情報システムとWeb技術＃2 — データベースと2層/3層モデル

久野　靖*

2011.10.11

1　データベースとデータベース設計

1.1　データベースとその位置づけ

前回は情報システム全体の構造分割やデータの流れを DFD で記述するという話でしたが、今回は情報システム内でデータを永続的に記憶しておく場所であるデータベースについて取り上げます (「計算機科学」でのデータベースの回の内容程度は前提としていますが、多少復習はします)。

データベース (DataBase) とは「統合化された、共有可能な、データの格納場所」であり、その機能は裸のハードディスク (や SSD など) の上にデータベース管理システム (DBMS、DataBase Management System) が提供しています。

なぜ情報システムにおいてデータベースが大切かというと、情報システムのさまざまな場所で発生したり取得する必要のあるデータを保持するのには、「統合化された、共有可能な、データの格納場所」が必要だから、ということになります。たとえば Web 上の情報システムでは、Web サーバ上でユーザインタフェースやビジネスロジック (の一部) が動くとしても、最終的なデータはその背後にあるデータベースに格納されているわけです (図 1)。

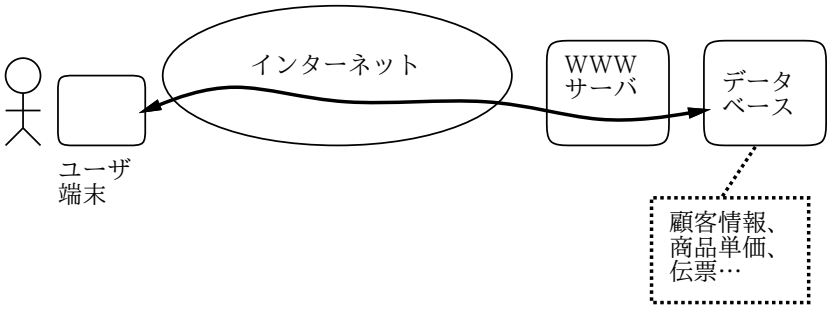

図 1: Web システムとデータベース

質問 1 データを格納するのであれば「ファイル」に入れておくことでも可能ではないでしょうか。なぜファイルではなくデータベースなのでしょうか?

1.2　DBMS が提供するさまざまな機能

では、より具体的に情報システムがデータベースの何を必要としているかというと、情報システムが扱うデータは失われてはならず、また矛盾や誤りがあってはならないので、DBMS が提供する並行制御、トランザクション、整合性管理などの機能が情報システムにとっては不可欠であり、これら

*経営システム科学専攻

を自前で実装するのはとても大変なので、DBMS で安定して実装されているものを利用する、ということでしょうか。

DBMS が提供するさまざまな機能について、一応列挙しておきます。

- 問い合わせ — さまざまな構造を持った検索を可能としてくれる
- 並行制御 — 複数のプロセスによるデータアクセスがあっても正しく処理が行われることを保証
- 排他制御 — あるプロセスによるアクセス中は他のプロセスによる干渉が無いことを保証
- 障害回復 — システムやアプリケーションの異常があっても整合性のある状態に復帰可能
- トランザクション — 複数の操作をひとまとまりにして原子的に実行可能とし、他の操作との干渉を排除
- 整合性管理 — データベース中のデータに関する整合性条件を指定しそれを維持させてくれる
- セキュリティ — アクセス制御により、権限のあるユーザだけがデータの操作を行えるよう保証

しかし個人的には、DBMS が提供するもっとも重要なモノは、「データモデル」であると思います。データモデルはデータをどのように構造化するかを決める「枠組み」であり、これによってデータに構造を持たせられるゆえに、1 つのデータをさまざまな箇所で共有できたり、更新に際して整合性の条件をチェックできたりするわけです。

具体的なデータモデルとしては、今日ではその汎用性、柔軟性、理論的背景がきちんとしていることから、関係モデル (Relational Model) が主流です。関係モデルに基づくデータベースや DBMS を RDB、RDBMS とも書きます。RDB には次のような利点があります。

- 標準化されたデータ操作言語 (DML、Data Manipulation Language) である SQL で検索/更新可能であり、DBMS ベンダに依存しないソフトウェア開発が可能である (細かくは違っている点もあるが…)。
- SQL は高レベルであり、また Access のようなフロントエンドとなるソフト経由で扱うこともできるため、カジュアルユーザによるオンデマンドでの柔軟な検索が可能。
- 高レベルな集合操作を基本としていて「どのような順番でデータを処理するか」を書かないので、内部で分散化や最適化がしやすい。

ただし、関係モデルではデータは基本的に数値や文字列などの基本型の集まりであるため、より複雑な構造を持つデータは扱いづらい面があります。このため、オブジェクト指向言語に対応した OODB などのモデルもありますが、思ったほどは普及していません。

また、関係モデルのようなきっちりしたデータモデルでは記述しにくいあやふやな情報を扱いたい、というニーズもあり、そのために半構造データ (Semistructured Data) を扱う研究も活発になされています。半構造データを扱うアプローチの 1 つが、汎用の標準データ表現である XML を格納し、XML に対する問い合わせをこれまた標準規格である XQuery などで表現するものです。このような XML を対象としたデータベースを (そのまんまですが)XML データベース (XMLDB) と呼びます。

