第1章

10年戦えるデータ分析の技術

この章では、本書の最初の章として、本書全体を貫くデータ分析の方針……すなわち「SQL で分析せよ」というテーゼについて説明します。

1.1 現実的な分析の方法とは

昨今、分析あるいはデータ活用という単語は大流行しています。分析を活用して売り上げを 50% アップ！ とか、データサイエンティストが 21 世紀で最もセクシーとか、IoT で O2O で顧客動線を最適化してなんちゃらという複雑なシステムまで、データを活用して業績を上げましたという話には枚挙にいとまがありません。

そんな説明をされるといかにもすごそうですし、自分の会社でもやってみよう、そろそろやってみないとまずいかな、と思うのが自然のなりゆきです。よし、じゃあうちでも眠っているデータを活用しよう！ ビッグデータの分析システムを入れよう！ いやさすがにそれはちょっと金かかりすぎだからまず Excel からかな……などと皮算用が始まってしまうのもやむをえないことなのです。

ところが実際に企業でいざ分析をしようとしてみると、分析を活用するまでの間には実に深く広い谷が広がっていることに気付くでしょう。すなわち……

- データが会社のあちこちに散逸していて、まず集めるのに一苦労
- 集めたはいいが、データの意味がわからない
- データの意味を教えてもらったら、そもそも「ユーザーID」がウェブの店舗と実店舗で違うことがわかってきた
- よくよくデータを調べてみると、あっちとこっちとでつじつまが合わない
- 最初はシステム部の人に個人的にお願いしてデータを出してもらっていたが、頻度が増えてさすがにいい顔をされなくなってきた
- すべての障壁を乗り越えてついに分析を始められたが、データがこまぎれになっていて Excel で分析するのが面倒すぎる
- データが 10 万行を超えたあたりから Excel が遅くてどうにもならない

分析の手法だとかデータの種類の話をする以前に、そもそもまともに分析を始

められない、始めても完了できない、そんな現実が待ち受けているわけです。

どうしてこうなってしまうのでしょうか？ すべてはシステム部が無能なせいでしょうか？ それとも分析をしようとしたマーケ部員のやりかたがまずかったせいでしょうか？

たぶんそのどちらでもありません。いま挙げた問題は特定の企業によらず構造的に発生しがちな問題であり、誰がやっても乗り越えなければならない壁なのです。この壁を乗り越えるためには、分析をしたい人（技術者かもしれないがそうでないことも多い）と、システムを作っている人（技術者）が同じ１つの方向を向き、力を合わせて進む必要があります。

■ SQL による分析こそ現実的

では、その向かうべき方向とはどこでしょうか。本書が全力をもってお勧めする現実的な分析手法とは、ズバリ、SQL の使えるデータベースを使ってデータを分析することです。

SQL の使えるデータベースの代表格と言えば**リレーショナルデータベース管理システム**（RDBMS, relational database management system）がまず挙げられます。RDBMS とは、「表」のような形式で大量のデータを高速に格納・操作できるコンピューターシステムです。代表的な RDBMS としては、Oracle、SQL Server、MySQL、PostgreSQL などが挙げられます。

そして **SQL**は、RDBMS に格納されたデータを処理するためのプログラミング言語です。次のような見ためをしています（意味は後ほど説明します）。

```
select
    count(*)
from
    access_log
where
    request_time between date '2014-11-16' and date '2014-11-23'
    and request_path = '/sale201411'
;
```

この SQL 文は、ウェブのアクセスログをもとに、2014 年 11 月 16 日～ 22 日の期間に /sale201411 というページがアクセスされた数を数えています。個々の記述の意味はわからないと思いますが、次の章から詳細に説明するのでいまはわからなくて構いません。とりあえずは、データを貯める RDBMS というシステムがあり、そこに貯めたデータは SQL という言語で操作できるのだということがわかれば、いまのところは十分です。

