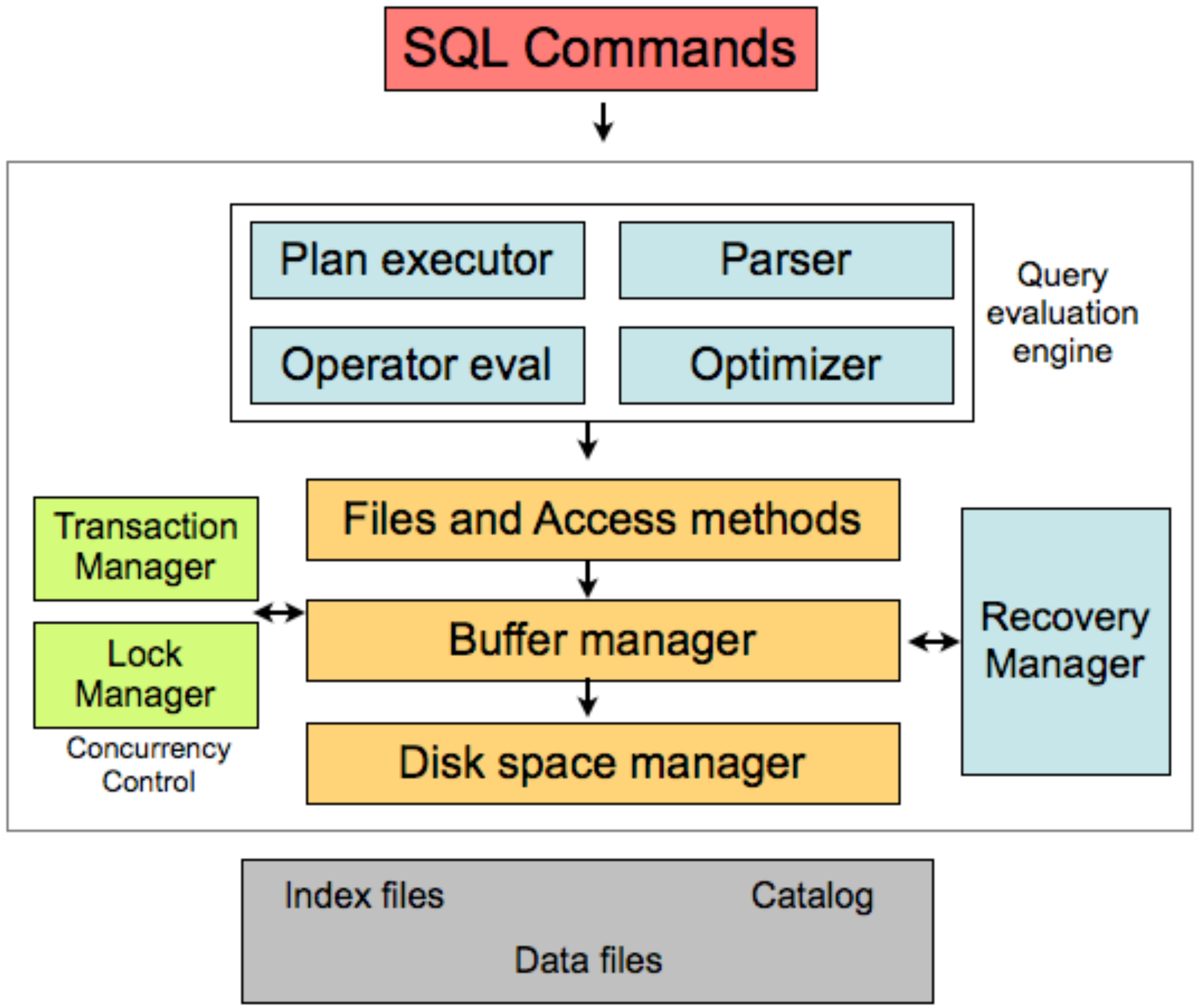
COSC 460 Lecture 14: Transactions 1

Professor Michael Hay

Fall 2018

Architecture of DBMS



Attribute-level inconsistency

Example 1

S1:

```
update college
set enrollment = enrollment + 100
where cname = ‘colgate’
```

concurrent with…

S2:

```
update college
set enrollment = enrollment + 150
where cname = ‘colgate’
```

Tuple-level inconsistency

Example 2

S1:

```
update apply
set major = 'cs'
where sid = 123
```

concurrent with…

S2:

```
update apply
set decision = 'y'
where sid = 123
```

Table-level inconsistency

Example: Alice and Kate both have GPA = 3.8 but are from "large" high schools (> 2500 students).

Inconsistent outcome: Alice rejected; Kate accepted!

Example 3

S1:

```
update apply
set decision = 'y'
where sid in (select sid
              from student where gpa > 3.9)
```

concurrent with…

S2:

```
update student
set gpa = (1.1) * gpa
where sizehs  > 2500
```

Multi-statement inconsistency

Example 4

Group 1:

```
insert into archive (select *
                     from apply
                     where decision = 'n');
delete from apply where decision = 'n';
```

concurrent with…

Group 2:

```
select count(*) from apply;
select count(*) from archive;
```

Concurrency and inconsistency

Instructions:
~1 minute to think/answer on your own; then discuss with neighbors; then I will call on one of you

Exercise

Give an example of an *inconsistent state* that could arise from concurrently executing the two statements shown. Suppose R is a relation with a single attribute A and two tuples, as shown below.

