Data Bases II
Theory

Christian Rossi

Academic Year 2023-2024

Abstract

The course aims to prepare software designers on the effective development of database applications. First, the course presents the fundamental features of current database architectures, with a specific emphasis on the concept of transaction and its realization in centralized and distributed systems. Then, the course illustrates the main directions in the evolution of database systems, presenting approaches that go beyond the relational model, like active databases, object systems and XML data management solutions.

Contents

Chapter 1

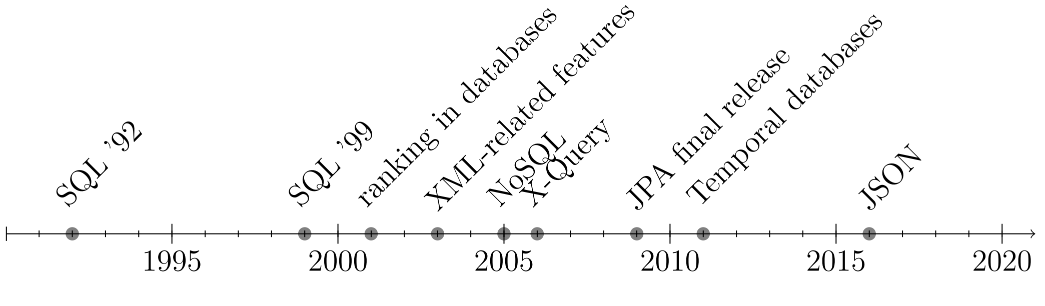
Introduction

1.1 Data Base Management System

Definition

A *Data Base Management System* is a software product capable of managing data collections that are:

- Large: much larger than the central memory available on the computers that run the software.
- Persistent: with a lifetime which is independent of single executions of the programs that access them.
- Shared: used by several applications at a time.
- Reliable: ensuring tolerance to hardware and software failures.
- Data ownership respectful: by disciplining and controlling accesses.



1.2 Transactions

Definition

A *transaction* is an elementary, atomic unit of work performed by an application. Each transaction is conceptually encapsulated within two commands:

- Begin transaction.
- End transaction.

Within a transaction, one of the commands below is executed to signal the end of the transaction:

- Commit-work (commit).
- Rollback-work (abort).

Definition

The *On-Line Transaction Processing* (OLTP) is a system that supports the execution of transactions on behalf of concurrent applications.

The application can run many transactions. So, the transactions are part of the application and not vice-versa. The transactions follow the ACID property:

1. Atomicity: a transaction is an indivisible unit of execution. This means that all the operations in the transaction are executed or none is executed. The time in which commit is executed marks the instant in which the transaction ends successfully: an error before should cause the rollback and an error after should not alter the transaction. The rollback of the work performed can be caused by a rollback statement or by the DBMS. In case of a rollback, the work performed must be undone, bringing the database to the state it had before the start of the transaction. It is the application's responsibility to decide whether an aborted transaction must be redone or not.

2. Consistency: A transaction must satisfy the database integrity constraints: if the initial state S_0 is consistent, then the final state S_f is also consistent. This is not necessarily true for the intermediate states S_i. For example, the sum of the worked hours per task should equal the planned work hours of the project. If the constraint holds before the transaction it must hold also after its execution. The constraint can

be temporarily violated during the execution of the transaction (e.g., when shifting work from a task tom another one, but must be satisfied at the end).

3. Isolation: the execution of a transaction must be independent of the concurrent execution of other transactions. In particular, the concurrent execution of a number of transactions must produce the same result as the execution of the same transactions in a sequence. Isolation impacts performance and trade-offs can be defined between isolation and performance.

4. Durability: The effect of a transaction that has successfully committed will last forever, independently of any system fault.

