TRANSACTIONS & CONCURRENCY CONTROL

Martin Read

Aim: Transactions & Extractions

- Describe the database transaction management process
- Identify four properties of a database transaction
- Explain concurrency control and its role in maintaining database integrity
- Describe how locking methods are used for concurrency control

Introduction

- Concurrent access is essential for good DBMS performance
- Disk accesses are frequent (&relatively slow)
  - important to keep the CPU going by working on several user programs concurrently

- Chapter 10 in the Database book

Transactions

- Logical unit of work composed of one or more SQL statements
- DBMS only concerned about what data is read/written
- A collection of DB changes that must all succeed, otherwise the DB would be left in an inconsistent state
- Transactions can have a number of parts
  - E.g. adjusting the product number after an order has taken place
  - Entering a value for an invoice once an order has been dispatched

Transaction

- SQL commands can control the processing
- Allowed to either complete transaction or abort & not have data errors
- Prevents only partial changes caused by interruptions (such as a crash) leaving the DB in an inconsistent state
  - E.g. Referential Integrity is compromised

Transaction Example

- A student enrolls to take a course
- Course enrolment number is updated
  - Check made to ensure enrolment number is not above capacity
- Student record is updated to link the student with the course

Choice: Abort or complete?

If you find the course enrolment is now above the capacity you want to reject the booking but stop the transaction to update the records

If it is under, you want to complete the whole thing

Database integrity

- At the start of the set of SQL statements the database must be stable
  - A consistent state where all constraints are in place
- Every transaction must begin with this state
- A transaction would contain both the check for the capacity as well as the insert & update for the student / course records

BEGIN TRANSACTION
SELECT capacity, enrolment FROM Course;

IF (capacity < enrolment) …UPDATE course….

INSERT into student_Details…

COMMIT;

Transaction Management

A transaction should continue until one of four events happen

- A COMMIT statement is reached
- A ROLLBACK statement is reached
- End of the program is reached
- Program is abnormally terminated

Automatic Rollback

If a transaction fails because of a severe error, such as
  - loss of network connection
  - failure of client application
  - failure of computer
- SQL Server automatically rolls back transaction
- Reverses all modifications performed & frees resources
- If a run-time statement causes an error, such as constraint or rule violation
  - SQL Server rolls back only statement in error

Rollback statement

- Terminates transaction
  - reverses any changes made
- If triggered in the middle of a transaction
  - rest of transaction ignored
- If in a stored procedure
  - entire procedure rolled back
  - processing resumes at next statement after the stored procedure call

Commit statement

- All modifications performed by a transaction are made permanent
- All resources used by the transaction are released
- Transaction cannot be rolled back after a commit

Transaction example

- Delete Customer
- We pre-suppose the foreign key will enforce the constraint

```
CREATE PROCEDURE Delete_Customers(@CustomerId as INT)
AS
BEGIN

    BEGIN TRANSACTION
        DECLARE @Error NVARCHAR(Max);

        BEGIN TRY

            DELETE FROM Customers
            WHERE CustomerId = @CustomerId;

            IF @@TRANCOUNT > 0 COMMIT;
        END TRY
        BEGIN CATCH
            SET @Error = 'An error was encountered : Delete could not
 happen';
            IF @@TRANCOUNT > 0
                ROLLBACK TRANSACTION;
            RAISERROR(@Error, 1, 0);
        END CATCH;
END;
```

Transaction results

- Not all transactions update or change the database

BUT

- any SQL code that accesses the database represents a transaction
- Incomplete transactions will seriously compromise data quality

Database designers need to consider what constraints need to be enforced to protect business considerations

Transaction Properties

Atomicity

Consistency

- Transaction must take the database from one consistent state to another. The database should be in a permanently consistent state

Isolation

- Transaction should appear as though it is being executed in isolation from other transactions
  - Data used during a transaction cannot be used by a second transaction until the first one has been completed

Serialisability

- The schedule for the concurrent execution of a number of transactions should yield consistent results

Atomicity of Transactions

- All parts of a transaction must be completed or not performed at all
- A transaction might *commit* after completing all its actions, or it could *abort* (or be aborted by the DBMS) after executing some actions
- A very important property guaranteed by the DBMS for all transactions is that they are ***atomic***
  - a user can think of a transaction as always executing all its actions in one step, or not executing any actions at all

Distributed systems

- How many users might a system have?
- What if ….

We have a train ticket booking system
So many tickets are left at price N
Thousands of users use the application
Worldwide

What could happen?
Why might transactions be used here?

