

Forum:[Hack Forums](#)

Thread: [Reservation system, handling dates?](#)

severos - April 30, 2018, 8:38 am

Long story short, I'm writing a reservation system for a warehouse, the warehouse has shelves all have the same size, so you can think of it like a 3D array. My current implementation uses a 3D bool array where false means the shelf is free, and true means it's taken, on start shelves data is loaded from DB into the manager class, all assigning of shelves is done using this class as 2 steps:

1- Check availability: user enters the # of storage spaces he wants, if they're available, he gets an ID for that transaction where he can confirm during a window of 10 minutes, after that the ID is invalidated and he'll have to check again.

2- Confirm: this is where the DB is updated and users now owns those spaces.

This implementation is good enough as a start to get a "working" API for others to use in development, however it sustains from serious business problems:

1- If someone reserves storage space with starting date after 2 months, the storage space shows as reserved immediately (due to bool being set to true)

2- If the warehouse is full or the user orders a # of storage spaces larger than currently available (but it will be available at the date the order is effective) it shows as can't be processed since currently the bool array doesn't have enough empty spaces.

Anyone have some solutions and maybe sample implementation for this?

I've already thought of querying DB on every order and building a "state" of the warehouse on everyday of the renting period and checking if there's available space, however this will look super ugly, harm the memory, put load on DB, and it'll be hard to track where each item is going since shelves may need to be changed during the period.

Dalvik - April 30, 2018, 9:30 am

How about posting your current implementation so we can give comments on it?

(Apr 30, 2018 08:38 AM)severos Wrote: [\[CITE\]](#) I've already thought of querying DB on every order and building a "state" of the warehouse on everyday of the renting period and checking if there's available space, however this will look super ugly, harm the memory, put load on DB

A query on a user-action is not 'bad' for the database. Databases are built for storing and retrieving large amounts of data. Imo this is fine.

severos - April 30, 2018, 9:44 am

(Apr 30, 2018 09:30 AM)Dalvik Wrote: [\[CITE\]](#) How about posting your current implementation so we can give comments on it?

A query on a user-action is not 'bad' for the database. Databases are built for storing and retrieving large amounts of data. Imo this is fine.

I'll post the code in couple hours soon as I'm home.

As for querying DB, it'll add delay specially that the manager class uses lock, also the main problem would be in building the state after the query, so if the warehouse has 1k shelves, then we're building 3D array with 1k elements per request... I don't think that's a friendly way of doing it right?

Dalvik - April 30, 2018, 5:45 pm

(Apr 30, 2018 09:44 AM)severos Wrote: [\[CITE\]](#) I'll post the code in couple hours soon as I'm home. As for querying DB, it'll add delay specially that the manager class uses lock, also the main problem would be in building the state after the query, so if the warehouse has 1k shelves, then we're building 3D array with 1k elements per request... I don't think that's a friendly way of doing it right? I still have a hard time painting a picture of what you are doing. I will wait until you post your implementation before I make any further comment.

severos - April 30, 2018, 9:28 pm

Here's the current implementation:

PHP Code

```
<?php
public class SingleWarehouseStorageService
{
    private readonly object _lock;
    private readonly ApplicationDbContext _context;
    private readonly int _warehouseId;
    private readonly Config _config;

    private bool[,] _usedStorage;
    private Dictionary<string, (Task task, List<(int x, int y, int z)>>)> _tempTasks;

    public PriceSchema PriceSchema => _context.Warehouses.Where(o => o.Id == _warehouseId)
    )
        .Include(p => p.PriceSchema).First().PriceSchema;

    public SingleWarehouseStorageService(ApplicationDbContext context, Config config,
    int warehouseId)
    {
        _config = config;
        _lock = new object();
        _context = context;
        _warehouseId = warehouseId;
        Refresh();
    }

    async void Refresh()
    {
        while (true)
        {
            LoadFromDb();
            await Task.Delay(new TimeSpan(0, _config.WarehouseCacheMinutes, 0));
        }
    }

    public void LoadFromDb()
    {
        var now = DateTime.UtcNow;
        var config = _context.Warehouses.Find(_warehouseId);
        var spaces = _context.Spaces.Where(s => s.EndDate > now);
        spaces.Load();
    }
}
```

```

lock (_lock)
{
    _tempTasks = new Dictionary<string, (Task task, List<(int x, int y, int z)> idx)>();
    _usedStorage = new bool[config.MaxX, config.MaxY, config.MaxZ];
    foreach (var space in spaces)
    {
        if (_usedStorage[space.X, space.Y, space.Z])
            throw new Exception("This should never happen, 2 items in same storage space...");
        _usedStorage[space.X, space.Y, space.Z] = true;
    }
}

public string TempAllocate(int count)
{
    lock (_lock)
    {
        var testStorage = new bool[_usedStorage.GetLength(0), _usedStorage.GetLength(1),
            _usedStorage.GetLength(2)];
        var list = new List<(int x, int y, int z)>(count);
        for (int i = 0; i < testStorage.GetLength(0) && count > 0; i++)
        {
            for (int j = 0; j < testStorage.GetLength(1) && count > 0; j++)
            {
                for (int k = 0; k < testStorage.GetLength(2) && count > 0; k++)
                {
                    testStorage[i, j, k] = _usedStorage[i, j, k];
                    if (testStorage[i, j, k])
                        continue;
                    testStorage[i, j, k] = true;
                    list.Add((i, j, k));
                    count--;
                }
            }
        }

        if (count != 0)
            return null;
        _usedStorage = testStorage;
        var id = Guid.NewGuid().ToString();
        _tempTasks.Add(id, (Task.Delay(new TimeSpan(0, _config.OrderTimeoutMinutes, 0)).
ContinueWith(task => FreeTempAllocate(id)), list));
        return id;
    }
}

public void FreeTempAllocate(string id)
{
    lock (_lock)
    {
        if (!_tempTasks.Remove(id, out var result))
            return;
        foreach (var (x, y, z) in result.Item2)

