बैच जॉब या एक-एक करके

आज हमारे अपार्टमेंट में पानी की आपूर्ति में कुछ समस्याएं आईं और कुछ समय के लिए पानी नहीं आया। खाना खत्म करने के बाद और अपने परिवार के सदस्यों और मेरे द्वारा छोड़े गए बर्तनों के ढेर को देखकर, मैंने कई सवाल सोचे।

एक यह है कि पानी की आपूर्ति के बिना डिशवॉशर को कैसे काम करने दिया जाए। इसे एक बाल्टी पानी से जोड़ने के लिए डिज़ाइन किया जा सकता है। इसके अलावा, पानी की पाइप का जोड़ने वाला हिस्सा लचीला होना चाहिए ताकि सार्वजनिक पानी की आपूर्ति से निजी, स्वयं बनाए गए बाल्टी पानी में आसानी से स्विच किया जा सके।

एक और मुद्दा यह है कि काम को बैचों में करना है या एक-एक करके। हम हर भोजन के बाद बर्तन धो सकते हैं या एक दिन या कुछ दिनों के बाद उन्हें धोने के लिए इंतजार कर सकते हैं। यह इस दृष्टिकोण से है कि हम भोजन कैसे करते हैं और बर्तन कैसे धोते हैं। हम इस समस्या को इस दृष्टिकोण से भी देख सकते हैं कि एक डिशवॉशर में कितने बर्तन आ सकते हैं।

यह मुझे प्रोग्रामिंग की याद दिलाता है। हम कार्यों को बैच में या एक-एक करके कर सकते हैं।

बैचों में काम करने से एक स्पष्ट समस्या उत्पन्न होती है: इसके लिए अधिक संसाधनों की आवश्यकता होती है। जैसे-जैसे हम बर्तन धोने में देरी करते हैं, हमें अधिक बर्तनों की आवश्यकता होती है, और जैसे-जैसे हम डेटा को संचित करते हैं ताकि उसे बाद में संभाला जा सके, हमें अधिक मेमोरी स्पेस की आवश्यकता होती है।

वास्तविक जीवन में, एक साथ कितनी जगह या कितनी वस्तुओं को संभाला जा सकता है, इसकी एक सीमा होती है। उदाहरण के लिए, डिशवॉशर शायद अधिकतम बीस बर्तनों को ही संभाल सकता है, ठीक वैसे ही जैसे किसी कंप्यूटर पर एक प्रोग्राम की मेमोरी सीमा होती है या सड़क पर एक साथ गुजरने वाली कारों की संख्या की सीमा होती है।

इसके अलावा, यह समस्या भी है कि काम को कैसे अलग किया जाए। क्या हमें इसे एक समय में एक आइटम अलग करना चाहिए या एक समय में तीन आइटम अलग करने चाहिए?

डिश या कारों के लिए, प्रत्येक आइटम को एक इकाई के रूप में मानना सरल है। इसका मतलब है कि एक डिश एक डिश है, और एक कार एक कार है। आमतौर पर, उन्हें छोटे टुकड़ों में तोड़ा नहीं जा सकता है। हालांकि, अभी भी अपवाद हैं, जैसे कि एक बड़ा ट्रक जो कई कारों को ले जाता है; एक बड़े ट्रक को एक बड़ी इकाई और सड़क से गुजरने वाली कई कारों में तोड़ा जा सकता है।

प्रोग्रामिंग में, यह बहुत अधिक लचीला होता है। यहां तक कि एक इंसर्ट या अपडेट □□□ को भी छोटे-छोटे टुकड़ों में तोड़ा जा सकता है, डाउनलोड जॉब, □□□ सर्च, या क्वेरी की तो बात ही छोड़ दें।

ठीक है, अब हमने हैंडलिंग यूनिट के बारे में सोच लिया है। फिर, सवाल यह है कि हमें एक बैच में कितनी यूनिट्स को प्रोसेस करना चाहिए। यह एक से लेकर कुल यूनिट्स की संख्या के बीच कोई भी संख्या हो सकती है।

यहाँ सवाल यह है कि क्या किसी कार्य के लिए बैचों की संख्या निर्धारित या गतिशील हो सकती है। जेनरेटिव □□ के लिए, इनपुट टेक्स्ट के कुल वर्ण लचीले होते हैं। इसमें कुछ संदर्भ सीमाएं या इनपुट सीमाएं होती हैं, लेकिन इसकी सीमा के भीतर यह लचीला होता है।

डिशवॉशर का उपयोग करते समय, इसकी आंतरिक जगह की एक सीमा होती है। उस सीमा के भीतर, बर्तनों की संख्या लचीली होती है। हम आमतौर पर जितने बर्तन धोने की जरूरत होती है, उन्हें मशीन में डाल देते हैं।

प्रोग्राम्स के लिए, डेटाबेस को एक बार में हैंडल करने के लिए □□□ के बैच की एक सीमा होती है। उस सीमा के भीतर, यह कितने □□□ को हैंडल कर सकता है, यह लचीला होता है। लेकिन हमें □□□ को क्लाइंट से डेटाबेस सर्वर तक पास करने के नेटवर्क टास्क, यूजर कितना समय इंतजार कर सकता है, और अगर बैच के एक यूनिट टास्क में विफलता होती है तो क्या होता है, इस पर विचार करना चाहिए।

इसलिए, एक बैच में हमें कितने यूनिट कार्य करने चाहिए, इस समस्या के लिए हमें नौकरी के लक्ष्य, डाउनस्ट्रीम कंज्यूमर या हैंडलर की सीमा, और विफलता की संभावना पर विचार करना चाहिए।

यह सोचने का तरीका कई चीजों पर लागू किया जा सकता है। मूल रूप से दो समस्याओं पर विचार करना होता है: यूनिट टास्क क्या है और हमें एक बैच में कितनी यूनिट्स को प्रोसेस करना चाहिए। इन समस्याओं पर विचार करके, हम एक इष्टतम समाधान पर पहुंच सकते हैं।