一方実務面では、関係モデルにきっちり従うよりも、単に「キーとデータの組」が集まったものを格納する (Key-Value Store)、という程度の機能を提供することで、軽く高速な処理を提供するものもあります。これは SQL に基づかない、という意味で「NoSQL データベース」などと呼ばれていて、クラウドサービスなどで多く提供されています (クラウドのような分散システム上ではきっちり関係モデルを提供することのコストがかなり高いため)。

1.3 データモデリング

ここまで、データベースについての説明をしてきましたが、実際の情報システム構築に際しては (どのデータモデルを使うかはまず選択した後で)、「どのようなデータを持ったどのような構造のデータベースを作って使うか」を決める必要があります。これをデータベース設計と言います。

データベースを設計するということは、対象システムにどのようなデータがあり、それらのデータがどのような関係を持っているかをまず分析する必要があります。この作業をデータのモデルを作る、という意味で (まんまですが) データモデリングと言います。

では具体的に、データモデリングはどのように進めて行けばいいのでしょうか? つまり、あるシステムが必要とするデータやその間の関係はどうやって決めて行けばいいでしょうか?

1 つの明らかな方法は、もともとデータベースの用途は「アプリケーションから見てデータを格納する場所」だということに基づくものです。つまり、アプリケーションを開発するに当たって、それが必要とする (記録したり参照する) データを列挙し、それをもとに表を作る、というものです。

これでもまだ抽象的だと思うのなら、実際にアプリケーションが扱う業務の伝票や請求書などを収集してきて、そこにあるデータに基づいてデータを列挙することもできます。このように、現実の業務やアプリケーションの仕様などに基づいてデータ設計を行うことを「ボトムアップアプローチ」と言います。

もちろん、請求書そのままを記録しようとするとうまく表の形にできないことがありますし、表の形になったとしても、もっと分解した方が良い場合もあります (正規化の問題…これについては後でまた説明します)。

しかし、ボトムアップアプローチの最大の問題は、それが「現実にあるもののつぎはぎ」であり、次のような弱点を抱えていることです。

- 複数のサブシステム間できちんと整合性が取れているかどうか疑問である。もともとデータベースは「あらゆるデータを 1 個所に置いて共有する」ものなので、その中に個別アプリむけにバラバラに表を用意していたのでは共有の利点がなく無意味。
- さらに問題なのは、データベースは「将来の業務の進化や、将来になってから作られるアプリケーションまでをサポートした」ものだということ。これら「まだない」ものはボトムアップではどこからも情報が来ないため、対応できない。

そこで、これと対照をなすアプローチとして「トップダウンアプローチ」が存在しています。トップダウンアプローチでは、まず対象とする業務の動き方のモデルを考え、そのモデルが動くために必要なデータは何であるかを分析し列挙していくことを通じて、データ設計を行うものです。たとえば、前回取り上げた DFD による構造化分析を行えば、どのようなデータが流れているかは捕捉できるわけですから、これをもとにデータモデルを作ることができますね。

トップダウンアプローチでは、「あるべき理想的な業務のあり方」を考えてそれに基づくデータ設計が行えますから、各種の業務がうまくその中にあてはまる、統一的なビジョンの設計を提供可能です。ただし、それには業務の動き方のモデルがうまく現実 (ないしその改善されたもの) と整合している必要があるわけですが。

そして実際には、現存する伝票などのデータも無視することはできませんし、想像だけできちんと細かい詳細を詰めることも難しいですから、トップダウンとボトムアップの双方を併用したシステム設計を行うのが通常の姿だと言えるでしょう (ボトムアップオンリーはあるかも知れませんが、上記の問題点のためあまり望ましくないと言えます)。

1.4 データモデルの図法 IDEF1X

データモデリングのためには、前回業務のプロセスを分析するのに DFD を使ったのと同じように、図法を決めてその図法でデータやその関係を記述します (普通は)。ここでは図法として IDEF ファミリーの図法である **IDEF1X** を用います。

データモデルは一般に、エンティティ (さまざまな対象データ) と、それらに間のリレーションシップ (関係) とを記述するので、その図法は「ER 図」(Entity-Relasinship Diagram) と呼ばれることもあります。なお、ER 図にもさまざまな流儀のものがあるので (UML のクラス図を ER 図のように使うこともできます)、その 1 つが IDEF1X ということです。

IDEF1X では、エンティティには次の 2 種類の形があります。

- 独立エンティティ — 長方形で表し、それ自体単独で存在し得るようなデータに対応。
- 依存エンティティ — 角の丸い長方形で表し、いずれかの独立エンティティと関連づけられてのみ存在するようなデータに対応。

たとえば、学校 (小中高校ないし大学) の生徒/学生とクラスの関係を考えます。高校まででは、生徒にはクラスごとの「出席番号」だけがあることが多く、その場合はまずクラスを決めて、それのクラス内での出席番号を指定することではじめて、生徒が特定できます。つまり生徒はクラスなしには存在できない依存エントリです。