では、あらゆるタイプのデータベースと様々な分析手法が氾濫する中で、なぜ SQL をお勧めするのでしょうか。理由は 3 つあります。

1. 企業のほとんどのデータは RDBMS か Hadoop にある
2. サイズの制約がない
3. エンジニアとプランナーの共通言語として優れている

順番に詳しく説明します。

■ 1. 企業のほとんどのデータは RDBMS か Hadoop にある

RDBMS は IT 業界標準の、そして現実的にほぼ必須の、データベースシステムです。ほとんどすべての業務システムは RDBMS を使って構築されています。言い換えると、企業において業務から発生するデータはまず間違いなく RDBMS 上にあるということです。

またもう一方の **Hadoop** は、主に巨大なデータを格納するために使われるデータベースシステムです。ウェブのログなどのデータは Hadoop に入っていることが多いでしょう。

RDBMS と Hadoop の共通点は、SQL を使えるということです。つまり SQL が使えれば、どんな現場のどんなシステムからも情報を引き出すことができるということになります。しかもこの状況は当分変わりそうにありません。この使い勝手のよさは圧倒的です。

また、どのような分析をするにも、まずはデータがなければ始まりません。そして現場においては「そもそもデータがあるかどうかわからない」という状態か

ら始まることも日常茶飯事です。つまり、データベースを自分で調査できる能力がなければ、データがあるのに使えないという可能性もあるわけです。

そのような場合も、SQL の知識があれば心配はありません。ある程度の制約や、守らなければならない約束事はありますが、それさえ守れば自分でデータベースに接続して、実データを見てデータを探索することができます。

■ 2. サイズの制約がない

SQL による分析は、他の分析手法と比べて非常にデータサイズに制約が少ない手法です。小さいサイズから手軽に使うことができる一方で、100 万件、1 千万件、1 億件になっても同じ手法で分析でき、最終的に対象がビッグデータになろうと変わらず分析することができます。

例えばお手軽分析と言えばまず Excel ですが、Excel が直接扱えるデータサイズは 100 万行が上限です。手元のパソコンで動かす前提だと、現実的な上限はもっと少ないでしょう。100 万行程度のデータはそれほど大きな企業でなくても越えてしまいますから、「わたしプログラマーじゃないし、Excel でいいよね」などと言っているとその瞬間に分析の道が断たれます。

では RDBMS ならばどうでしょうか。MySQL や PostgreSQL のようなオープンソースの RDBMS でも、現代の強力なハードウェアをもってすれば 1 千万行くらいはまあふつうに扱えます。ちょっと速度を我慢すれば 1 億行もなんとかいけます。

さらに Redshift や Teradata のような**並列リレーショナルデータベース管理システム**（parallel relational database management system）や Hadoop を使えば、上限はほぼなくなります。コンピューターの台数を増やすことで、データサイズの増加に対応できるからです。Excel や R、Python のような別の手法ではこう簡単にはいきません。

うちの会社でそんな巨大なデータを扱うことなんてないよ、と思われるかもしれませんが、データは年々増加します。単純に、年数が経過して蓄積されることでも増えますし、会社が成長すれば年ごとのデータ量もどんどん増えます。分析が深まるに従って、扱いたいデータの種類も増えていくでしょう。そうなれば、

相対的にデータ分析はどんどん重い処理になるわけで、上限なく性能を向上できる仕組み、ソフトウェアがより重要になってくるわけです。

SQL は、分析する対象サイズに制限がないことによって、これからますます投資する価値の高い仕組みになるでしょう。

■ 3. エンジニアとプランナーの共通言語として優れている

SQL を使うべき 3 つめの理由は、「エンジニアとプランナーの共通言語として優秀だから」です。ここでは「プランナー」を「エンジニア以外の人」というくらいの意図で使っています。例えば企画や商品開発をする人や、分析を専門に行うデータアナリストなどを想定しています。

