It will leave the database in a consistent state.
  - The interleaving is appropriate and will result in a state as if the transactions were serially executed, yet will achieve efficiency due to concurrent execution
- Current approach used in most DBMSs:
  - Use of locks with two phase locking

Characterizing Schedules based on Serializability

- **Testing for conflict serializability: Algorithm 17.1**
  - Looks at only read_Item (X) and write_Item (X) operations
  - Constructs a precedence graph (serialization graph) - a graph with directed edges
  - An edge is created from Ti to Tj if one of the operations in Ti appears before a conflicting operation in Tj
  - The schedule is serializable if and only if the precedence graph has no cycles

Constructing the Precedence Graphs

- FIGURE 17.7 Constructing the precedence graphs for schedules A and D from Figure 17.5 to test for conflict serializability.
  - (a) Precedence graph for serial schedule A.
  - (b) Precedence graph for serial schedule B.
  - (c) Precedence graph for schedule C (not serializable).
  - (d) Precedence graph for schedule D (serializable, equivalent to schedule A).



Characterizing Schedules based on Serializability

- Under special **semantic constraints**, schedules that are otherwise not conflict serializable may work correctly
  - Using commutative operations of addition and subtraction (which can be done in any order) certain non-serializable transactions may work correctly

Characterizing Schedules based on Serializability

- Example: bank credit / debit transactions on a given item are **separable** and **commutative**
  - Consider the following schedule S for the two transactions:
  - Sh : r1(X); w1(X); r2(Y); w2(Y); r1(Y); w1(Y); r2(X); w2(X);
  - Using conflict serializability, it is **not serializable**.
  - However, if it came from a (read,update, write) sequence as follows:
    - r1(X); X := X – 10; w1(X); r2(Y); Y := Y – 20;r1(Y);
    - Y := Y + 10; w1(Y); r2(X); X := X + 20; (X);
  - Sequence explanation: debit, debit, credit, credit.
  - It is a *correct schedule <u>for the given semantics</u>*

Outline

- Introduction to Transaction Processing
- Transaction and System Concepts
- Desirable Properties of Transactions
- Characterizing Schedules based on Recoverability
- Characterizing Schedules based on Serializability
- Transaction Support in SQL

Transaction Support in SQL2

- A **single** SQL statement is always considered to be **atomic**
- With SQL, there is <u>no explicit Begin</u> Transaction statement
- Every transaction <u>must have an explicit end</u> statement, which is either a COMMIT or ROLLBACK

Transaction Support in SQL2

Characteristics specified by a SET TRANSACTION statement in SQL2:

- **Access mode**:
  - READ ONLY or READ WRITE: the default is READ WRITE unless the isolation level of READ UNCOMITTED is specified, in which case READ ONLY is assumed
- **Diagnostic size** n: specifies an integer value n, indicating the number of conditions that can be held simultaneously in the diagnostic area

Transaction Support in SQL2

- **Isolation level** <isolation>: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ or SERIALIZABLE
  - The default is SERIALIZABLE
  - With SERIALIZABLE: the interleaved execution of transactions will adhere to our notion of serializability
  - However, if any transaction executes at a lower level, then serializability may be violated

Transaction Support in SQL2

Potential problem with lower isolation levels:

- **Dirty Read**:
  - Reading a value that was written by a transaction which failed
- **Nonrepeatable Read**:
  - Allowing another transaction to write a new value between multiple reads of one transaction
  - A transaction T1 may read a given value from a table. If another transaction T2 later updates that value and T1 reads that value again, T1 will see a different value
- **Phantoms:**
  - New rows being read using the same read with a condition

Transaction Support in SQL2

- Sample SQL transaction:

```
EXEC SQL whenever sqlerror go to UNDO;
EXEC SQL SET TRANSACTION
        READ WRITE
        DIAGNOSTICS SIZE 5
        ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT
       INTO EMPLOYEE (FNAME, LNAME, SSN, DNO, SALARY)
       VALUES ('Robert','Smith','991004321',2,35000);
EXEC SQL UPDATE EMPLOYEE
       SET SALARY = SALARY * 1.1
       WHERE DNO = 2;
EXEC SQL COMMIT;
       GOTO  THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END:  ...
```

Transaction Support in SQL2 (7)

- Possible violation of serializabilty:

Isolation level	Dirty read	Nonrepeatable read	Phantom
READ UNCOMMITTED	yes	yes	yes
READ COMMITTED	no	yes	yes
REPEATABLE READ	no	no	yes
SERIALIZABLE	no	no	no

Summary

- Transaction and System Concepts
- Desirable Properties of Transactions
- Characterizing Schedules based on Recoverability
- Characterizing Schedules based on Serializability
- Transaction Support in SQL
- Next lecture: Concurency control techniques (chapter 18)