一方、大学では学生は学籍番号を持っていて、クラス (語学のクラスなど) に学生が所属しているという関係自体はありますが、クラスに入っていなくても特に困りません。つまり、学生は独立エンティティになります。この両者を図示したものを図 2 に示します。

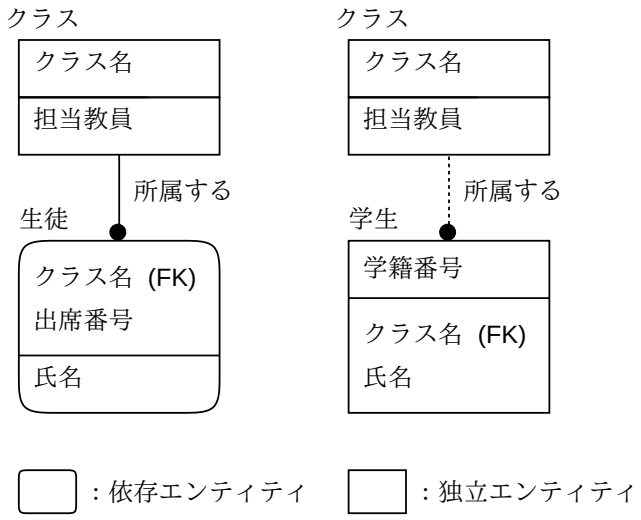

図 2: 依存エンティティと独立エンティティ

ここでもう少し記法の説明をしましょう。エンティティの上側に名前、箱の中にその要素名の並びを書きますが、横線が引いてある場合は線の上側にある要素 (列) がそのエンティティを一意に識別します。つまり、RDB 的に言えば主キーとなります。たとえばクラス名はクラスを識別する主キーです。また、生徒はクラス名と出席番号を連結したもので一意に識別できるので、これらが主キーです。

主キー以外の要素の性質はかっこ書きで記述しますが、次のものがあります。

- FK (Foreing Key) — 他のデータの主キーを格納している。
- AK (Alternate Key) — 代替キー、つまり主キーとはしていないが、このデータを用いてエンティティを一意に識別することもある。
- IE (Inversion Entry) — エンティティを一意に識別するという性質はないが、このデータを用いてエンティティを検索することがある。

エンティティ間の関係 (リレーションシップ) は線で示していますが、その線が「必ずある」ときは実線、「ない場合もある」ときは点線で示します。図 2 の場合は、生徒は必ずクラスに所属しているが、学生はクラスに所属していない場合もあるわけです (依存エンティティは必ず親に相当するエンティティと実線で結ばれます)。線についている黒丸は、その黒丸がある側のエンティティが複数対応することを表します。生徒も学生もクラスに複数名が所属するのでこうなるわけです。そして、「どういう関係か」を示す動詞句を書いておくとその関係の意味が分かりやすくなります。

黒丸の横に記号や数を書いて「何個対応している」という情報をさらに表すことができます。具体的には次のものがあります。

- ● — 0 個以上
- ● P — 1 個以上
- ● Z — 0 個または 1 個
- ●数 — その数以上
- ◇ — 0 個または 1 個 (点線にのみ使う)

また、IDEF1X ではあるデータを複数通りのどれかに分類する場合には図 3 のような書き方を使います。ここで 3 つの記法の意味はそれぞれ次の通りです。

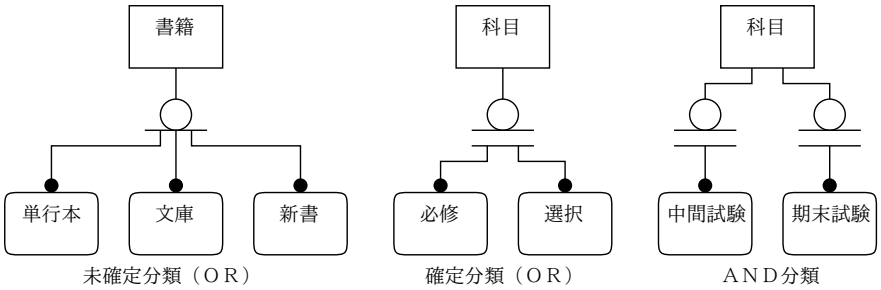

図 3: IDEF1X の分類の記法

- 未確定分類 — 親要素が複数の場合 (下に描かれた依存エントリ) のどれかに分類 (=対応づけ) されるが、ここに描かれたものが全部ではなく、これ以外の場合もあり得る。
- 確定分類 — 上と同様だが、ただしここに描かれたものが全て。つまり、親要素は必ず複数の場合のどれかちょうど 1 つと対応する。
- AND 分類 — 親要素が下に描かれた依存エントリのそれぞれと対応することもしないこともある。つまり、2 つ以上と対応することもある。

ではもっと実際のデータベースに近い例として、(「計算機科学」で例題に使った) 簡単な部品販売データベース図 4 の 4 つのリレーションを記述してみましょう (図 5)。

