データベース 2009 #1

久野　靖*

2009.10.8

1　はじめに

この科目「データベース」は、「計算機ソフトウェア」に含まれる「データベース」の回の内容程度を学習ずみであることを前提として、次のことがらを目標として開講します。

科目の目標:

- データベースとは何であるかを体験に基づいて理解すること。
- 複数のデータモデル (特に関係モデル) とその特徴について理解すること。
- SQL を用いた RDB 操作ができること。
- 業務モデルに基づく RDB 設計ができること。
- PHP+PostgreSQL による簡単な Web アプリ作成ができること。
- DBMS のデータ操作以外の各種機能について理解すること。
- データウェアハウスとは何であるか理解すること。

各回とも、できるだけ具体的な演習を交えて、机の上の知識だけでなく、実際に手を動かして体験して頂くことを方針とします。評価については、出席および授業の中で出す課題の提出状況 (+内容) によります。ではよろしくお願いします。

2　復習:データベース入門

ここは復習ですので簡単に。データベースとは、**統合化された**、**共有可能な**、データの格納場所を言います。

普通のファイルに格納されたデータはそのデータを格納したプログラムに依存している (**データ依存**) です。その状態ではデータを直すたびにプログラムも直す必要があるし、複数のプログラムで共有するのはすごく大変です。これに対し、データベースを使うとプログラムとデータの格納形式とが分離できます (**データ独立**)。このため、データだけを単独で拡張することもできますし、複数のプログラムがデータを共有することもできます。

データが共有できるということは、1 つのデータを重複して複数個所に格納しなくてもよくなるということ。つまり**冗長性**を排除できる。なぜ冗長性がよくないかというと、更新の手間が掛かることと、データの**矛盾**が生じ得るためです。

そのほか、データベースを使うことで提供される機能としては、**問い合わせ**、**並行制御**、**排他制御**、**障害回復**、**トランザクション**、**整合性管理**、**セキュリティ**などがあります。

*経営システム科学専攻

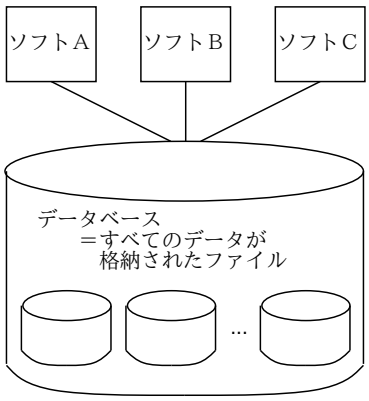

図 1: データベースの概念

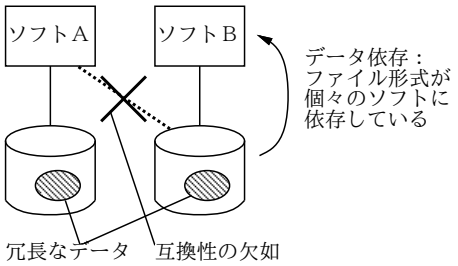

図 2: データ依存

3 DBMS とデータモデル

3.1 復習:DBMS とその役割/構造

プロセスやファイルシステムが裸の CPU やディスク上に OS というソフトウェアの働きによって実現されているのと同様、データベースはプロセスやファイルシステムの上で動く **DBMS**(データベース管理システム) と呼ばれるソフトウェア (群) によって実現されています (図 3)。先に挙げた各種の機能実現も DBMS の役割です。

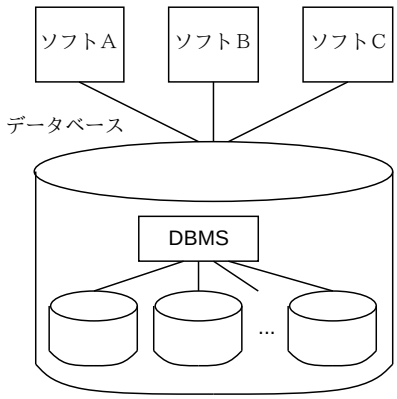

図 3: データベースと DBMS

データベースの中は、論理的には次の 3 つのレベルに分けることができます (図 4):

- **内部レベル** — ディスク上でそれぞれのデータの格納形態を定めるレベル。物理レベルとも呼ばれる。

- **概念レベル** — すべてのプログラム/ユーザが使用するデータの形態を統合した形で定めるレベル。このレベルによってデータ独立性が実現されている。
- **外部レベル** — 個々のプログラム/ユーザの必要に合わせて、概念レベルのデータをマッピングしたもの。