この章の最初に話したように、エンジニアが分析をすることもありますが、分析をしたい人はエンジニアだけではありません。むしろプランナーのほうが分析をしたいことも多いでしょう。つまり、分析をすべき目的と欲求はプランナーにしかない、分析をする技術はエンジニアにしかない……という状況に陥りがちなのです。そうなるとプランナーとエンジニアがチームを組んで分析を進めていかなければなりません。

しかし、およそ IT のプロジェクトに関わった人ならば、エンジニアとプランナーの間の意思疎通がどれだけたいへんかは経験があるでしょう。すり合わせによって最終的にうまくイメージを一致させられればまだいいですが、齟齬のあるままプロジェクトが進行してしまい、完成してから「ちょっと違うんだけど……」なんてことも往々にしてあると思います。

この「みんなで共通のイメージを持てない」問題の原因と対策は多岐にわたります。しかし根本的に解決するには、お互いに相手の領域を勉強し、理解するしかありません。つまり、エンジニアもビジネスの構造や KPI やマーティングを理解すべきだし、プランナーも IT システムの仕組みをある程度知っているべきだということです。

では IT システムを知ってもらううえで最も学習効果が高い分野は何でしょうか。それは断然 SQL です。なぜなら RDBMS はほとんどあらゆる業務 IT システムの中核であり、すべてのデータがそこにあるからです。プランナーに

RDBMS と SQL を学んでもらうことはすなわち、企業のデータを理解することとイコールです。その学習効果ははかりしれません。

また、より現実的でベタベタな理由として、「SQL がわかればプランナーが自分でデータを取ってくることができる」という理由もあります。

さて、こういうことを言うと確実に「プランナーには SQL なんてとても書けない」……という反論（愚痴？）が想定されます。しかしそれはプランナーをバカにしすぎた意見だと思います。

IT に関しては、プランナーをよく言えば「弱者」、悪く言えば「バカ扱い」する風潮があるように感じています。「これは IT 専門知識が必要で難しいからプランナーに直接扱わせるのは無理だからこちらでやろう、せめて ×× のインターフェイスをかぶせよう」という方向になりがちです。例えば「SQL はプログラミング言語だからプランナーに書かせるなんてとんでもない、いたれりつくせりの分析アプリを用意してあげないと無理だ」という具合です。

この意見が正しいときも確かにあります。わたしもすべてのプランナーが SQL を書けるとは思いません。しかし、理解できるプランナーも必ず一定数存在します。そして、そういう数少ない人々は確実に「デキる」ので、ビジネス上のキーマンです。「デキる」人間に IT システムを理解してもらうことは、必ず大きな意味を持つはずです。

1.2 分析 SQL の世界

ここまでは、SQL が最も現実的に最適な分析ツールだという話をしてきました。ですが、具体的な分析方法の話に行く前に、いくつか、たいへんありがちな誤解を解いておくことにしましょう。

プランナーのかたには聞き慣れない話もあるかもしれませんが、この節はどちらかというとすでに SQL を使っているエンジニア向けの内容なので、雰囲気だけわかってくれれば十分です。

■「SQL で分析なんてできるの？」への回答

「SQL で分析をするのだ」――と言うと、業務システムやウェブアプリを作っているエンジニアの皆さんは真っ先に Oracle や MySQL を思い出すのではないでしょうか。そしてもしかするとその次には、O/R マッパーやアプリケーションフレームワークや EJB や分散トランザクションがどうとか……と考えてしまうかもしれません。

しかし、このさいそういった「データベースの外の道具」はいったん忘れてください。本書で言う「SQL を使う」とは、文字通り自分で SQL を書くことによって利用する話であって、O/R マッパーやフレームワークを経由して間接的に SQL を使う話ではありません。純粋に SQL のみで分析をしようというのが本書の方針です。

そしてそんなことを言うと必ず「え、SQL だけでそんなことできるの？」と言い出す人が出るので最初に断言します。可能です。実際にそれだけで分析をしている人たちが存在するので、それはもう事実です。信じられなければ本書の続きを読んでください。