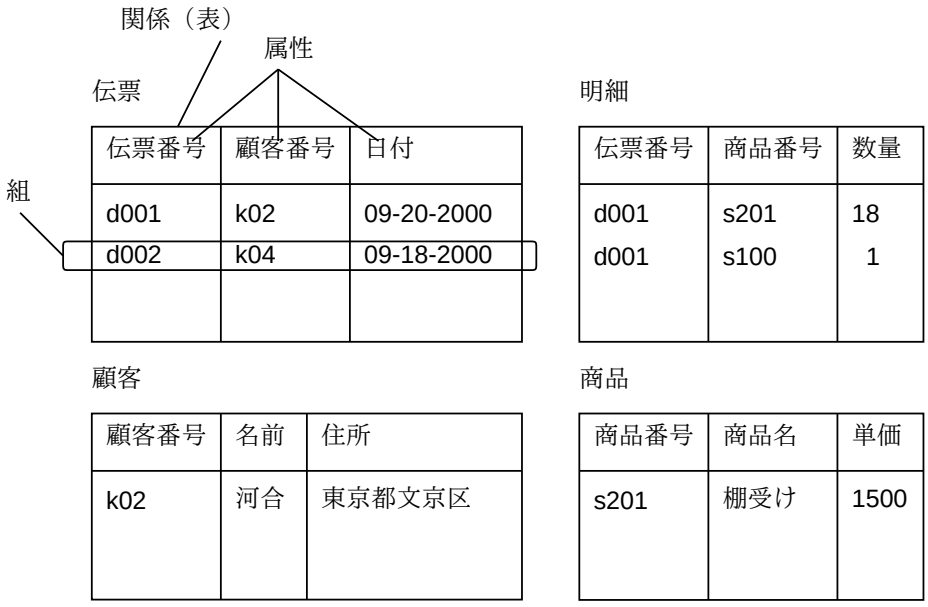

図 4: 部品販売データベースの例

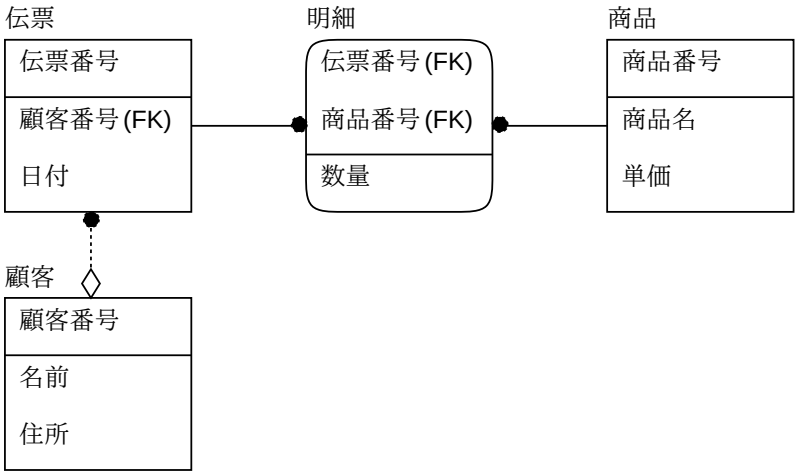

図 5: 部品販売データベースのデータモデル図

ここにはデータモデルでよく使われるパターンが 2 つ現れています。1 つは、「明細」は伝票と商品の両方に依存したエンティティで、1 つの伝票に複数の商品を対応づけ、さらに余分のデータ (数量) を付属させています。もう 1 つは、「顧客」が伝票が必要とする顧客のデータを保持していますが、伝票自体は独立エンティティで、顧客から見て対応する伝票が 1 つもないこともあるという形になっている点です。

1.5 例題: 共有日記サービス

ではここで、今回分析の例題として使う一種のソーシャルサイトのようなサービスである「共有日記サービス」について説明しましょう。このサービスの概要は次のようなものです。

> 1 人では日記をつけても続かないし面白くないので、仲間うちで共同で日記を書けるようにする。日記は日毎に分かれていて、登録者はどの日でも最大 140 文字のコメントを書き込むことができる。各書き込みはそれぞれの日の中では書き込み順に並んでおり、冒頭に誰が書き込んだかの表示がつく。

以上が基本部分ですが、さまざまな拡張が考えられます (下記演習参照)。

演習 1 この共有日記サービスの基本部分のデータモデルを作成しなさい。完成したら次のような拡張を行う場合のデータモデルも作ってみなさい。各エンティティにどれくらい詳しい属性を持たせるかは自由に決めてよい。

a. 日記とは別に、それぞれの日がどんな日であるかを (個人とは無関係に) 記入できるようにする。
b. それぞれの人は「友達」を指定でき、各書き込みは「友達」にだけ見られるようにする。
c. それぞれに人は複数の「場」に所属でき、それぞれの日の書き込みも通常のもの (無名の場) に加えて場ごとの書き込みができるようにする (もちろん、所属している場しか読み書きできない)。
d. そのほか、面白そうなソーシャル機能を考案してみる。

2 Javaによるデータベース2層/3層アプリケーション

2.1 プログラミング言語によるデータベースアクセス

実際にデータベースを使用するソフトウェアを作るには、プログラムから DBMS の機能を呼び出すことが必要です (当り前)。そのためには、次のような選択肢があります。

