データベース講習会（初級編）

於　Zoom オンライン開催
主催 天文データセンター
講師 小澤 武揚（天文データセンター）

データベース講習会
2021年12月21―22日

目次

1. はじめに

自己紹介

講師近影

名前：小澤 武揚 (33)
おざわ たけあき

所属：天文データセンター

仕事：

- Tomo-e Gozen データのSMOKAでの公開作業
- 多波長データ解析システムのシステム情報データベース化
- 計算機室の環境情報のデータベース化
- データベースサーバの性能評価試験

1.1. 背景

- 観測機器の高性能化に伴うデータの大規模化によって高速な検索や統計処理の必要性が生じ、個人や各研究所でローカルにデータベースをもつ必要性が生じてくるケースが増えてくると考えられる。
- 大規模なデータの検索や集計は従来データセンターの役目であったが、計算機の高性能化と低廉化により個人においてもデータベースを構築・活用できるようになってきた。
- ローカルにデータベースを持つことで必要な付加情報を加えたオリジナルのカタログを作成し、効率的にデータの検索や統計処理を行うことができる。

1.2. 本講習会の目的

- リレーショナルデータベース管理システムであるPostgreSQLのインストール方法・初期設定方法・データベース構築方法・問い合わせ方法をLinux端末を操作して実践的に学び習得
- 天文カタログをデータベース化し、天文データベースのノウハウを習得
  - PGC2003カタログ：983,261 天体
  - NVSSカタログ：　1,773,484 天体

1.3. 参考資料

- 天文データセンター 2012年度SQL講習会[初級編]資料（山内 千里）
- 天文データセンター 2016年度SQL講習会[中級編：データベース構築編]資料（山内 千里）
- PostgreSQL本家（https://www.postgresql.org）
- 日本PostgreSQLユーザ会（https://www.postgresql.jp）
- PostgreSQL 13.1文書（https://www.postgresql.jp/document/13/html/index.html）
- Let’s Postgres（https://lets.postgresql.jp）
- 基礎から始めるデータベース入門セミナー（https://www.oracle.com/technetwork/jp/articles/index-155234-ja.html）
- PostgreSQL 全機能バイブル（技術評論社）（PostgreSQL 9.2 対応）
- 内部構造から学ぶPostgreSQL 設計・運用計画の鉄則（技術評論社）

2. データベースの基礎

2.1. データベースとは

❖データベースとは

- 系統的に整理・管理された情報の集まり。特にコンピューターで、さまざまな情報検索に高速に対応できるように大量のデータを統一的に管理したファイル。また、そのファイルを管理するシステム（広辞苑第五版より）。
- 整理・統合して蓄積したデータを検索できるようにした仕組みのこと。