この分類は**データベースの 3 層モデル**として知られています。この考え方を使うと、DBMS の機能や役割りが分かりやすく整理できます。

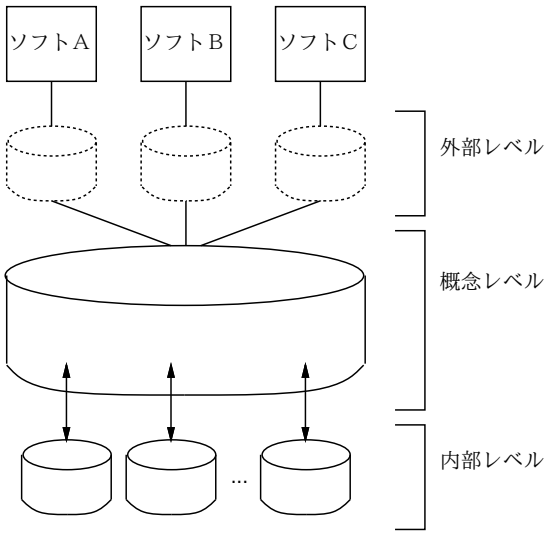

図 4: データベースの 3 層モデル

DBMS が概念レベルや外部レベルでデータをユーザに提示する枠組みのことを**データモデル**と呼びます。データベース上のデータに対して施せる (DBMS が提供する) 操作も、データモデルによっておおよそ定まってきます。また、データモデルごとにそれをうまく (効率よく) 実装するための内部レベルの設計が工夫されていきます。

3.2 ファイル編成とディスク上データ構造

もともと計算機の主記憶は限られた容量しかないため、大量のデータ保管が必要な場合はディスク (さらに古くはテープ) 上に適切なデータ構造を作ってそれを読み書きしながらデータ処理を行って来ました。このような構造を一般に**ファイル編成**などと言います。

これらの特徴は、とにかくディスク上のデータ構造なので読み書きに時間が掛かるため、できるだけ読み書きの回数を減らすことに注力されていたことです。それには、1 つの読み書き単位 (ブロック) に必要なデータをうまくまとめて、1 回読むとそれらがまとめて取れるようにする、などの工夫が必要です。このあたりが、語単位でどの番地でも (モデル的には) 同じ時間でアクセス可能な主記憶上のデータ構造と違っているところです。

以下に代表的なファイル編成を簡単に挙げておきます。

- **順編成** — ファイルをレコード単位で順番に並べたもの。順番に読んで行くことしかできないが、たとえばテープ上にデータを保管している場合はどのみちこの編成にするしかない。
- **相対編成** — データがブロック単位でディスク上に格納されていて、「何番のブロック」という指定をするとそのブロックが直接読み書きできる。ディスク上のデータのモデルになっている。たとえばハッシュ表を用いて、キーを指定してハッシュ関数でブロック番号を計算するといった使い方だとそのキーのデータが 1 回で読める、といった使い方をした (しかしハッシュ表をプログラムで実現すると色々大変)。

- **索引編成** — データを一連のキーの順番で格納することとして、どのブロックにはどの範囲のキーが格納されているかを別の場所に用意した索引で管理しておく方式。データが多くなると索引が主記憶に入り切らないため 2 段、3 段に索引を作る必要がある。また、特定位置に方よってデータが挿入されると索引構造がアンバランスにやって再編成が必要になるなどの弱点があった。
- **B 木編成** — 比較的新しく考案されたデータ構造である B 木を用いて索引編成の弱点を克服している。B 木では索引とデータが一緒に 1 つのブロックに格納されていて、挿入によりブロックが一杯になるとブロックの分割が起きる。また削除によりブロックの利用率が減少するとブロックの併合が起きる。これにより領域の利用率が一定範囲内にとどまり、索引も自動的に増減できる。

今日ではプログラマが直接このようなディスク上データ構造を扱うことはなく、DBMS の実現としてこれらが使用され、プログラマは DBMS を介してこれらを使うことになりますが、内部がどうなっているかを意識できるようになっておくとチューニングなどの時に有利だと思われます。

3.3 主要なデータモデル

DBMS がユーザに提供するデータの定式化の枠組みを**データモデル**と呼びます。データベースの歴史ではさまざまなデータモデルが現れては消えて行きましたが、ここで代表的なものを簡単に説明しましょう。

- **索引つきレコードモデル** — もっとも初期のデータベースは先に述べた索引編成のファイルのようなものであり、各レコードの主キー値の順にレコードが並んでいて、主キー値を指定するとそのレコードが取得できる (ないし、そのようなレコードが存在しないことが分かる) ものだった。しかしそれだけでは主キーでしか検索できず不自由なので、それとは別のキーを持った転置索引 (inverted index) が作成でき、こちらのキーでも同様にレコードが検索できるようにしたものが広く使われた。ADABAS などが代表的。
- **階層データモデル** — ある種のデータは、本質的に階層構造を持っている。たとえば、ある大学の授業科目であれば、学部→学科→専門/非専門→科目名、などのように。このとき、索引つきレコードモデルでは、多くのレコードに繰り返し同じ学部名が現れるといった無駄が生じる。そこで、階層ごとに別の索引を用意し、それぞれのレベルでの値を指定すると検索でき、その範囲内のレコードが順次処理できるといった、階層構造を活かしたデータモデルの DBMS が作られ普及した。IBM の IMS が代表的。
- **ネットワークデータモデル** — 階層データモデルでは、1 つのレコードは 1 つの親 (ならびに祖父等) しか持てない。しかし、データを複数の観点から分類したいことはよくある。転置索引はその 1 つの方法だが、データモデルとして一般に「あるレコードは、任意個の子レコードを持てる」という形にしたものがあり、ネットワークデータモデルと呼ばれる。CODASYL 委員会 (データ処理に関する標準を定める委員会) がそのようなデータモデルを標準化し、これに基づく DBMS が複数作られた。しかし関係データベースの普及とともに下火になった。
- **関係データモデル** — すぐ後で詳しく復習するが、データを表のような形で定式化するモデル。理論的な土台がしっかりしていて、データ操作言語で集合操作により簡潔に検索や操作が指定できる。今日のメジャーな DBMS は関係データベース (RDB) が多い。Oracle、MS SQL Server、PostgreSQL、MySQL。
- **オブジェクト指向データベース**(OODB) — RDB には、個々のデータは単純な文字列や数値などに限られるという制約がある。これに対し、オブジェクト指向言語の台頭とともに、そこで扱われる複雑な構造のデータをそのまま**永続化**して保存しておけるようにする、という方式のデータベースが作られるようになった。O2、Orion など。ただしデータ操作はプログラム言語で逐一実行する必要がある。
- **オブジェクト指向関係データベース**(ORDB) — RDB の利点である集合操作を提供するが、データ自体は継承やマルチメディアデータのように通常の RDB より OODB に近いものを提供するというあいの子。RDB の DBMS のいくつかは機能拡張により ORDB としても使える。