R(A)
5
6

S1:

```
update R
set A = A+1
```

concurrent with…

S2:

```
update R
set A = 2*A
```

Concurrency & Inconsistency

- In previous examples, ***inconsistency*** was caused by ***concurrency***
- Goal: execute sequence of SQL statements so they *appear* to be running in ***isolation***
- Solutions
  - Simple (but bad): execute in isolation (serial order)
  - Better: enable concurrency whenever safe to do so (locking, details soon!)

Inconsistency due to system failure

Example 5

Group:

```
insert into archive (select *
                     from apply
                     where decision = ’n’);
delete from apply where decision = ‘n’;
```

Suppose system crashes after insert but before delete?

Given what you know, explain how changes made by the insert could be lost even though the crash happened after insert “completed.”

Inconsistency and failure

- In previous example, ***inconsistency*** was caused by ***system failure***
- Goals:
    - Guarantee *"all or nothing"* execution
    - Guarantee that changes *persist* in database
- Solution: logging (details later) + stable storage

Transactions

- **Transaction** a sequence of SQL operations treated as a unit.
- Transactions appear to run in isolation
- Transaction either runs to completion or not at all
- ACID Properties
  - Atomicity
  - Consistency
  - Isolation
  - Durability

```
BEGIN TRANSACTION;
insert into archive
(select *
 from apply
 where decision = 'n');
delete from apply
where decision = 'n';
COMMIT;
```

ACID properties

- **Isolation:** varying degrees of isolation supported in DBMSs (e.g., “REPEATABLE READ”, “READ COMMITTED”)
- We will focus on *serializability*
- **Serializability**: operations from *different* transactions may be interleaved but execution must be *equivalent* to some serial order

Isolated transactions

Result is same whether T1 then T2 or T2 then T1.

Example 1

T1:

```
update college
set enrollment = enrollment + 100
where cname = 'colgate'
```

concurrent with…

T2:

```
update college
set enrollment = enrollment + 150
where cname = 'colgate'
```

Isolated transactions

Result is same whether T1 then T2 or T2 then T1.

Example 2

T1:

```
update apply
set major = 'cs'
where sid = 123
```

concurrent with…

T2:

```
update apply
set decision = 'y'
where sid = 123
```

Isolated transactions

Result can be different!
T1 then T2: Kate and Alice rejected
T2 then T1: Kate and Alice accepted
While different, both are *consistent*.

Example 3

T1:

```
update apply
set decision = 'y'
where sid in (select sid
              from student where gpa > 3.9)
```

concurrent with…

T2:

```
update student
set gpa = (1.1) * gpa
where sizehs  > 2500
```

Isolated transactions

Result can be different!
T1 then T2: archive count includes updated records
T2 then T1: archive count does not include updated records

Example 4

T1:

```
insert into archive (select *
                     from apply
                     where decision = 'n');
delete from apply where decision = 'n';
```

concurrent with…

T2:

```
select count(*) from apply;
select count(*) from archive;
```

Poll

Instructions: *I will give you 1-2 minutes to think on your own.*
Vote 1.
Then you will discuss w/ neighbor (1 min).
Vote 2.
Then we'll discuss as class.

R(A)
5
6

Suppose R is a relation with a single attribute A and two tuples, as shown. Suppose the two statements shown are executed in *isolation*. Which of the following states is impossible?

A. {11,13}
B. {12, 14}
C. {11,14}
D. {12,13}
E. more than one of above

S1:

```
update R
set A = A+1
```

concurrent with…

S2:

```
update R
set A = 2*A
```

ACI**D** properties

- **Durability**: if system crashes after transaction commits, all effects of transaction persist in database.
- **Atomicity**: each transaction is "all-or-nothing", it either runs to completion and commits, or it fails and all partial changes are undone.

Atomic transactions

Example 5

```
BEGIN TRANSACTION;
insert into archive (select *
                     from apply
                     where decision = 'n');
delete from apply where decision = 'n';
COMMIT;
```

Suppose system crashes after insert but before delete?

Rollback: undoing effects of a partially completed transaction.

Without durability

Instructions: *I will give you 1-2 minutes to think on your own.*
Vote 1.
Then you will discuss w/ neighbor (1 min).
Vote 2.
Then we'll discuss as class.

Exercise

Suppose our DBMS supports atomicity and isolation but not durability. Which of the following states of R is *not possible?*

A. {11,13}
B. {12, 14}
C. {10,12}
D. {11,12}
E. more than one of above

Assume R is a relation with a single attribute A and two tuples, as show.

R(A)
5
6

S1:

```
update R
set A = A+1
```

concurrent with…

S2:

```
update R
set A = 2*A
```

A**C**ID properties

- **Consistency**: if DB is in a consistent state when each transaction starts, it will be in consistent state when transaction ends
- User responsible for writing semantically correct transactions (i.e., consistent with real world)
- If transaction is consistent and DBMS ensures serializability, then DB always *appears* to be in a consistent state.