❖データベースの例

- 辞書、電話帳、ウェブ検索システム、販売時点情報管理（POS）システム、銀行の勘定系システム・・・。
- 観測データ、天体カタログ、文献・・・。

2.2. データベースの種類

❖データベースは主に次の3種類に分類

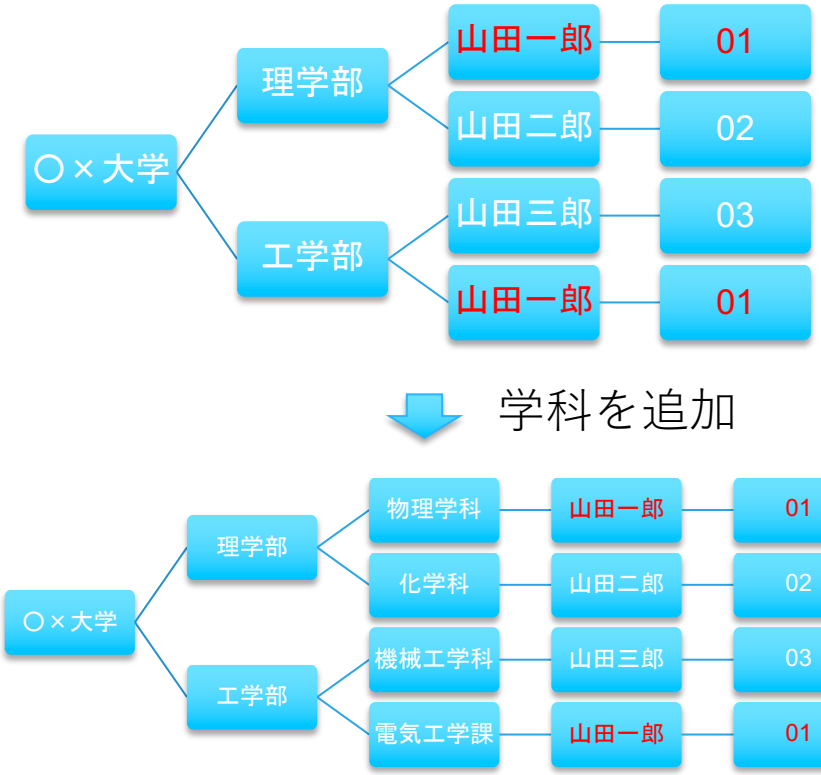
- 階層型データベース
- ネットワーク型データベース
- リレーショナルデータベース

❖階層型データベース

- **データを親子関係を持った階層構造で管理するデータベース**

- **利点**
  - 検索ルートが限られており高速に検索可能。
- **欠点**
  - 子データが複数の親データに属する場合データが重複。
  - データを追加したときに階層構造の再設計が必要。
  - 階層構造が変化した際にデータベースを利用するプログラムの修正が必要。

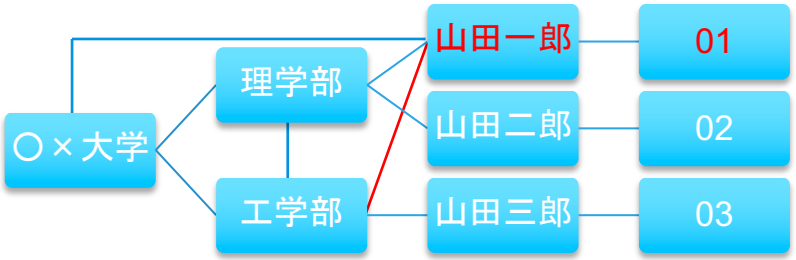
教員データベース



❖ネットワーク型データベース

- **データを網の目構造で管理するデータベース**
- **利点**
  - データの冗長化がおこならない。
- **欠点**
  - データ同士の関係が複雑。
  - データを追加したときにデータベース構造の再設計が必要。
  - 階層構造が変化した際にデータベースを利用するプログラムの修正が必要。