- **半構造データ** — ここまでのデータベースではデータはすべて設計者が定めた**スキーマ**(データのひな型) に一致する形で保存される。これは、決まった形のデータを効率よく大量に管理するという DBMS の目的からすれば当然。しかし世の中には、もっとアドホックな、形があまりはっきり決まっていないデータも多数ある。そのようなものを一般に**半構造データ**(semi-structured data) と呼び、それをどう扱うかという研究も多くなされている。

3.4 復習:関係モデルと RDB

関係データモデルは、E.F.Codd にって 1970 年に提唱されたもので、その後のデータベースに大きな影響を与えました。何が画期的だったかというと、それ以前のデータベースはデータを格納するための「方法 (実装)」を中心に考えていたのに対し、Codd はを数学的に簡潔な形式で定式化したデータモデルと、それを数学的な演算により操作することで多様な問い合わせを統一的に記述する方法とをともに提案したという点です。これにより、「プログラムでデータを 1 レコードずつ見る」という手間が掛かり分かりにくい方法を取らなくても、集合演算として大量のデータを一括して検索することが可能になったわけです。今日でもこの性質は RDB の最大の利点であり、それを越えるものはまだ見つかっていません。

関係モデルでは、データベース中のあらゆるデータを「表」のようなものとして表します (ちなみに、「関係」というのは、表のようなデータを数学的に定式化する際に使われる言葉です)。関係モデルのデータは次の 3 つの概念から組み立てられます:

- **関係**(relation) — 1 つの「表」のこと。
- **属性**(attribute) — 表の各欄のこと。
- **組**(tuple) — 表中の 1 つの (値が横にならんだ) 列のこと。

モデル的には組は属性の積 (属性の値を連結したもの) であり、関係は組の集合ということになっています。集合なので、順序は関係なく、同じ内容の組が複数存在することもありません。しかし、実際の DBMS では必ずしもそうはなっていません。

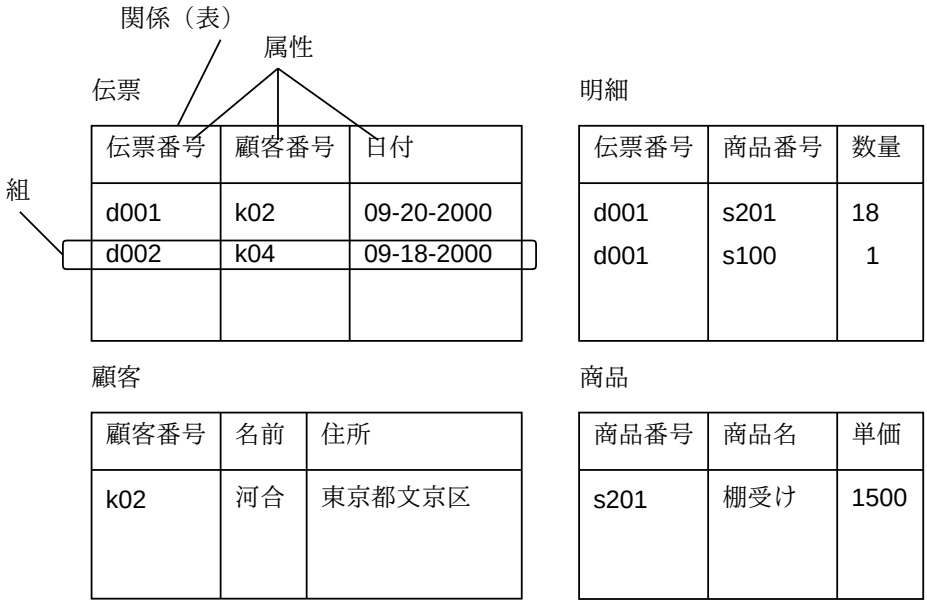

図 5: 関係データベースの例

関係データモデルの利点は、理論的土台があって関係どうしの「演算」が簡潔に定義でき、それを用いて多様なデータ処理/検索が自然に行なえるということです。たとえば、ネットワークデータモデルまでで