Concurrency

Multiple users access databases & use computer systems simultaneously

- Better transaction throughput & response time
- Better utilization of resource

Example: Airline reservation system

- used by hundreds of travel agents concurrently

Concurrency

- DBMS interleaves actions (reads/writes) of various transactions
- Each transaction must leave the database in a consistent state if the DB is consistent when the transaction begins
  - DBMS will enforce some constraints
  - DBMS does not really ‘understand’ semantics of the data
    - e.g., does not understand how interest on a bank account is computed

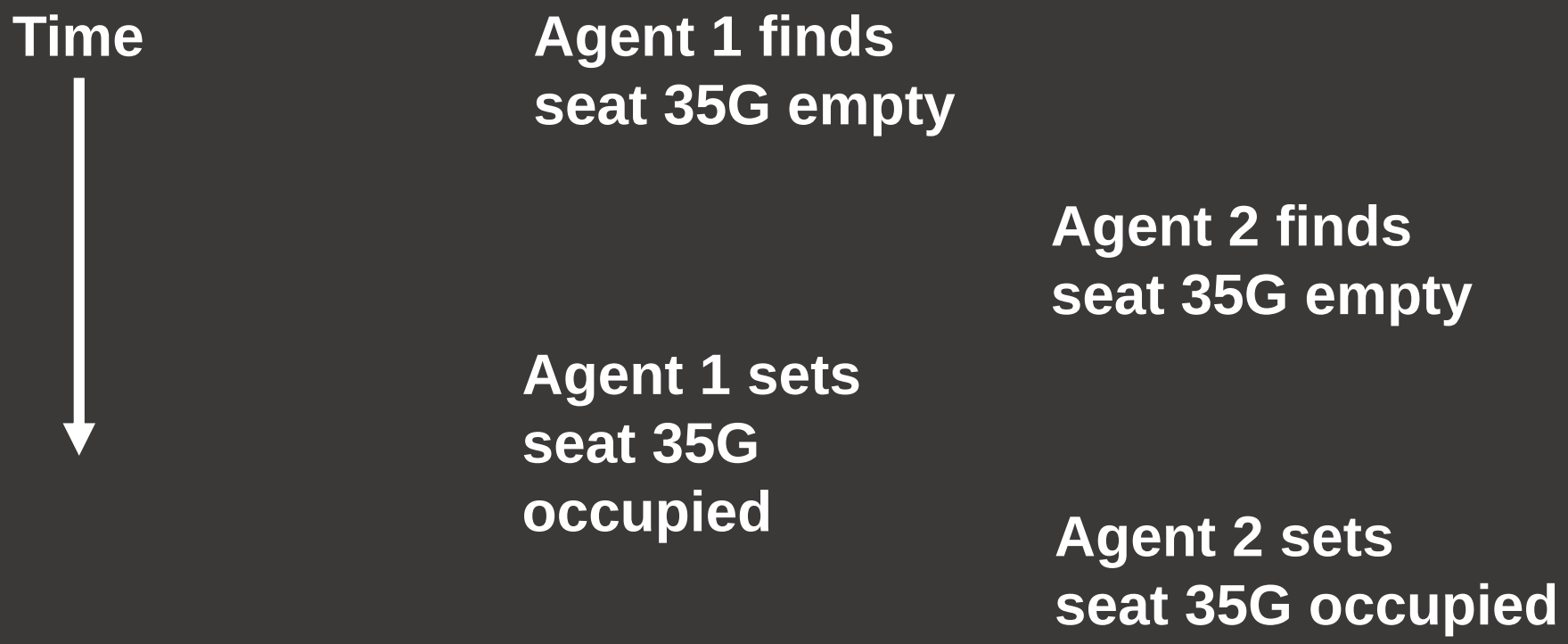
Concurrency control

- Coordinates simultaneous transactions execution in a multi-user database system
  - Simultaneous execution of transactions over a shared database can create data integrity & consistency problems

Main problems are lost updates, uncommitted data, inconsistent retrievals

Airline reservation system

- In concurrent access to data in DBMS, two users may try to book the same seat simultaneously



What can result?

- Reading uncommitted data, aka “dirty reads”
- Unrepeatable reads
- Lost updates

CRASH!

