エピソード1

SQLは集合指向の言語と心得よう

とある開発現場で……

2 00X年のある日のこと、大阪のシステムインテグレータ、浪速（なにわ）システムズ株式会社（仮名）の一角で若手技術者の大道君と上司の五代さんが顔をつきあわせて悩んでいました。

五代　「遅い……」
大道　「遅いですね……」

問題はあるEC（Electronic Commerce、電子商取引）サイトの商品一覧画面。お客様がインターネットを通じてアクセスし商品カテゴリを選ぶと、該当するカテゴリの商品一覧をその時点の在庫数量とともに表示する、この商品一覧画面の表示が遅いことに困っていたのです。大道君は数日前からこの問題を解決しようと悪戦苦闘していましたが、いっこうに進展がありませんでした。今もちょっとした修正をしてあらためてテストしてみたところですが、相変わらず画面は返ってきません。

五代　「……来た。タイムは？」
大道　「3分です」
五代　「変わらんか……しゃあない、奥の手出したるか!!」
大道　「奥の手？」

いぶかしむ大道君の横で五代さんはあるところに電話をかけました。

1.1 本番システムの商品一覧画面が遅い！

長いこと大阪でシステム開発会社を営み、ECや生産管理などのシステム開発に携わってきた私、ジーワン・システムの生島勘富は、リレーショナルデータベース（以下、RDB）がらみで相談を受けることがよくありました。今回の話もそんな相談の1つで、話を単純化してはありますが、今からXX年前の実話です。

五代さんからの依頼を2つ返事で引き受けて状況を聞いてみたところ、画面自体はどこのECサイトにでもあるような何の変哲もない商品一覧でした。データ構造もとくに性能に影響しそうなところはなく、普通に作れば1秒もかからないだろうと思われました。それが3分もかかるというのは確かに異常です。

生島　「開発中も遅かった？」
大道　「いえ、本番のデータを食わせたら遅くなったんです」
生島　「開発用のデータは本番と同じ件数ある？」
大道　「ありません。1/100もないと思います」

これもよくあるパターンですので、次からは早めに本番と同じデータでテストしておくべきでしょう。

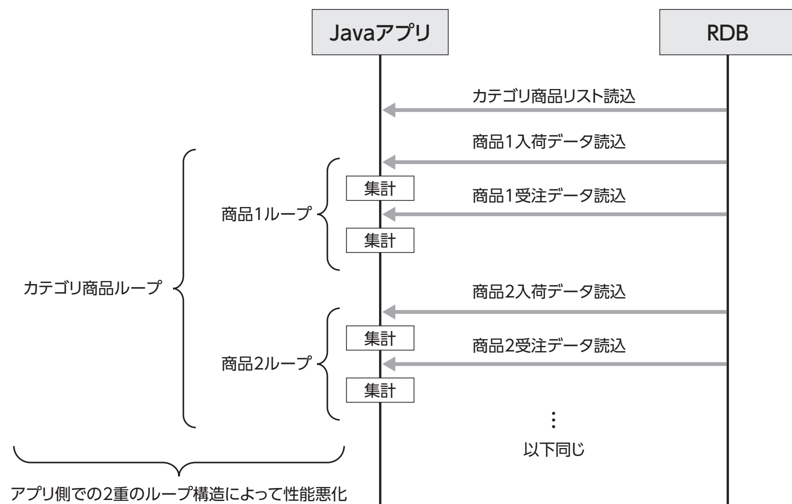
大道　「やっぱりそうですよね……次からはそうします」

お、素直なリアクション。若いときにこの姿勢はとくに大事。見どころあるじゃないか、と密かに思いつつ、次はプログラムを見せてもらいました。

1.2 原因はアプリ側でデータ集計を行っていたこと

プログラムを見ると原因はすぐにわかりました。在庫数量は入荷数量の合計から受注数量の合計を引くことで得られますが、要するにその集計処理をアプリケーション（以下、アプリ）側で行っていたのです（**図1-1**）。

図1-1 怒濤の二重ループ構造による性能悪化

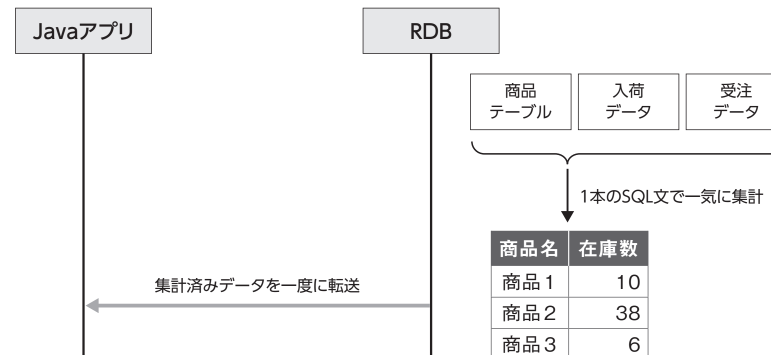


生島　「ふむ……問題はこれやな」
大道　「えっ、わかったんですか？」

と驚く大道君。まあ、この2週間ほど上司と2人で四苦八苦してダメだったのに、DBコンサルタントの生島とかいう知らないオッサンに30分もしないうちに「わかったでえ」とか言われても、にわかに信じられないことでしょう。それは無理もないことなので、その場で簡単なプログラムを作ってみせたところ、応答時間は3分から1秒以下に縮まりました。その差約200倍。大道君、口あんぐり状態。