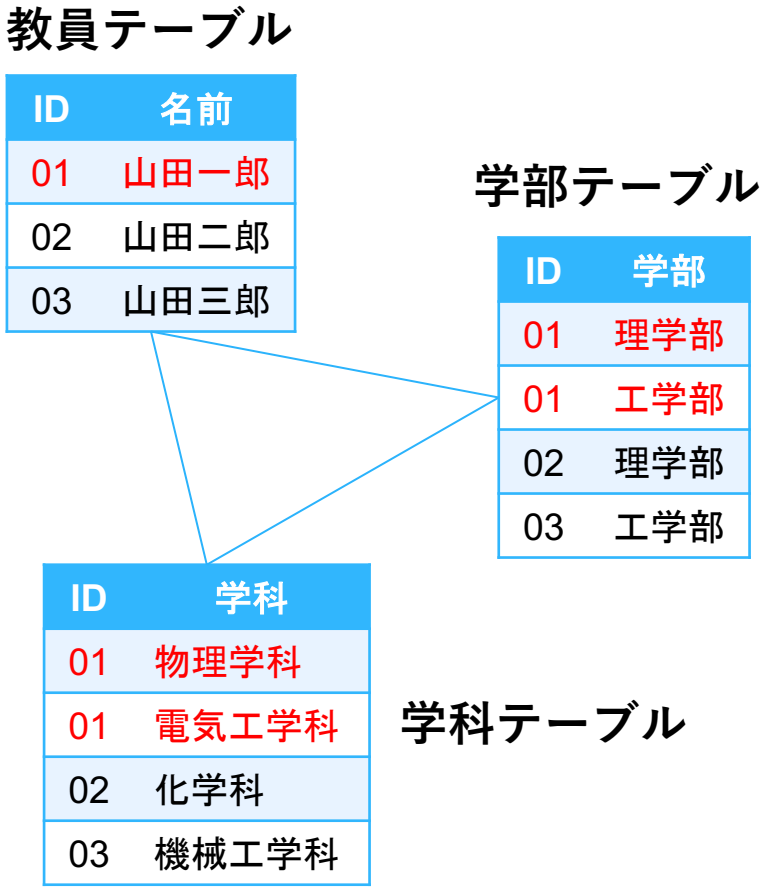
教員データベース



❖リレーショナルデータベース

- **データを行と列からなる表で管理するデータベース**
  - 表と表に共通するデータをもたせてデータを関連づける。

- **利点**
  - データ構造に変更が生じても、プログラムへの影響が少ない。
- **欠点**
  - どのようなデータであっても表形式にしなければならない。
  - 設計方法でパフォーマンスに差が出る。



