Posted by mingodad on Mon, 18 Feb 2008 19:54:17 GMT

View Forum Message <> Reply to Message

In my opnion the first approach could allow for one user to overwrite work for another user. The second approach prevent that but at the cost of block the database, no one will be able to work on that record reliable.

In my apps I use to include a field in every table that has high probability of concurrency and increment it at each update, when I retrieve data for edit I use the value of that field as one clause in that way the database is free and the faster user will be able to work, at expenses of the slow one.

Example:

```
create table test (
id int primary key,
rec_version int default 0,
data text
);

insert into test(data) values('First value');
-- id = 1, rec_version = 0, data = 'First value'

-- when one user edit
select * from test;
-- id = 1, rec_version = 0, data = 'First value'
-- let's suppose this user want data = 'Second value'
-- but he went to have a coffe

-- when another user edit too
select * from test;
-- id = 1, rec_version = 0, data = 'First value'
-- let's suppose this user want data = 'Flower value'

update test set
data = 'Flower value',
rec_version = rec_version +1
where id = 1 and rec_version = 0;
-- rec_version should be equal to when we got data to edit
-- id = 1, rec_version = 1, data = 'Flower value'

-- First user return from coffe
-- try update

update test set
data = 'Second value',
rec_version = rec_version +1
```

where id = 1 and rec_version = 0;

-- when checkin updated records we found 0
-- tell user that another user has changed the record
-- reload new data to replay

No locks to the database biggest concurrency possible, when updating more than one table, use transactions and the same
behavior.

When the update involve more than one record at randon, I use to have one record as semaphore.

I hope that this can help you and others !

___