Practice and Applications of Data Management

CMPSCI 345

Lecture 13: Transactions
Transactions

Problem: An application must perform several writes and reads to the database, as a unit.
  
  Example: Two people attempt to book the last seat on a flight.

Solution: Multiple actions of the application are bundled into one unit called **Transaction**
  
  Transactions guarantee certain properties to hold that prevent such problems.
Transactions in Applications

- Default: each statement = one transaction

- Multi-statement transactions:

  ```
  START TRANSACTION

  [SQL statements]

  COMMIT or ROLLBACK (=ABORT)
  ```
Practice

- Start 2 psql instances on our practice database
practice=# start transaction;
START TRANSACTION
practice=#

practice=# select * from product;
practice=# start transaction;
START TRANSACTION
practice=# update product
practice-# set price = price+10
practice-# where pname='Gizmo';

practice=# select * from product;
... practice=# select * from product;
practice=# select * from product;
practice=# commit;
commit
practice=#
practice=# start transaction;
START TRANSACTION
practice=#
... 
practice=# update employees 
practice-# set phone = '1234' 
practice-# where name= 'John'; 

...
practice=# update employees 
practice-# set phone = NULL 
practice-# where name= 'Cecilia';
... practice=# update employees
practice-# set phone = '1234'
practice-# where name= 'John';

UPDATE 1

practice=# update employees
practice-# set phone = NULL
practice-# where name= 'Cecilia';

UPDATE 1

practice=# update employees
practice-# set phone = '9876'
practice-# where name= 'John';
... 

practice=# commit;

... 

practice=# select * from employees;

... 

practice=# rollback;
practice=# start transaction;
START TRANSACTION
practice=#
... 
practice=# update employees
practice-# set phone = '0000'
practice-# where name= 'John';
...

practice=# update employees
practice-# set phone = NULL
practice-# where name= 'Cecilia';
... 

practice=# update employees 
practice-# set phone = '0000' 
practice-# where name= 'John';

... 

practice=# update employees 
practice-# set phone = NULL 
practice-# where name= 'Cecilia';
UPDATE 1 
practice=# update employees 
practice-# set phone = NULL 
practice-# where name= 'John';
... practice=# update employees
practice-# set phone = '5555555'
practice-# where name= 'Cecilia';
...  
practice=# update employees  
practice-# set phone = '5555555'  
practice-# where name= 'Cecilia';

ERROR: deadlock detected
practice=# update employees
practice-# set phone = '5555555'
practice-# where name= 'Cecilia';

ERROR:  deadlock detected

practice=# commit;
ROLLBACK
practice=# start transaction;
START TRANSACTION
practice=#
... practice=# update employees
practice-# set phone = '555-3245'
practice-# where name= 'John';

...

practice=# select * from employees;
... practice=# update employees
practice-# set phone = '555-3245'
practice-# where name='John';
practice=# commit;
COMMIT

... practice=# select * from employees;
... practice=# select * from employees;

Unrepeatable read!
Isolation Levels in SQL

1. “Dirty reads”
   SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

2. “Committed reads”
   SET TRANSACTION ISOLATION LEVEL READ COMMITTED

3. “Repeatable reads”
   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

4. Serializable transactions
   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
Choosing Isolation Level

- Trade-off: efficiency vs correctness

- DBMSs give user choice of level

Beware!!
- Default level is often NOT serializable
- Default level differs between DBMSs
- Some engines support subset of levels!
- Serializable may not be exactly ACID

Always read DBMS docs!
Isolation levels (PostgreSQL doc.)

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Dirty Read</th>
<th>Nonrepeatable Read</th>
<th>Phantom Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read uncommitted</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Read committed</td>
<td>Not possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Repeatable read</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Serializable</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
</tr>
</tbody>
</table>

In Postgres:

SELECT current_setting('transaction_isolation');
READ-ONLY Transactions

Client 1:  START TRANSACTION
           INSERT INTO SmallProduct(name, price)
           SELECT pname, price  
           FROM Product  
           WHERE price <= 0.99

           DELETE FROM Product  
           WHERE price <=0.99
           COMMIT

Client 2:  START TRANSACTION
           SET TRANSACTION READ ONLY
           SELECT count(*)  
           FROM Product

           SELECT count(*)  
           FROM SmallProduct
           COMMIT

Can help DBMS improve performance
1. Isolation Level: Dirty Reads

Implementation using locks:

- “Long duration” WRITE locks
  - A.k.a Strict Two Phase Locking
- Do not use READ locks
  - Read-only transactions are never delayed

Possible problems: dirty and inconsistent reads
2. Isolation Level: Read Committed

Implementation using locks:

- “Long duration” WRITE locks
- “Short duration” READ locks
  - Only acquire lock while reading (not 2PL)

- Possible problems: unrepeatable reads
  - When reading same element twice,
  - may get two different values
3. Isolation Level: Repeatable Read

Implementation using locks:

- “Long duration” READ and WRITE locks
  - Full Strict Two Phase Locking

- This is not serializable yet !!!
practice=# start transaction;
START TRANSACTION
practice=# set transaction
practice-># isolation level
practice-># repeatable read;
SET
practice=# select count(*)
practice-# from projects
practice-# where empID=2;

practice=# select count(*)
practice-# from projects
practice-# where empID=1;
practice=# insert into projects
practice-# values(1,'Web design');
INSERT 0 1

practice=# insert into projects
practice-# values(2,'Web design');
INSERT 0 1
... practice=# commit;
COMMIT

... practice=# commit;
COMMIT
```
practice=# start transaction;
START TRANSACTION
practice=# set transaction
practice-# isolation level
practice-# serializable;
SET
```
... practice=# select count(*) from projects where empID=2;

... practice=# select count(*) from projects where empID=1;
practice=# insert into projects
practice-# values(1,'Android game');
INSERT 0 1

practice=# insert into projects
practice-# values(2,'Android game');
INSERT 0 1
practice=# commit;
COMMIT

practice=# commit;
ERROR: could not serialize access due to read/write dependencies among transactions
```bash
practice=# commit;
COMMIT

ERROR: could not serialize access due to read/write dependencies among transactions
...

practice=# select * from projects;
```