はデータの処理はプログラムが「1 レコードずつ読み進めながら」行っていましたが、関係モデルでは「集合」が値なので、1 つの演算で何万レコードでも一挙に処理できるわけです。

他方、RDB の弱点には、理論的には関係演算により簡潔に記述できる処理でも、実際に実行させようとした場合に多量のデータ操作に対応するため、他のモデルに比べて性能が低い、というものがありました。しかし実装技術とハードウェア性能の向上のおかげで、この弱点は現在ではほぼ克服ずみです。RDB のもう一つの弱点は、データとして文字列、数値など基本的なデータ型しか扱わない点です。これについてはオブジェクト指向データベースによって扱われており、また RDB の DBMS にそのような機能を追加したものもあります (ORDB)。とりあえず、今日の一般的なデータ処理では RDB は十分有効に使われている、といえるでしょう。

というわけで、以下では RDB を中心にデータベースの各種機能とその利用について見て行くことにします。

4 SQL

4.1 SQL の由来

IBM 社は (Codd も IBM 研究所の人でした) は RDB に関する研究をするなかで、関係データベースの検索を行う計算機言語の開発も進め、1974 年に SEQUEL(Structured English Query Language) を発表しました。その後何回かの改定を経て名前は SQL(Structured Query Language) に変更され、IBM の RDB プロトタイプ DBMS である System R に搭載されました。

このプロトタイプが好評だったことから、IBM は商用製品の DBMS である DB2 を開発してそこに SQL を搭載し、また他の DBMS ベンダーもそれぞれのバージョンの SQL を開発して搭載するようになりました。現在では、SQL にできることは検索だけではなく、挿入や削除などを含むデータ操作一般であることから、SQL は検索言語ではなく**データベース操作言語**と呼ばれるようになっています。

初期の時点から SQL には各ベンダーごとの「方言」つまり違いがありましたが、それでは不便なため 1982 から ANSI(米国規格協会) が SQL の標準制定に取り組み、最初の標準規格である SQL/86 が 1987 年に制定されました。これは ISO(国際標準化機構) の標準にもなっています。その後、ISO/ANSI は SQL の機能拡張に取り組み、SQL89、SQL92、SQL99 と改訂されてきています。

以下では SQL99 を土台に解説しますが、ただし標準化といいながら実際の DBMS における SQL 処理系では SQL99 と細かく違うところが多数あります。このため、実習に使う PostgreSQL の SQL99 との共通部分を説明し、両者で違いがある場合はその旨注記していきます。

4.2 データ型と値

データ型 (data types) とは「データの種類/種別」を表す言葉です。RDB では数値、文字など比較的限定された形のデータ型のみが利用可能です (それ以上の構造はリレーションで表現)。SQL99 では多数のデータ型を定めていますが、代表的なものだけ挙げておきます。

- `integer` — 整数型 (`int`)
- `numeric` — 真数値型
- `real` — 実数型
- `float(`p`,`s`)` — 精度と位取りを指定した実数
- `boolean` — 論理値
- `bit` — ビット列
- `bit varying` — 可変長ビット列
- `character(`n`)` — 文字列 (`char`)

- character varying(n) — 可変長文字列 (varchar)
- date — 日付
- time [with time zone] — 時刻
- timestamp [with time zone] — タイムスタンプ (整数の時刻値)
- interval — 時間間隔

リテラル(定数) の書き方は、数値については C などのプログラミング言語と同様です。文字列は「'…'」のようにシングルクォートで囲みます。文字列の中に「'」を入れたければ「'DON''T」のように 2 つ連続して書くと 1 つぶんとして扱われます。

いずれの型の属性についても、「NOT NULL」という制約が指定されていなければ、値として NULL が格納可能です。NULL は「0」「"」などのデータ値とは別個のものであり、「値がない」ことを表します。数を数えたり最大/最小/平均などを求めるときには NULL は除外されて処理されます。

4.3 SQL の主要な文

4.3.1 CREATE DATABASE

1 つのデータベースの作成。SQL99 では SQL の文として用意されていますが、PostgreSQL では Unix のコマンドを用いてデータベースを作成します。

```
createdb データベース名
```

データベース名を省略すると自分のユーザ名と同名のデータベースが作成されます。

4.3.2 DROP DATABASE

データーベスの削除。SQL99 では SQL の文として用意されていますが、PostgreSQL では Unix のコマンドを用いてデータベースを削除します。

```
dropdb データベース名
```

データベース名を省略すると自分のユーザ名と同名のデータベースが削除されます。

4.3.3 GRANT

```
GRANT 権限指定 ON 表名 TO 対象者
```

指定した表に対するアクセス権を対象者に付与します。対象者は次の形のどれかで指定します。

- PUBLIC — 誰にでも
- ユーザ名,... — 指定したユーザに

権限指定は次のものをカンマで区切って複数並べられます。

- SELECT、INSERT、DLETE — 組の検索/挿入/削除
- UPDATE [(属性名,...)] — 組 (ないしその属性の一部) の書き換え
- REFERENCES [(属性名,...)] — この表 (の指定した属性) を外部キーとする表の作成権限