本来この集計処理は**図1-2**のようにデータベース（以下、DB）側で1本のSQL文で行い、集計済みのデータを一度にアプリ側に転送するべきものでした。

図1-2 本来はSQL文一発で読んでこなければならない



商品名	在庫数
商品1	10
商品2	38
商品3	6

それをアプリ側のループ処理でやろうとすると、ループの回数分だけSQLを発行することになるため、何百何千……下手をすると何千万回ものSQL文が飛ぶことになり、①SQL文の送信・パース・コンパイル回数、②DBサーバとAPサーバ間のデータ転送量、③AP側の実行命令数、という3つの点で不利になり、負荷が重くなりやすいのです。

1.3 なぜアプリ側でSQL発行ループを書こうとしてしまうのか？

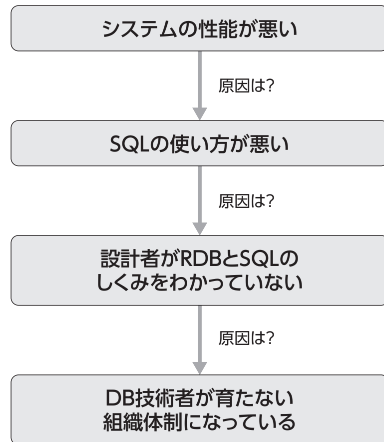
さてこのパターン、アプリ側で多重にループを回してその中でSQL文を発行することになるので怒濤（どとう）の多重ループ問題とでも呼んでおきましょう。単純ですが性能トラブルの原因としてよく見かけるものです。ひどいときには7重ループで実装されていたこともありました。原因はすぐにわかりましたが、問題はなぜこれを自力で解決できず、私のところに相談が回ってきたのかということです。率直に言ってこの多重ループ問題、RDBとSQLの基本を知っていれば起こすはずがありませんし、たとえ起きてもすぐわかるはずなのです。それがわからなかった、ということは……

RDBとSQLの基本を知らずに設計・プログラミングをしているのか？

そう考えざるを得ない事例を、私は長年見てきました。現代のWeb系システム開発、とくに業務系の開発ではRDBは必ず使われると言って良いでしょう。にもかかわらず前述の事例のようなケースに限らず、RDBの基本をわきまえて

いないと考えざるを得ないような設計をよく目にします。そしてこれは根本的には会社の組織体制に問題がある、と私は考えるようになりました（**図1-3**）。

図1-3　DB技術を重視しない組織体制が根本原因



「システムの性能が悪い」問題を受けて原因を調べると「SQLの使い方が悪い」部分が見つかります。コードは直せば動きますが、実際には「設計者がRDBとSQLのしくみをわかっていない」場合は同じ失敗を繰り返しますので、教育が必要です。しかし、現実には「DB技術者が育たない組織体制になっている」会社が多いのです。

生島　「こういう多重ループはアカン、て、誰か教えてくれへんかった？」
大道　「いえ、誰も……」
生島　「SQLは集合指向の言語やって、聞いたことないか？」
大道　「集合指向？　なんですかそれ？」

この答えが象徴しているように、きちんと教えてくれる先輩が身近にいなかったわけですね。技術というのは日々の実践で向上していくものなので、社外に1日2日の研修に出すだけでは限界があります。教育をするにしてもそれが可能な組織体制でない限り成果は上がりません。とはいえ、組織体制を変えるには会社を動かす必要があります。それに対して勉強だったら自分がやれば済むことなので、エンジニア自身で1人でもできます。というわけで、本書ではまずDB技術を学びたいエンジニアに役立つ情報提供をしていきます。まずは集合指向言語と手続き型言語の違いを押さえておきましょう。

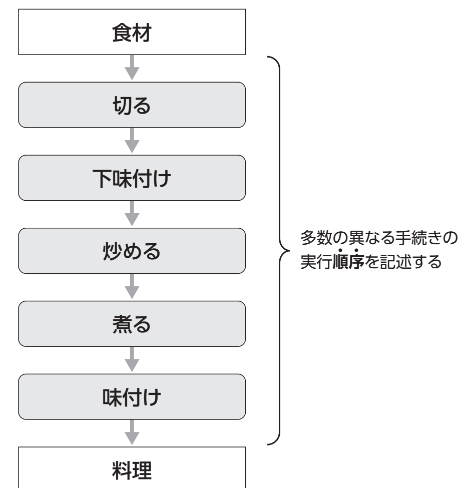
1.4 SQLは「集合指向」言語です

現在のIT技術者がおもに使うプログラミング言語というと、C/C++、Java、JavaScript、C#、Python、PHP、Ruby、Visual Basicなどが挙げられますが、これらはいずれも「手続き型言語、またはそれを発展させたオブジェクト指向言語」であり、コードの細部は手続き型の考え方を基本としています。一方、RDBへの問合せに使うSQLは「集合指向」という、手続き型とは根本的に異なる設計思想を持った言語です。これが、一般のIT技術者にとってSQLの理解を難しくしている原因なのです。では、手続き型と集合指向ではどのように違うのでしょうか？