■ 業務システムの世界と分析の世界

なぜこんなことをわざわざ言うのでしょうか。実は SQL と一口に言っても SQL のまわりには 2 つの世界が広がっており、業務システムやウェブシステムを作っているエンジニアがよく見知っているのも、話題になりやすいのも、その片方だけだからです。

2 つの世界とは何でしょうか。それはズバリ、「基幹系の世界」と「情報系の世界」です。これから本書で話そうとしているのは後者における分析の話であり、前者の話はあまり関連がありません。

ちなみに、特にウェブ界隈で育ってきたエンジニアを見ていると、「RDBMS はデータを格納するただのストレージ」と考える傾向が強いと感じています。言い換えると、「RDBMS からアプリにデータを取ってきて、計算して、書き戻す」というスタイルが唯一の RDBMS の使いかただと考えている、ということです。

これはウェブの世界においては確かに正しく、適切な考えかたです。しかし分析の世界はそもそも文脈が異なるので、もはや正しくありません。

■ 基幹系の中心となる OLTP とはどんな処理か

ではいわゆる基幹系の、業務システムやウェブアプリケーションはどのような特徴を持つシステムなのでしょうか。これらは、単純化して一言で言うと **OLTP**（online transaction processing）システムと言ってよいでしょう。OLTP という処理の特徴は、「オンライン」で「トランザクション」を大量に処理する必要があることです。

オンラインで、というのは、システムを使うユーザーはリアルタイムに回線の向こうにいる、という意味です。例えばウェブアプリケーションを考えましょう。ユーザーがスマホやパソコンを使ってウェブブラウザーを操作すると、リクエストが「即座に」ウェブアプリケーションに送られて、ウェブアプリケーションは「できるだけすぐに」応答を返します。これが「オンラインである」ということです。

「トランザクション」のほうはもう少し抽象的で、何らかの取引のことです。

普段多くの人が接しているトランザクションとしては、商品の注文、チケットの予約、現金の引き落としや送金、などが挙げられます。これらのトランザクションはユーザーごと・取引ごとに独立していて、たいていは細かいトランザクションが大量に発生します。

まとめると、OLTP システムとは、小さな取引（トランザクション）を、リアルタイムにその場で大量に処理するシステムということになります。この用語は古い時代にできた用語なので「トランザクション」と言われてもピンとこないでしょうが、ウェブアプリケーションは特徴が完全に OLTP なので、ウェブはようするに OLTP だと思っておいてよいでしょう。

■ 分析とはどんな処理か

では、その一方で、分析処理にはどんな特徴があるのでしょうか。トランザクションと比較したときの分析処理の特徴は 3 つです。

まず、分析処理では一般に、何らかの集計が発生します。例えばチケットの予約を例にして考えてみると、「1 日ごとの予約件数を数える」「ユーザーの年齢層ごとの割合を計算する」「1 日のうち予約件数が多い順に時間帯を並べる」などが分析処理です。これらの処理はいずれも、大量の予約データを集計する必要があります。

次に、分析処理はほとんどデータを書き込みません。さきほどの例を見てもわかるように、分析処理の主眼は OLTP システムによって発生したデータを集計することにあります。つまりどこからもデータが発生しないので、新たにデータを記憶しておく必要がないのです。

最後に、分析処理では非常に大量のデータ集計が発生する場合があるので、処理時間も長くかかる可能性があります。そのため同じオンラインシステムではあっても、応答時間のリミットが OLTP システムより長く設定されている場合がほとんどです。

以上の特性を**表 1.1**にまとめました。

表 1.1 OLTP と分析処理の特徴比較

項目	OLTP	分析処理
アクセスパターン	読み書き	読み込みのみ
1 回のアクセス量	少ない	多い
アクセス数	多い	少ない

■ 分析向けの RDBMS

ここで RDBMS に話を戻します。

さきほど典型的な RDBMS として Oracle や MySQL などを挙げましたが、実はこれらはいずれも元は OLTP を想定して設計された RDBMS です。分析に使えないわけではないのですが、特に向いているわけでもありません。特に名指しするなら MySQL は絶望的です。ウェブエンジニアが分析 SQL に疎いのは MySQL のせいではないかと疑いたくなるほどです。