または「ALL [PRIVILEGES]」により「すべての権限」を指定することもできます。

4.3.4 REVOKE

```
REVOKE 権限指定 ON 表名 FROM 対象者
```

GRANT で付与した権限を引き上げます。権限や対象の指定は GRANT と同様。

4.3.5 START TRANSACTION

SQL99 ではトランザクションは「START TRANSACTION」により明示的に開始することもできますが、データ操作を行うコマンドによっても自動的に開始されます。そして、トランザクションの終了は COMMIT か ROLLBACK によります。これに対し PostgreSQL では、その設計上、データ操作を行うコマンドがすべて単独のトランザクションとして実行されますが、明示的に

```
BEGIN
```

によりトランザクションを開始することもでき、その場合は COMMIT か ROLLBACK で終了させる必要があります。

4.3.6 COMMIT

```
COMMIT
```

トランザクションを成功終了させます。トランザクションの内部での変更は COMMIT によりはじめて確定されます。

4.3.7 ROLLBACK

```
ROLLBACK
```

トランザクションを中止させます。トランザクションの内部での変更は ROLLBACK を実行した場合すべて「なかったこと」になり、データの状態は開始時点に戻ります。

4.3.8 CONNECT

DBMS との接続。SQL99 では SQL の文として用意されていますが、PostgreSQL では Unix のコマンドを用いて DBMS と接続します (当り前か)。

```
psql データベース名
```

psql で DBMS に接続すると、SQL の文を入力して実行させられます。

```
SQL 文 ; ←「;」が出て来るまで何行でも入力できる
```

SQL 文とは別に、次のような psql コマンドも利用可能です (こちらは必ず「\」で始まり、「;」は不要)。

- \copy　表 (from | to) ファイル [using delimiters ’文字’] [with null as ’文字列’] — 手元のファイルとデータベースの間でデータをコピー
- \d　表 — 指定した表の属性一覧を表示
- \e — コマンドバッファの内容をエディタで編集
- \g — コマンドバッファの内容を実行
- \g　ファイル — コマンドバッファの内容を実行し、その結果を指定したファイルに書き込む

- \h — ヘルプメッセージの表示
- \i　ファイル — ファイルの内容を実行
- \p — コマンドバッファの内容を表示
- \q — psql を終了
- \z — データベース中の表とその許可を一覧表示

4.3.9 CREATE TABLE

CREATE TABLE 表名 (属性定義, [属性定義 | 表制約定義] …)

表を定義します。属性定義は次の形になります。

属性名　データ型　[制約,...]

制約はその属性に対する制約を指定する場合に記述し、主なものとして次のものがあります。

- NULL、NOT NULL、UNIQUE、PRIMARY KEY — NULL を許すか否か、一意的な値か、主キー (これも一意的な値である必要がある) かを指定
- DEFAULT 標準値 — 暗黙値の指定
- CHECK (検査条件) — 検査条件を指定
- REFERENCES 表名 (属性名) — 他の表の属性を参照することを指定

上記は「属性につける」制約でしたが、表につける制約としても NULL/NOT NULL 以外は同じものが書けます。

- PRIMARY KEY (属性名,...) — 上と同様
- UNIQUE (属性名,...) — 上と同様
- CHECK (検査条件) — 上と同様
- FOREIGN KEY (属性名,...) REFERENCES 表名 (属性名,...) — 他の表の主キーを参照することを指定

4.3.10 CREATE VIEW

CREATE VIEW 表名 [(属性名,...)] AS SELECT 文

ビューを定義します。ビューの内容は SELECT 文による検索結果ですが、必要なら属性名をつけ替えることができます。**なお、PostgreSQL ではビューは読み取り専用であり、ビューに対する挿入/削除/更新ができないという重大な制約があります。**

4.3.11 ALTER TABLE

ALTER TABLE 表名 変更指定

作成ずみの表のスキーマを変更する機能を提供します。変更する対象に応じて変更指定として次のような書き方を使います。

- ADD [COLUMN] 属性名 型 [制約,...] — 属性を追加。
- ALTER [COLUMN] 属性名 SET DEFAULT 値 — 標準値を指定。

- ALTER [COLUMN] 属性名 DROP DEFAULT 値 — 標準値を削除。
- RENAME 属性名 TO 新属性名 — 属性名を変更。
- RENAME TO 新表名 — 表名を変更。
- OWNER TO ユーザ — 所有者を変更。
- ADD 制約 — 制約を追加。
- DROP CONSTRAINT 制約 — 制約を削除。

4.3.12 SELECT /単一表検索

SELECT は SQL で最も中心となる文であり、非常に複雑な機能が盛り込まれているので、複数の場合に分けて説明します。まずは 1 つの表のみに関する検索から。

```
SELECT [ ALL | DISTINCT ] 選択リスト FROM 表名 [ AS 別名 ]
  [ WHERE 条件式 ]
  [ GROUP BY 属性名,...  [ HAVING 条件式 ] ]
  [ ORDER BY 順序指定 ]
```