Property	**Actions**	**Architectural element**
Atomicity	Abort-rollback-restart	Query Manager
Consistency	Integrity checking	Integrity Control System
Isolation	Concurrency control	Concurrency Control System
Durability	Recovery management	Reliability Manager

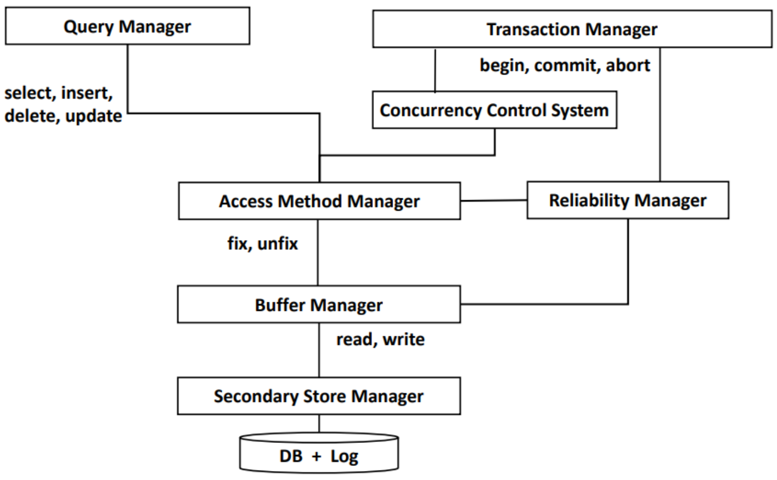


Figure 1.1: Architecture of a Data Base Management System

Chapter 2

Concurrency

2.1 Introduction

A DBMS usually needs to manage multiple applications. A unit of measurement used to evaluate the DBMS workload is the number of transaction per second (tps) handled by it. To have an efficient usage of the database the DBMS needs to be able to handle concurrency while avoiding the insurgence of anomalies. The concurrency control system schedules the order of the various transactions.

2.2 Anomalies in concurrent transactions

The typical types on anomalies in concurrency are:

- Lost update: an update is applied from a state that ignores a preceding update, which is lost.

Transaction t_1	**Transaction t_2**
$r_1(x)$	
$x = x + 1$	
	$r_2(x)$
	$x = x + 1$
	$w_2(x)$
	commit
$w_1(x)$	
commit	

- Dirty read: an uncommitted value is used to update the data.

Transaction t_1	**Transaction** t_2
$r_1(x)$	
$x = x + 1$	
$w_1(x)$	
	$r_2(x)$
	commit
abort	

- Non-repeatable read: someone else updates a previously read value.

Transaction t_1	**Transaction** t_2
$r_1(x)$	
	$r_2(x)$
	$x = x + 1$
	$w_2(x)$
	commit
$r_1(x)$	
commit	

- Phantom update: someone else updates data that contributes to a previously valid constraint.

Transaction t_1	**Transaction** t_2
$r_1(x)$	
	$r_2(y)$
$r_1(y)$	
	$y = y - 100$
	$r_2(z)$
	$z = z + 100$
	$w_2(y)$
	$w_2(z)$
	commit
$r_1(z)$	
$s = x + y + z$	
commit	

- Phantom insert: someone else inserts data that contributes to a previously read datum.

2.3 Concurrency theory

Definition

A *model* is an abstraction of a system, object or process, which purposely disregards details to simplify the investigation of relevant properties.

Concurrency theory builds upon a model of transaction and concurrency control principles that help understanding the real systems. Real systems exploit implementation level mechanisms (locks, snapshots) which help achieve some desirable properties postulated by the theory.

Definition

An *operation* consist in a reading or in a writing of a specific datum by a specific transaction.

Definition

A *schedule* is a sequence of operations performed by concurrent transactions that respects the order of operations of each transaction.

The transactions can be: serial, interleaved or nested. The number of serial schedules for n transaction is equal to

$$N_S = n!$$

while the total number of distinct schedules given the number of transaction n is equal to:

$$N_D = \frac{(\sum_{i=1}^{n} k_i)!}{\prod_{i=1}^{n} (k_i!)}$$

Example : Given two transaction T_1 and T_2 we have six possible different schedules, where only two are serial:

1. $r_1(x) \rightarrow w_1(x) \rightarrow r_2(z) \rightarrow w2(z)$
2. $r_2(z) \rightarrow w_2(z) \rightarrow r_1(x) \rightarrow w_1(x)$
3. $r_1(x) \rightarrow r_2(z) \rightarrow w_1(x) \rightarrow w_2(z)$
4. $r_2(z) \rightarrow r_1(x) \rightarrow w_2(z) \rightarrow w_1(x)$
5. $r_1(x) \rightarrow r_2(z) \rightarrow w_2(z) \rightarrow w_1(x)$
6. $r_2(z) \rightarrow r_1(x) \rightarrow w_1(x) \rightarrow w_2(z)$

The first two are serial, the third and the fourth are nested, and the last two interleaved.