いま話してきたように、OLTP と分析処理では処理の特性がまったく異なります。当然ながら、データベースへのアクセスもそのような特性を反映しています。OLTP システムではそれぞれ独立した少量データの読み書きが大量に発生しますし、分析処理システムではデータ全体にわたる大規模な読み込みが少量発生します。これだけ特性が違えば同じ 1 つのデータベースで処理できるわけがないのが当然でしょう。

では、Oracle や MySQL 以外の、分析処理に特に向いた RDBMS というものもあるのでしょうか？ もちろんあります。そのような RDBMS の例としては、Teradata 社の Teradata Database（テラデータ）、IBM 社の IBM Netteza（ネティーザ）、Pivotal 社の Pivotal Greenplum Database（グリーンプラム）、Amazon 社の Amazon Redshift（レッドシフト）などが挙げられます。

これらの RDBMS はいずれも複数のコンピュータ（ノード）を使った **shared nothing** と呼ばれる並列処理に対応しており、大量のデータを分析することに特化した RDBMS です。基本的に高価で、膨大な量のデータを持っている企業しか導入していないので知名度もイマイチでした。以前は Teradata くらいしかない市場だったのですが、2009 年ごろから競争が激化し、現在は以前に比べるとだいぶ多くの人が知るようになりました。

■ バッチ処理とは

この節の最後に、オンライン処理と対応する用語として**バッチ処理**（batch processing）について説明しておきましょう。

オンライン処理はユーザーが実際に回線の向こうにいて、その場で応答を返さなければいけない処理のことでした。それに対してバッチ処理とは、対応するユーザーがおらず、たくさんのトランザクションや分析処理を（たいていは定期的に）一括処理する方式のことです。

例えば、チケット予約システムの例で再び考えてみましょう。チケットを予約するとき、ウェブや電話で問い合わせてすぐ予約が確定する場合もありますが、一度抽選して後日結果が知らされるという予約方法があると思います。このような場合、予約の受け付け処理は OLTP ですが、抽選処理は別です。期日が来たら、受け付けておいたすべての予約について、一気に抽選を行うことになるでしょう。この、抽選を一気に処理するという処理方式がバッチです。

他の例としては、メールマガジンの送付が挙げられます。メールマガジンはユーザーのリクエストを受けたその場で送るわけではないでしょう。「毎週水曜」とか「毎月 1 日」とか「毎日」のような特定のタイミングで、すべての読者にメールが配信されるわけです。このメール配信の処理もバッチです。

なお、注意してほしいのですが、バッチ処理はオンライン処理と対応するのであって、OLTP と対応するわけではありません。つまりどういうことかと言うと、「大量のトランザクションをバッチ処理する」場合もあれば、「大量データの分析をバッチ処理する」場合もあるということです。

また、分析処理もオンラインで処理される場合とバッチで処理される場合があります。オンラインの分析処理は特に **OLAP**（online analytical processing）と呼ばれます。表にまとめるなら**表 1.2**のような区分になるでしょう。

表 1.2　オンライン処理とバッチ処理

	トランザクション処理	分析処理
オンライン処理	OLTP	OLAP
バッチ処理	トランザクションバッチ	分析バッチ

■ すべてが SQL（を使ったシステム）になる

さて、この分類を踏まえて本書のテーマを言い直します。本書のテーマ、目的は、「すべての分析処理を SQL で行う」ことです。

「すべての分析処理」であって、「すべての OLAP」ではないことに全力で注意してください。OLAP は人がシステムをリアルタイムにさわって SQL を投げて分析する処理ですが、「すべての分析処理」となると、人が介在しない処理……つまりバッチも含むことになるのです。これは、分析システムのほぼすべてを SQL でまかなおうと言っているのと同じです。