ALL/DISTINCT は、組に重複があってもそのままにするか、重複を除去するかの指定で、何も指定しないと ALL として解釈されます。表名は検索対象の表ですが、文の中で短く指定するため別名をつけることもできます。

選択リストは、「何と何を」出力 (これも表) とするかを指定するもので、「*」(元の表の属性をそのまま全部) という指定、または次のものの「,」で区切った並びを指定ができます。それぞれについて「AS 名前」をつけることで、名前のつけ替えが可能です。

- 属性名 — 元の表の属性そのまま
- 計算式 — 属性名、定数を+-*/() で組み合わせた式。
- 集計関数 — 以下に説明

集計関数は、検索により取り出したデータ (なしいグループ) 内で個数カウント (COUNT)、合計 (SUM)、平均 (AVG)、最大 (MAX)、最小 (MIN) のいずれかを計算するもので、以下のいずれかの形が使えます。

- COUNT(*) — 単純に全個数を数える。
- 集計関数 ([ALL|DISTINCT] 属性名) — 属性の値 (重複そのまま/重複除去) について関数計算。
- 集計関数 (計算式) — 計算式の値に対して関数計算。

WHERE 句は検索条件を指定する部分で、条件式とは以下のような条件を and、or、not、() で結合したものです。

- 式 比較演算 式 — 比較の結果
- 式 [NOT] BETWEEN 式 AND 式 — 範囲内か否か
- 式 IN (値, ...) — 集合に含まれるか否か
- 式 LIKE ’文字列’ [ESCAPE ’文字’] — 文字列照合
- 式 IS [NOT] NULL — NULL か否か

比較演算は=、<>、 >、 >=、 <、 <=のいずれかが使えます。

GROUP BY 句は指定した列の値が等しいものごとに組をグループ化した上でそれぞれのグループについての計算を指定します。加えて HAVING 句を指定することで「できたグループのうちどのようなものだけ」という限定をつけられます。

GROUP BY 属性名, ... [HAVING 条件式]

GROUP BY を使うと、FROM には GROUP BY に指定した属性、そのような属性を集計関数で集計したもの、または定数のいずれかしか指定できなくなります (意味を考えれば当然)。

ORDER BY 句は、結果を特定の順に並べたい場合に指定します。

ORDER BY (属性名 | 番号) [ASC | DESC] ,...

基本的に、どの属性の昇順/降順で並べるかを指定しますが、属性名のかわりに、出力の何番目の列という形で指定することもできます。

4.3.13 SELECT /副問い合わせ

副問い合わせとは、ある SELECT の WHERE 句の中に別の SELECT(内側検索) が含まれているようなものを言います。まず、内側検索では使える属性は 1 つです。そして、結果集合の大きさが 1、つまり単一の値のみを返す場合はそれを普通の式の値として使うことができます。

- (内側検索) — 任意の式として使える。

また、1 つの値ではなく複数の値 (0 個や 1 個でもよい) を返す場合、つまり一般の集合の場合には、次のいずれかが使えます。

- 式 比較演算 ALL 内側検索 — 内側の SELECT で検索した結果のすべてが左辺の式との比較演算を満たす。
- 式 比較演算 ANY 内側検索 — 内側の SELECT で検索した結果のどれか 1 つ以上が左辺の式との比較演算を満たす。ANY の代わりに SOME と書いても同じ。
- 式 [NOT] IN 内側検索 — 内側の SELECT で検索した結果の集合に式が含まれている/いない。
- [NOT] EXISTS 内側検索 — 内側の SELECT の結果が空集合でない/ある。

要するに、条件の一部として内側の SELECT を使った検索結果を利用できるわけです。

4.3.14 SELECT /集合演算

複数の SELECT の検索結果を集合と考え (もともと関係は集合ですが)、その和/積/差を求めるのが集合演算です。

```
SELECT … FROM 表名 1 [ AS 別名 1 ] [ WHERE … ]
  ( UNION | INTERSECT | EXCEPT ) [ ALL ]
  SELECT … FROM 表名 2 [ AS 別名 2 ] [ WHERE … ]
  [ ORDER BY 順序指定 ]   [ LIMIT ( 個数 | ALL ) [ OFFSET 開始点 ] ]
```

UNION、INTERSECTION、EXCEPT が和、積、差に対応します。ALL を指定すると重複する組がそのまま残りますが、指定しないと重複する組は 1 つだけが出力されます。ORDER BY は通常の順序指定、LIMIT は最大何個取り出すかを指定します (普通は ORDER BY と一緒に指定して上位 N 個を取り出します。また OFFSET を指定して N 番目〜M 番目を取り出すといったこともできます)。

4.3.15 SELECT /結合 (θ形式)

SELECT で結合 (join) を指定するには、単に複数の表を指定して WHERE で条件指定を行う θ 形式 (これしか使えない DBMS がまだある) と、JOIN 句で明示的に指定する新形式とがあります。θ 形式の方が予約語が少ないので、こちらから説明しましょう。

```
SELECT … FROM 表名1, 表名2 WHERE …
```