手続き型言語では実行順序を記述する

図1-4に食材からカレーやシチューのような料理を作る作業をイメージしたフローを示しました。

図1-4　手続き型言語の処理モデル

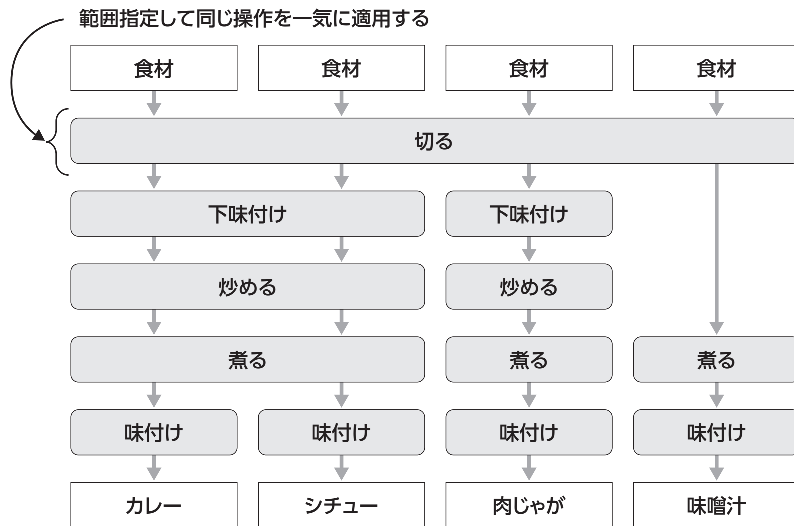


手続き型言語が想定しているのはこのように「多数の異なる手続きの実行順序を記述する」ことです。そしてその際よく出てくる特定のパターンを簡潔に書くために、「判断」「繰り返し」のような制御構造、あるいはサブルーチン化、例外処理、オブジェクト指向といった仕様が導入されてきました。その具体的な仕様は個々の言語によって違いますが、「実行順序を記述するものである」という基本は手続き型言語に共通しています。

SQLでは範囲指定で同じ操作を一括適用する

一方、SQLが想定しているのは**図1-5**のようなモデルです。

図1-5　集合型言語（SQL）の処理モデル



こちらは、ホテルの宴会場のような場所で、何種類ものメニューを一気に大量に作る場面をイメージしてください。最終的にできる料理がカレー、シチュー、肉じゃがと違っていても、中には共通の部分があります。たとえば「食材を一口大に切る」ところはすべての料理に共通だから一括してできるし、カレーとシチューについては味付け以外は共通なのでまとめられるでしょう。SQLはこのような場面で「範囲指定して同じ操作を一括適用する」ための言語なのです。手続き型言語は**図1-4**の縦方向の処理、SQLは横方向の処理に向いているわけです。

こうした違いが典型的に表れるのが、「ループ」です。

リスト1-1と**リスト1-2**のコード例はいずれも「注文データを集計して金額の合計、最大、平均を出す」という想定のものですが、手続き型言語の例（**リスト1-1**）ではsum、maxやiといった作業変数とif文、for文を使ったループ処理があり、SQLの例（**リスト1-2**）ではそれがないことに注目してください。

リスト1-1　手続き型言語（Java）での集計処理
（orders配列のBillingの合計、最大、平均値を算出）

```
int sum = 0, max = 0, avg = 0;
for(int i = 0; i < orders.length; i++){
  sum += orders[i].Billing;
  max = (max > orders[i].Billing) ? max : orders[i].Billing;
}
if(orders.length > 0) avg = sum/orders.length;
```

いくつもの作業変数、判断／ループ処理が必要で、バグが入り込みやすい

リスト1-2　集合指向言語（SQL）での集計処理
（orderテーブルのBillingの合計、最大、平均値をcustomer_idごとに集計）

```
SELECT customer_id, sum(Billing) , max(Billing), avg(Billing)
FROM order
GROUP BY customer_id;
```

作業変数も判断もない単純なコードなので、バグが入りにくい

SQLでは「同じ性質を持ったデータの集合に対して同じ操作を一括適用」するのが基本です。ループ制御構造を記述する必要がない分、SQLの例のほうが簡潔に書けていることがわかりますね。制御構造が不要ということはバグも入りにくいということです。

しかも、SQLのコード例が「customer_idごとに分類して集計」しているのに対して、手続き型言語のコード例では全体を集計しています。もし手続き型言語でもcustomer_idごとに分類しようとして、そのために連想配列を使わずにもう一段ループをかますと……はい、こうして「怒濤の多重ループ構造」が発生するわけです。

設計思想が違うものは理解しにくい

リスト1-2のSQLには作業変数もループもない、ということにあらためて注目してください。「変数」や「ループ」と言えばプログラミングの基本中の基本