- 専用言語 — データベース機能を最初から持つ専用言語を使用する。4GL(第 4 世代言語) と呼ばれるものの中にはこのようなものが多くある。MAPPER(Univac)、PL/SQL(Oracle) など。
- 埋め込み言語 — 普通のプログラミング言語の一部にデータベースを扱う機能を拡張として埋め込めるもの。COBOL は汎用言語だがデータ処理に昔から使われているので、COBOL の構文の中に SQL の構文が埋め込めるような処理系は古くから使われている。PostgreSQL パッケージには C 言語に SQL を埋め込む処理系が含まれている。最近では C ♯ (3.0) に LINQ と呼ばれる埋め込み言語機構が含まれる。
- ライブラリ呼び出し — 普通の言語からライブラリとしてデータベース機能を呼び出す。一番普通だが繁雑になりやすい。現在ではオブジェクト指向や関数型とかで大きな構造を扱いやすくなったので、以前よりは繁雑さが減っているという説もある。
- O/R マッピング (Object/Relation Mapping) — RDB のタプルを 1 つのオブジェクト (レコード) に対応させ、そのフィールドを読み書きすることで結果としてデータベースの読み書きができるような機構。プログラムの字面上は分かりやすくなるとされるが、検索などで SQL を書くのならある意味同じことだし、SQL でないのなら新しい言語もどきを覚えるか繁雑に 1 レコードずつ処理するかなのでいまいちという説もある。

ここではそんなに面倒なことをするつもりはないので、普通にライブラリ呼び出しを用いることとし、JDBC(Java DataBase Connectivity) 機能により Java から PostgreSQL を呼び出すことにします。

2.2 JDBC の概要

JDBC は簡単に言えば「Java プログラムから DBMS に接続して SQL を送り、その結果を受け取ることができるようなライブラリ API」だと言えます。ここでは (話を短くする t めに)PostgreSQL 接続を前提に最小限の機能を説明します。

- import 文 — JDBC API は `java.sql` パッケージに入っています。

```
import java.sql.*;
```

- JDBC ドライバによる DB サーバとの接続 — JDBC ドライバを用いて DB サーバと接続するにはクラス `DriverManager` のクラスメソッド `getConnection()` を用いて次のようにします。

  `Connection con = DriverManager.getConnectoin(`URL`, ユーザ名, パスワード);`

  ここで URL には以下に例示するような形式の PostgreSQL JDBC URL を文字列として指定します「`jdbc:postgresql://ホスト:5432/データベース名`」。5432 は PostgreSQL の標準ポートです。ユーザ名とパスワードも文字列で指定します。なお、PostgreSQL JDBC ドライバは実習用に久野のところに置いてありますので、次のようにしてクラスパスを設定して使用してください。

```
export CLASSPATH=.:/u1/kuno/work/psql.jar
```

- コネクションが張れたら次のように実行文を作成して SQL を与えて実行します。

```
Statement stmt = con.createStatement();
stmt.executeUpdate("SQL 文の文字列");                    // 結果を返さない文
ResultSet rs = stmt.executeQuery("SQL 文の文字列");    // 結果を返す文
```

  SQL の insert、delete、update など結果を返さない文は前者で実行します (影響されたレコード数または 0 が返されます)。結果を返す文 (select) は後者を使います。

- 結果オブジェクト (ここの例では `rs` としました) からは次のようにして各結果を取り出します。
  - `rs.next()` — カーソルを 1 行進める (初期状態ではカーソルは先頭行の 1 つ前にある)。次の行の有無を `boolean` で返す。
  - `rs.getInt(`i`)` — 現在行の i 番目の属性 (フィールド) を整数として返す。
  - `rs.getString(`i`)` — 現在行の i 番目の属性 (フィールド) を文字列として返す。

本来は `Statement` オブジェクトや `ResultSet` オブジェクトは使い終わったら `close()` を読んで資源を解放した方がよいのですが、今回は省略します (`^_^;;`)。

2.3 日記共有サービスのデータベース

例題として用意した日記共有サービスのデータベース設計を図 6 に示します。

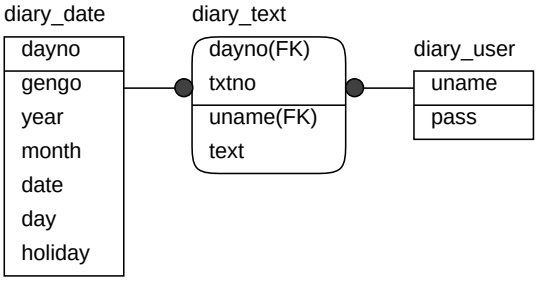

図 6: 日記共有サービス例題のデータベース設計

次のコマンドによって SQL インタフェース psql に接続できます。

```
psql -h smb -d kuno
```

以下の演習で実際にどのようなデータがあるか等を試してみてください。

演習 2 3 つのテーブルの内容を「`selet * from` テーブル`;`」で確認してみなさい。確認できたら次のような問い合わせをしてみなさい。

a. 今日の日番号 (dayno) を表示する。
b. 今日から来月の今日までの月・日・曜日・祝日の情報を表示する。
c. 10 月 10 日の書き込みをすべて表示する。
d. 10 月 10 日の最大の書き込み番号 (txtno) を表示する。
e. 自分のユーザ名とパスワードを追加する。(ヒント: `insert into diary_user values('`ユーザ名`','`パスワード`')`)