のように FROM で表を 2 つ指定すると、その 2 つの表の組のあらゆる組み合わせ (組の数が M 個と N 個なら、$M \times N$ 個) の組を生成し、その上で通常の WHERE 等による絞り込みを行うことになります。ここで条件として「表名 1.x = 表名 2.y」などのように両者の間である属性どうしが等しいことを指定すると、自然結合 (natural join) になるわけですが、それ以外のものを指定しても、また絞らずにあらゆる組み合わせを生成する直積結合 (cross join) として使ってもいいわけです。

ただし、この方法だと外結合 (後述) は指定できないので、DBMS によって「*=」とか「+=」とか独自の比較演算子を用いて外結合を指定するような拡張がなされていました。現在では次に説明する JOIN 句があるのでこのような独自拡張は不要になっています。

4.3.16 SELECT /結合 (JOIN 句)

現在の SQL では、θ 形式も使えますが、普通は JOIN 句を使って結合のしかたを明記するようになっています。この方が書き方を覚える手間は必要ですが、覚えてしまえば読みやすく、また外結合 (outer join) も指定できるという利点があります。

```
SELECT … FROM 表名1 [ AS 別名1 ]
  [ NATURAL ] 結合種別 表名2 [ AS 別名2 ]
  [ ON 条件式 | USING 属性名,... ]
  [ WHERE … ] [ ORDRER BY … ] [ GROUP BY … ]
```

ここで結合種別としては次のものが指定できます。

- [INNER] JOIN — 普通の結合、つまり条件を満たした組だけが取り出される。
- CORSS JOIN — 直積結合、つまりあらゆる組み合わせ。
- LEFT [OUTER] JOIN — 条件にあてはまる組、プラスそれ以外の左側の表の組を返す (欠けている属性値には NULL が入る)。
- RIGHT [OUTER] JOIN — 条件にあてはまる組、プラスそれ以外の右側の表の組を返す (欠けている属性値には NULL が入る)。
- FULL [OUTER] JOIN — 条件にあてはまる組、プラスそれ以外の左側および右側の表の組すべてを返す (欠けている属性値には NULL が入る)。

結合条件は次の 3 つのいずれかで指定できます (どれか 1 つだけしか指定できません)。

- ON 条件式 — 一般に任意の条件を条件式として指定。
- USING(属性名,...) — 指定した属性どうしが等しいという条件指定。「ON 表1.属性名1 = 表2.属性名1 and 表1.属性名2 = 表2.属性名2 and …」と同等。
- NATURAL — 2 つの表で共通する属性名をすべて USING で指定したのと同等。

4.3.17 INSERT

```
INSERT INTO 表名 [ ( 属性名,... ) ] ( VALUES ( 値,... ) | 検索指定 )
```

表に組を追加します。属性名を指定した場合は、そこにない属性の値は NULL になります。挿入するデータは VALUES で直接指定することもできますが、他の表を検索した結果を挿入することもできます。

4.3.18 DELETE

`DELETE FROM` 表名 [`WHERE` 条件式]

検索条件にあてはまる組、または条件を省略した場合すべての組を表から削除します。

4.3.19 UPDATE

`UPDATE` 表名 `SET` 更新指定, ... [`WHERE` 条件式]

検索条件にあてはまる組、または条件を省略した場合すべての組のデータを更新します。更新指定は次の形になります。

- 属性名 `=` (式 | `NULL`) — 指定した属性の値を更新

4.4 SQL のまとめ

SQL の特徴としては、大きな表であっても 1 つの値として集合操作が行えることから、手続き言語のループなどを使わなくてもデータが扱え、多様な検索が行えるという点があります。この特徴のために、簡潔な記述で高度な操作が可能になっているわけです。その文、命令が複雑に思えるかも知れませんが、ここまでに挙げた基本部分程度を頭に入れておけば、あとはこれらの組合せで必要なことは大体行えるはずです。

SQL は RDB を操作する上での標準言語であり、DBMS による細かい違いはあるにしても、基本的な SQL を知っておけばそこにプラスして固有の部分を学べば済むので、その意味でも知っておく価値があります。

また、SQL コマンドとして用いて直接データベースをさまざまに操作して見られるので、はじめて接するデータベースの内容を調べたり、非定型業務としてデータベースから情報を取り出すなどの目的にも有効に活用できます。

定型業務を開発する場合は、通常のプログラミング言語の中に SQL の文を埋め込んで RDB の読み書きを行うことが普通なので、この点でも SQL を知っておけば、あとは開発する言語から SQL へのインタフェース方法を学ぶだけで開発が行えます。

A SQL の演習

A.1 はじめに

SQL の構文規則やそれぞれの構文の意味について授業で説明を受けたからといって、それだけで SQL 書けるようになるわけではありません。やはり実際に演習をして体験してみないと、納得が行かないし書けるようにもならないと思います。以下にいくつか SQL を用いた演習の課題を挙げますので、順次やってみてください。