```

```

        {
            _usedStorage[x, y, z] = false;
        }
    }
}

public List<StorageSpace> ConfirmAllocate(IEnumerable<ConfirmAllocationRequest> requests
, int orderId)
{
    var result = new List<StorageSpace>();
    lock (_lock)
    {
        var total = 0d;
        var schema = PriceSchema;
        foreach (var request in requests)
        {
            if (!_tempTasks.Remove(request.Id, out var entry))
                throw new Exception("Couldn't find id...");
            result.AddRange(entry.Item2.Select(location =>
                {
                    total += (schema.BaseCost + (request.EndDate - request.StartDate).Days * schema.
DailyRate) *
                        (1 + schema.TaxPercent);
                    return _context.Spaces.Add(new StorageSpace
                    {
                        X = location.x,
                        Y = location.y,
                        Z = location.z,
                        StartDate = request.StartDate,
                        EndDate = request.EndDate,
                        OrderId = orderId,
                        WarehouseId = _warehouseId,
                        PriceSchemaId = schema.Id
                    }).Entity;
                }).ToList());
        }

        var order = _context.Orders.Single(o => o.Id == orderId);
        order.Price = total;
        _context.Orders.Update(order);
        _context.SaveChanges();
    }

    return result;
}
}

```

severos - May 1, 2018, 8:58 pm

(May 1, 2018 02:47 PM)Orka Wrote: [CITE](#) An immediate (but naive) solution would be to replace the 3D array as type DateTime, using null to represent a vacancy. You should really consider redesigning as a proper DB since you're implementing concurrency control and order tracking yourself.

I would table reserves separately and consult them when determining if an order can be accommodated. This way you don't face overbooking or false rejections.

I'm pretty open to any solution, the reason I'm handling concurrency my self is to reduce calls to DB and building the model on every request.

ppaida - May 2, 2018, 9:38 pm

You should use a database to persist data state. In memory is ephemeral. Is your data important enough that if the app throws an exception, the power goes out, that the reservation data doesn't need to be persisted?

You should profile your database before being concerned about database load. Regardless, you can eager load the entire DB to mem if you want and there wouldn't be any trips to the DB from read.

Refactor your 3D array into business classes/objects:

```
public class Warehouse
{
    public ICollection<Shelf> Shelves {get;set;}
    public int GetAvailableShelves(Reservation reservation) { // find available shelves }
}

public class Shelf
{
    public int Row {get;set;}
    public int Col {get;set;}
    public Reservation {get;set;}
    public Available => Reservation == null;
}

public class Reservation
{
    public DateTime StartTime {get;set;}
    public DateTime EndTime {get;set;}
    public string UserId {get;set;}
}
```

severos - May 2, 2018, 10:56 pm

(May 2, 2018 09:38 PM)ppaida Wrote: [\[CITE\]](#) You should use a database to persist data state. In memory is ephemeral. Is your data important enough that if the app throws an exception, the power goes out, that the reservation data doesn't need to be persisted?

You should profile your database before being concerned about database load. Regardless, you can eager load the entire DB to mem if you want and there wouldn't be any trips to the DB from read.

Refactor your 3D array into business classes/objects:

```
public class Warehouse
```

```
{
    public ICollection<Shelf> Shelves {get;set;}
    public int GetAvailableShelves(Reservation reservation) { // find available shelves }
}
```

```
public class Shelf
{
    public int Row {get;set;}
    public int Col {get;set;}
    public Reservation {get;set;}
    public Available => Reservation == null;
}
```

```
public class Reservation
{
    public DateTime StartTime {get;set;}
    public DateTime EndTime {get;set;}
    public string UserId {get;set;}
}
```

Data is persistence in DB, I'm using memory to cache the result instead of querying DB over and over, the only data not being stored in a DB is checked but not confirmed, and it's not really a problem if this is deleted since client side can handle it by resending check.

The idea of refactoring DB have crossed my mind, but never found a good implementation for it, the implementation you provided will force me to create a shelf entry seeding, not that big problem, but it assumes that every shelf has 1 reservation, while in fact it has a history of reservations and only allowed a single active reservation.

ppaida - May 3, 2018, 7:07 am

(May 2, 2018 10:56 PM)severos Wrote: [\[CITE\]](#) Data is persistence in DB, I'm using memory to cache the result instead of querying DB over and over, the only data not being stored in a DB is checked but not confirmed, and it's not really a problem if this is deleted since client side can handle it by resending check.

The idea of refactoring DB have crossed my mind, but never found a good implementation for it, the implementation you provided will force me to create a shelf entry seeding, not that big problem, but it assumes that every shelf has 1 reservation, while in fact it has a history of reservations and only allowed a single active reservation.

You could just as easily do:

```
public class Shelf
{
    public int Row {get;set;}
    public int Col {get;set;}
    public ICollection<Reservation> Reservations {get;set;}
    public bool IsReserved(DateTime start, DateTime end)
    {
        return Reservations.Any(//whatever your criteria is for being reserved);
    }
}
```