では次に簡単な JDBC の例として、上のような問い合わせを決め打ちで実行して表示するプログラムを示します。

```
import java.sql.*;

public class Sample21 {
  public static void main(String[] args) throws Exception {
    String url = "jdbc:postgresql://smb:5432/kuno";
    Connection con = DriverManager.getConnection(url, "kuno", "");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(
      "select month, date, day, user, text from diary_date" +
      " right join diary_text on diary_date.dayno = diary_text.dayno" +
      " order by month, date");
    while(rs.next()) {
      System.out.printf("%d/%d(%s) @%s: %s\n",
        rs.getInt(1), rs.getInt(2), rs.getString(3),
        rs.getString(4), rs.getString(5));
    }
  }
}
```

動かす様子も示しておきます。

```
% export CLASSPATH=.:/u1/kuno/work/psql.jar  ← 1 回だけでよい
% export LANG=ja_JP.eucJP                    ←〃
% export XMODIFIERS='@im=kinput2'            ←〃 (後で GUI から日本語入力するため)
% javac Sample21.java
% java Sample21
1/1(土) @kuno: お正月。
9/29(木) @kuno: 今日は授業の準備中。
10/1(土) @kuno: 博士発表会だ。
10/8(土) @kuno: 今年は体育の日のあたりで 3 連休になっている。私は関係ないけど。
10/10(月) @kuno: わーい体育の日!
%
```

演習 3 上の例題プログラムを直して、好きな問い合わせを表示してみなさい。OK なら、演習 2 のいくつかを決め打ちで実行するプログラムを作ってみなさい。

2.4 日記共有サービスの 2 層アプリケーション

2 層 (2 tier) アプリケーションとは、ユーザが使用するインタフェース (今日では普通は GUI ですね) とビジネスロジック、および背後にある DBMS の 2 つの部分に分かれた構造を持つアプリケーションです。DBMS はアプリケーションとは分かれているのが当り前ですから、2 層アプリケーションは「最小限の分割による」ものだと言えます。

ここでは先に示したデータベース用に簡単な GUI を持ったクライアントを作成してみます (図 7)。基本的な機能は次の通り。

- 起動時にユーザ名とパスワードを指定して起動。
- 窓の上半分にユーザの書き込みが見える。
- 日付を設定して View ボタン押すとその日以降 50 日ぶんの書き込みの表示になる。