- Transaction may be aborted before committed
  - undo uncommitted transactions
  - undo transactions that sees the uncommitted change before the crash

Concurrency problems

Lost updates

- occurs when two concurrent transactions attempt to update the same data element - one of those will be lost

Uncommitted data

- occurs when two concurrent transactions conflict. First transaction rolled back after second transaction has already accessed uncommitted data

Inconsistent retrievals

- Transaction accesses data before & after one or more other transactions finish working with that data

Lock-based Concurrency Control

Mechanism to control concurrent access to a data object

- each element has a unique lock
- Each transaction must first acquire the lock before reading/writing that element
  - If lock taken by another transaction - wait

Transaction must release the lock(s)

- Transaction getting the lock first must complete before the other transaction can proceed

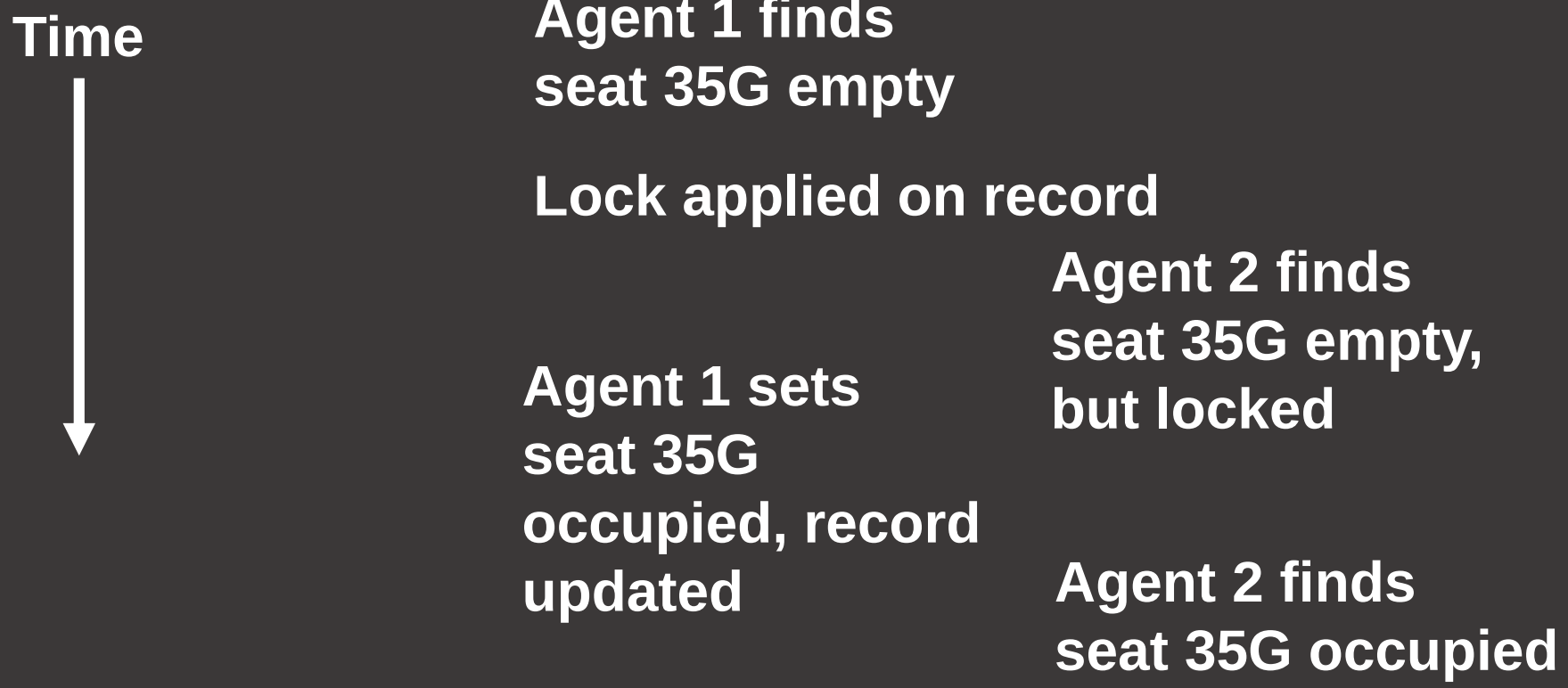
Locking

Stops access to data whilst a transaction is going on

- Guarantees exclusive use of data item to current transaction
- Lock manager – responsible for assigning & policing the locks used by the transactions
- SQL Server uses locking – ensures logical integrity of transactions & data

Airline reservation system

- In concurrent access to data in DBMS, two users may try to book the same seat simultaneously



Conflict operations

- Two operations in a schedule are said to be conflict if they satisfy all three of the following conditions:
    1) They belong to different transactions
    2) They access the same item A;
    3) At least one of the operations is a write(A)

Data Control Language (DCL)

COMMIT : Stores changes permanently in the database

ROLLBACK : Restores database to a stable state after the last Commit

Views!


University of Plymouth
Advancing knowledge, transforming lives