その前に、「計算機ソフトウェア」でやった「動かし方」を忘れているかも知れないので一応 PostgreSQL の使い方を説明しておきます。

- PostgreSQL はクライアントサーバ方式だが、サーバ (DBMS) への接続はローカルマシン内のみからに設定してある。このため、`smb` の窓を開くか、または「`rlogin smb`」でこのマシンに接続して以下を行うこと。
- 日本語文字コードは「日本語 EUC」になっている。このため、X の端末窓 (`kterm`) の「Control+中ボタンメニュー」を出し、「Japanese EUC Mode」がチェックされていることを確認。チェックされていなければこの項目を選んでチェックをつける。
- なお、日本語入力 ON/OFF は Emacs とは違い「Shift+Space」で行う。

- 最初に自分が使うデータベースを作成するコマンドは「createdb」。特に指定しなければ自分のユーザ名と同じ名前のデータベースが用意される。これは 1 回だけやればよい。既に作ってある人は「既にあります」と言われるだけで何も起こらない。
- SQL 解釈プログラムの名前は「psql」。これを実行開始するとその中で SQL が打ち込める。終了は \q。
- あとは CONNECT のところの説明を参照。psql 固有の機能のなかでも、\z、\d、\e \i、\copy はよく使うので十分マスターすること。

A.2 テーブルの作成と変更

- 日、月、…、土をそれぞれ 0〜6 までの整数に対応させたい。そのような表を定義せよ。(CREATE TABLE)
- \d で表のスキーマを確認 (以下適宜行う)。
- 表名は「曜日表」、属性は「曜日番号」「曜日名」とする。型は適切と思われるものを自分で選択。
- 実際に 7 つの曜日のデータを挿入してみよ。(INSERT)
- select *でデータを確認してみよ (以下適宜行う)。
- 1 つどれかのデータを削除してみよ。(DELETE)
- 「曜日番号は 0〜6 の範囲である」という制約を追加してみよ。(ALTER TABLE xx ADD CHECK)
- 制約に反するデータを追加しようとしてエラーになることを確認せよ。
- 「曜日番号は主キーである」という制約を追加してみよ。(ALTER TABLE xx ADD PRIMARY KEY)
- どれかと同じ曜日番号の行を追加しようとしてエラーを確認せよ。
- 英語の名前も入れるため、属性「英曜日名」を追加してみよ。(ALTER TABLE xx ADD COLUMN)
- 英語の名前も設定してみよ。(UPDATE) [1]
- 曜日名、英曜日名とも、一意 (重複があってはいけない) という制約を追加してみよ。
- (ALTER TABLE xx ADD UNIQUE)
- これらに重複のあるデータ追加を行ってみてエラー確認。

A.3 単一表内の検索

- テーブル「月表」を作る。属性は月番号 (整数)、うるう年 (論理値)、月名 (文字列)、日数 (整数)。「月番号+うるう年」が主キー、日数は 28〜31 の範囲。
- データをファイルに用意した (/u1/kuno/wrok/month.data) ので、これを読み込んでデータをロード。以下はすべて SELECT 使用。
- 「1 月」の月名を表示せよ。1 件だけ表示させるにはどうする?
- 「大の月名」を月番号順に表示。
- 「小の月名」を ABC 順に表示。
- 「先頭が J で始まる月の月名」を ABC 順に表示。(LIKE を使う)
- 「月の日数としては何があるか」を大きい順に表示。
- 「うるう年の月番号と日数の一覧」を月番号の順に表示。

[1] 英語の曜日名は Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday である。知らない人はいないと思うが。

- 「うるう年の日数合計」を表示。
- 「うるう年とそうでない年それぞれの月あたり平均日数」を表示。
- 「うるう年とそうでない年で日数が違う月の名前」を表示。

A.4 ビューの作成

- 月表からビュー「月名表」を作る。これは月番号と月名だけの表。(CREATE VIEW を使う)

A.5 副問い合わせと集合演算

- テーブル「誕生日」を作る。属性は年 (整数)、月 (整数)、日 (整数)、名前 (文字列 20 文字)。
- データをファイルに用意した (`/u1/kuno/work/birth.data`) ので、これを読み込んでデータをロード。どんなデータか見ておく。
- 1965 年の大の月に生まれた人名の一覧を年、月とともに表示。
- 月ごとの生まれた人数 (もちろんんこのデータ中で) のビューを作る。名前は「月毎生まれ人数」。属性は「月」と「数」。
- 最も生まれた人が少ない月を表示。
- 最も生まれた人が少ない月に生まれた人の一覧を年、月とともに表示。
- 月名と英曜日名の合わさった一覧を ABC 順に表示。意味はないけど。

A.6 結合

- 「誕生日」に加えて月名が名前で入った表を表示せよ (簡単)。
- 同じ年、月、日に 2 人以上が生まれている場合に、それらの人の生まれた年、月、日、名前を表示せよ (ヒント:誕生日を 2 回結合)
- すべての月名 2 つの組みを表示せよ。ただし同名の月の組みは除く。(ヒント:交差結合)
- 1965 年に生まれた人すべてについて、1965 年以外でそれと同じ月・日に生まれた人の生まれた年と名前を並べて表示せよ。そのような人がいない場合は空欄にせよ。(ヒント:左外結合)