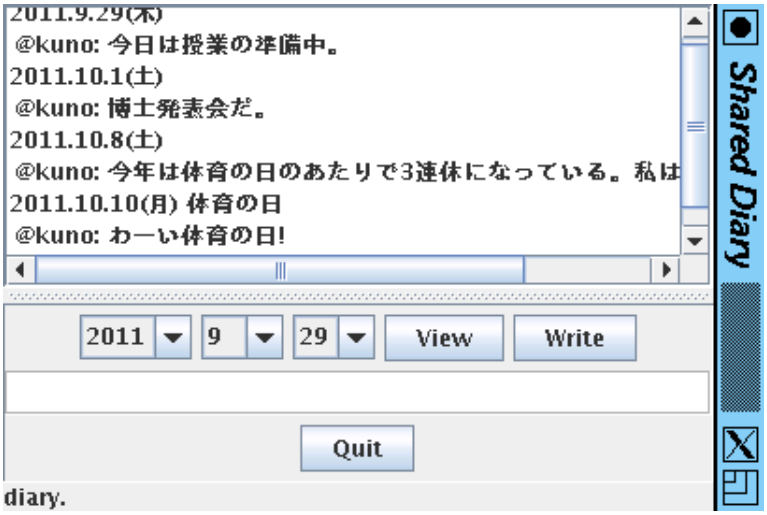

図 7: 共有日記クライアント

- 日付を設定して書き込みを記入し Write ボタン押すとその書き込みがその日の書き込み群の末尾に追加される。

コードは全部説明を書いていると大変なのでその場で説明します。

```
import java.io.*;
import java.sql.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;

public class Sample21 extends JFrame {
  Connection con = null;
  String uname = null;
  int startday = 244, endday = 304;
  DefaultListModel cont = new DefaultListModel();
  JList list = new JList(cont);
  JScrollPane sc = new JScrollPane(list);
  JPanel p1 = new JPanel(), p2 = new JPanel(), p3 = new JPanel();
  JSplitPane sp = new JSplitPane(JSplitPane.VERTICAL_SPLIT, sc, p1);
  JTextArea a1 = new JTextArea();
  JComboBox c1 = new JComboBox(new String[]{"2011","2012","2013"});
  JComboBox c2 = new JComboBox(new Integer[]{1,2,3,4,5,6,7,8,9,10,11,12});
  JComboBox c3 = new JComboBox(new Integer[]{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
                              16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31});
  JButton b0 = new JButton("View");
  JButton b1 = new JButton("Write");
  JButton b9 = new JButton("Quit");
  JLabel l0 = new JLabel("diary.");
```

```
public Sample21(Connection c, String u, int yy, int mm, int dd) {
  super("Shared Diary");
  con = c; uname = u;
  c1.setSelectedIndex(yy-2011); c2.setSelectedIndex(mm-1); c3.setSelectedIndex(dd-1);
  p1.setLayout(new BorderLayout());
  p1.add(a1); p1.add(p2, BorderLayout.NORTH); p1.add(p3, BorderLayout.SOUTH);
  a1.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED));
  add(sp); sp.setDividerLocation(250); sp.setResizeWeight(1.0);
  add(l0, BorderLayout.SOUTH);
  p2.add(c1); p2.add(c2); p2.add(c3); p2.add(b0); p2.add(b1); p3.add(b9);
  b0.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
      l0.setText("OK.");
      try {
        setDate(); updateView();
      } catch(Exception ex) {
        l0.setText("erro: " + ex);
      }
    }
  });
  b1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
      l0.setText("OK.");
      try {
        addText(c1.getSelectedIndex()+2011, c2.getSelectedIndex()+1,
                c3.getSelectedIndex()+1, a1.getText());
        a1.setText("");
      } catch(Exception ex) {
        l0.setText("erro: " + ex);
      }
    }
  });
  b9.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) { System.exit(0); }
  });
  try {
    setDate(); updateView();
  } catch(Exception ex) {
    l0.setText("error: " + ex);
  }
}
public void setDate() throws SQLException {
  int yy = c1.getSelectedIndex()+2011, mm = c2.getSelectedIndex()+1,
      dd = c3.getSelectedIndex()+1;
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery(String.format(
```

```
    "select dayno from diary_date where year=%d and month=%d and date=%d", yy, mm, dd));
  if(rs.next()) { startday = rs.getInt(1); endday = startday + 50; }
  else { throw new RuntimeException("specified day out of range."); }
}
public void addText(int y, int m, int d, String t) throws SQLException {
  if(t.equals("")) { throw new RuntimeException("text is empty."); }
  t = t.replaceAll("([\'\\\\])", "\\\\$1");
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery(String.format(
    "select dayno from diary_date where year=%d and month=%d and date=%d", y, m, d));
  if(!rs.next()) { throw new RuntimeException("day out of range"); }
  int n = rs.getInt(1), o = 1;
  rs = stmt.executeQuery(String.format(
    "select txtno from diary_text where dayno=%d order by txtno desc limit 1", n));
  if(rs.next()) { o = rs.getInt(1) + 1; }
  stmt.executeUpdate(String.format(
    "insert into diary_text values(%d,%d,'%s','%s')", n, o, uname, t));
  updateView();
}
public void updateView() throws SQLException {
  cont.removeAllElements();
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery(String.format(
    "select * from diary_date where dayno between %d and %d",
    startday, endday));
  String[] days = new String[endday-startday+1];
  boolean[] shown = new boolean[endday-startday+1];
  int i = 0;
  while(rs.next()) {
    days[i++] = String.format("%d.%d.%d(%s) %s",
      rs.getInt(3), rs.getInt(4), rs.getInt(5), rs.getString(6),
      rs.getString(7));
  }
  rs = stmt.executeQuery(String.format(
    "select * from diary_text where dayno between %d and %d order by " +
    "dayno, txtno limit 100", startday, endday));
  while(rs.next()) {
    if(rs.getInt(2) == 1) { cont.addElement(days[rs.getInt(1)-startday]); }
    cont.addElement(String.format(" @%s: %s", rs.getString(3), rs.getString(4)));
  }
}
public static void main(String[] args) {
  Connection con = null;
  int yy = 0, mm = 0, dd = 0;
  try {
    if(args.length != 2) { throw new Exception("usage Java PGM user pass"); }
    String url = "jdbc:postgresql://smb:5432/kuno";
```

```
      con = DriverManager.getConnection(url, "kuno", "");
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(String.format(
        "select pass from diary_user where uname = '%s'", args[0]));
      if(rs.next() && rs.getString(1).equals(args[1])) { /* OK */ }
      else { throw new Exception("cannot authenticate"); }
      Calendar now = Calendar.getInstance();
      yy = now.get(Calendar.YEAR); mm = now.get(Calendar.MONTH)+1;
      dd = now.get(Calendar.DATE);
    } catch(Exception ex) {
      System.err.println("database open failed: " + ex);
      System.exit(1);
    }
    JFrame app = new Sample21(con, args[0], yy, mm, dd);
    app.setPreferredSize(new Dimension(400, 400)); app.pack();
    app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); app.setVisible(true);
  }
}
```