ですがそうなると、人が分析をするための SQL と、バッチ処理のための SQL、両方が必要ということでしょうか。

それが、そうではないのです。バッチ処理でしか使わない SQL の機能というものもありますが、処理の 9 割は OLAP とバッチで共通しています。つまり、これから説明していく SQL の話は自分の手で分析をする場合に限らず、次の 3 つのシチュエーションすべてに通用するのです。

- 人がその場で SQL を書いて分析する場合
- 人がアプリケーションを使って間接的に SQL を実行する場合
- システムがバッチ処理で SQL を実行する場合

たぶんエンジニアのかたからは「バッチで SQL を使って大丈夫か、問題ないか、必要な処理をこなせるのか……」などなど様々な疑問・不安が浮かんでいることと思いますが、この点については第 2 部で改めて細かく議論するので心配ありません。

いかがでしょうか。SQL は、分析のありとあらゆる場面に応用が可能な万能ツールです。もはや SQL を使わない理由は見当たらないと言ってよいでしょう。

1.3 この本を読み進めるためのガイド

この章では、分析に対する本書のスタンスをお話ししました。

根幹となる方針は、「SQL ですべての分析処理を行うこと」です。なぜならSQL は標準であるがゆえに最も多くのデータにアクセスでき、サイズの制約もなく、さらに分析を本当に行うべき人（現場のプランナー）とエンジニアの双方が理解可能であり、橋渡しをすることができるからです。しかも特典として分析バッチを書くこともできます。

■ 本書の構成

本書の章構成は**表 1.3**と **表 1.4**のとおりです。

表 1.3　第 1 部の構成

章	内容
第 1 章「10 年戦えるデータ分析の技術」	SQL で分析すべき理由と、本書の概要
第 2 章「さわってみよう RDBMS」	テーブルやカラムなど RDBMS の基礎概念と接続方法
第 3 章「簡単！ select 文でデータ探索」	select 文の基礎
第 4 章「すべての分析は集計から始まる」	集約関数と group by 節を使った集計について
第 5 章「関数で自由自在に新しいカラムを作り出す」	関数と演算子を使って、他のカラムから関連する値を計算する方法
第 6 章「ジョインを制するものは RDBMS を制す――基礎編」	複数のテーブルを連結するジョインの基本とテーブル設計
第 7 章「ジョインを制するものは RDBMS を制す――応用編」	様々なジョインのバリエーションとアソシエーション分析
第 8 章「遅れて来た分析 SQL 最強の武器――ウィンドウ関数」	サブクエリーとウィンドウ関数
第 9 章「縦と横は難しい」	縦持ちテーブルと横持ちテーブルの変換
第 10 章「アクセスログのセッション分析をする」	章を通じて大きな分析に取り組む

1 章から 10 章までの第 1 部では、SQL の書きかたを具体的に解説し、SQL に

よる分析の進めかたを話していくことになります。つまり、分析システムを使う側の視点で話を進めていきます。

表 1.4　第 2 部の構成

章	内容
第 11 章「10 年戦えるデータ分析システム」	SQL を用いた分析システムのアーキテクチャ
第 12 章「ビッグデータに立ち向かう」	ビッグデータ技術を活用した分析システムの構成
第 13 章「SQL バッチの技法」	分析データベースにおいてデータを加工する手法
第 14 章「本書を読み終えた後に」	発展的なトピックについて

そして 11 章から 14 章までの第 2 部では視点を変えて、分析システムを作る側から話をしていきます。どのようなシステムを作るべきで、そのシステムをどう設計するか、アーキテクチャをどうすべきか、ということを話していきます。SQL でバッチ処理をこなすための技術についても説明します。

あなたがプランナーや初心者エンジニアなら、すべての章をこの順序で読むほうがよいでしょう。

ふだん RDBMS は使っているが分析で使ったことのないというエンジニアのかたは、4 章以降の自分の知らない内容が出てきたところから読み始めるのがお勧めです。

もし SQL に慣れているベテランエンジニアであれば……読むべきところは自分でわかると思うので、興味のあるところを好きな順でお読みください。