演習 4 上の例題プログラムをそのまま動かしてみなさい。OK なら、(先に検討したものなども含め) 自分固有の機能追加を行ってみなさい。

2.5 日記共有サービスの 3 層アプリケーション

3 層 (3 tier) アプリケーションとは、2 層アプリケーションにおけるユーザインタフェースとビジネスロジックを分離したものを言います。これにより、1 つの作業に多様なインタフェースを用意できるなどの利点が得られます。また、今日では Web アプリケーションが一般になっていますが、この場合はユーザインタフェースはブラウザ、Web サーバ上のコードがビジネスロジック、という形で 3 層になっていると言えます。ただしこの一方で、Web サーバ上のコードはユーザインタフェースの一部であるという考え方もあります。この場合は Web サーバ上のコードとは分離してビジネスロジックを持つものが 3 層ということになります (図 8)。

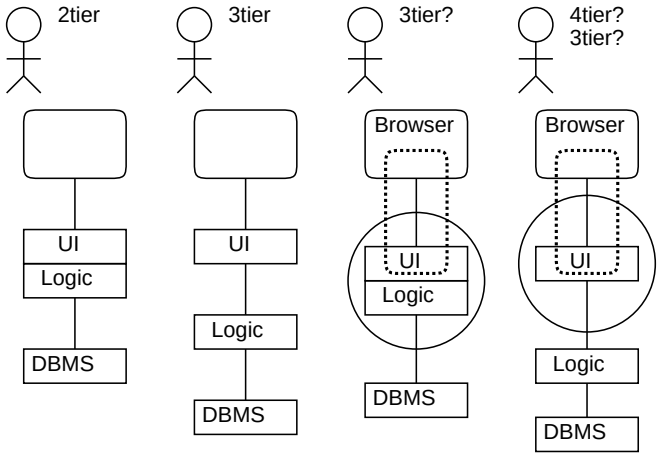

図 8: 2 層と 3 層

では、前回やったなんちゃってな Web サーバを手直しして、日記の書き込みを表示するようにしてみましょう。機能としては、指定した日から 50 日ぶんの日記をまとめて表示、とします。問題は

その「指定した日」の渡し方ですが、ここでは安易に次のようにします。

```
http://ホスト:4192/西暦/月/日
```

前回やったように、URL のこれらの情報はそのまま Web サーバに渡って来ますから、それをどのように解釈しようともサーバの勝手なわけです。

```
import java.sql.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class Sample23 {
  public static void main(String[] args) throws Exception {
    String url = "jdbc:postgresql://smb:5432/kuno";
    Connection con = DriverManager.getConnection(url, "kuno", "");
    Statement stmt = con.createStatement();
    ServerSocket ss = new ServerSocket(4192);
    System.out.println("starting...");
    while(true) {
      Socket cs = ss.accept();
      PrintWriter out = new PrintWriter(cs.getOutputStream(), true);
      try {
        Scanner sc = new Scanner(cs.getInputStream());
        boolean first = true;
        String[] a, b = {"2011","1","1"};
        while(true) {
          String line = sc.nextLine();
          if(first) {
            a = line.split(" +"); b = a[1].split("/"); first = false;
          }
          System.out.println(line);
          if(line.equals("")) { break; }
        }
        int yy = Integer.parseInt(b[1]);
        int mm = Integer.parseInt(b[2]);
        int dd = Integer.parseInt(b[3]);
        int dayno = 0;
        ResultSet rs = stmt.executeQuery(String.format(
          "select dayno from diary_date where year=%d and month=%d and" +
          " date = %d", yy, mm, dd));
        if(rs.next()) { dayno = rs.getInt(1); }
        rs = stmt.executeQuery(String.format(
          "select month, date, day, user, text from diary_date" +
          " right join diary_text on diary_date.dayno = diary_text.dayno" +
          " where diary_date.dayno between %d and %d order by month, date",
          dayno, dayno+50));
        out.print("HTTP/1.0 500 Internal Server Error\r\n");
        out.print("Content-type: text/html\r\n");
```

```
        out.print("\r\n");
        out.printf("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01//EN\">");
        out.printf("<html><head><title>calendar</title></head><body>");
        out.printf("<h1>Calender from %d.%d.%d</h1>", yy, mm, dd);
        out.printf("<table border=2>");
        while(rs.next()) {
          out.printf("<tr><th>%d/%d(%s)</th><td>@%s: %s</td></tr>\n",
            rs.getInt(1), rs.getInt(2), rs.getString(3),
            rs.getString(4), rs.getString(5));
        }
        out.printf("</table></tbody></html>");
      } catch(Exception ex) {
        System.out.println("error: " + ex);
        out.print("HTTP/1.0 200 OK\r\n");
        out.print("Content-type: text/html\r\n");
        out.print("\r\n");
        out.println("<body><p>error: " + ex + "</p></body>");
      }
      cs.close();
    }
  }
}
```

これの動いている様子を図 9 に示します。たいして何もしてませんが、表にしてセルにわけるとか幅で畳むとかは HTML とブラウザの機能で達成できますから、ある意味では先の自前 GUI よりも見やすかったりします。

図 9: サーバの出力をブラウザから表示

演習 5 上の例題プログラムをそのまま動かしてみなさい。OK なら、何らかの機能追加や改良をおこなってみなさい。