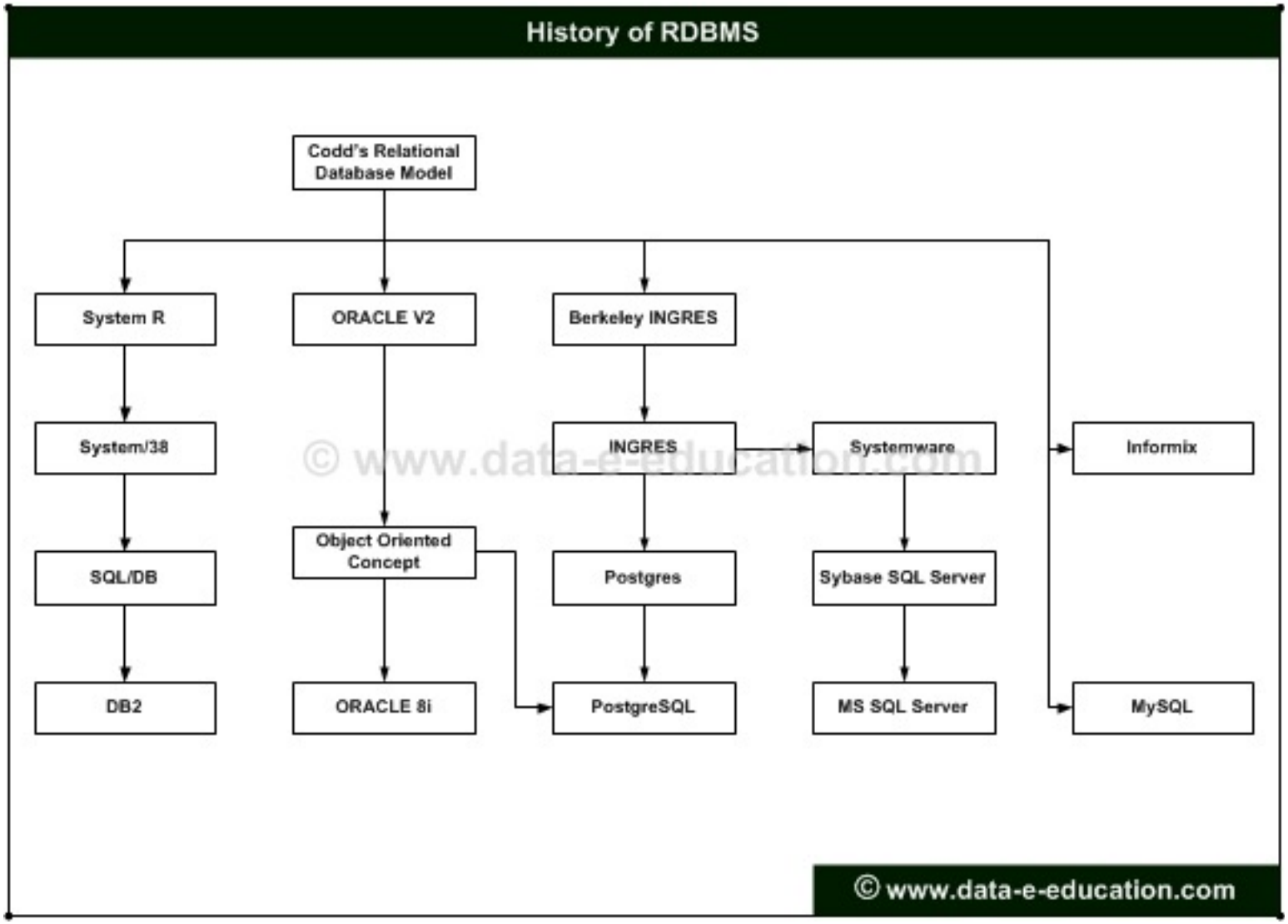
2.3. リレーショナルデータベース管理システム

❖RDBMS（Relational Data Base Management System）

- リレーショナルデータベースをコンピュータ上で実現するためのソフトウェア。
  - 有償のものと無償のものが存在。
    - > 有償：Oracle Database、Microsoft SQL Server、IBM DB2
    - > 無償：MySQL、PostgreSQL、MariaDB
- クライアント・サーバモデルを採用しており、クライアントから要求された検索を、データベースを持つサーバが実行。
- データベースへの検索要求にはSQL（言語）を使用。

❖RDBMSの系譜

- http://data-e-education.com/E107_History_of_RDBMS.html
  - 参照元はリンク切れで閲覧不可。



2.4. SQL

❖SQL（Structured Query Language）

- リレーショナルデータベースにおいて、データの定義や問い合わせを行うための言語。
- 米国規格協会（ANSI）と国際標準化機構（ISO）により標準SQL規格が策定（最新： SQL:2016）。
- RDBMSは基本的に標準SQL規格に則るため、RDBMS間で使用するSQLが大きく変わることはない。
  - 関数に関しては互換性が無いことが多々ある。

3. PostgreSQLについて

3.1. PostgreSQLとは

❖PostgreSQLとは

- オープンソースソフトウェア（OSS）のオブジェクトリレーショナルデータベース管理システム（ORDBMS）。
- PostgreSQL Global Development Group（PGDG）が開発。
  - PGDG：7人のコアメンバー、39人の主要開発者（日本人5名）、数十名の貢献者で構成（2021年10月現在）。
- 日本では「ポストグレスキューエル」、「ポストグレス」、「ポスグレ」と呼ばれる。
- 日本では2009年前後までOSSのRDBMSの中でトップシェアだったが、現在はMySQLがトップシェア。
  - 参考：第3回オープンソースソフトウエア活用ビジネス実態調査

3.2. PostgreSQL小史

❖ PostgreSQL小史

- PostgreSQLは最初期のRDBMSであるIngresが発端。
- 1986年にカリフォルニア大学バークレー校（UCB）でIngresにオブジェクト指向を取り入れたPOSTGRES（Post-Ingres）の開発が開始され、1993年に公開されたPOSTGRES 4.2をもって開発が終了。
- 1995年にUCBの大学院生であったAndrew YuとJolly ChenによってSQL言語インタプリタが追加されたPostgres 95が公開。
- 1997年にSQLに対応したことを踏まえて名称がPostgreSQLに改称され、UCB時代のバージョン番号を引き継いだPostgreSQL 6.0が公開。同時期にPGDGが結成され開発体制が確立。
- 1999年に日本PostgreSQLユーザ会（JPUG）が設立。
- 2021年12月現在ではPostgreSQL 14が公開中。

3.3. PostgreSQLの特徴

- **個人・商用問わず無償で利用・配布・改変可能**
- **豊富な資料**
  - **公式ドキュメント**
    - 基礎的な内容から応用的な内容まで多くの例題を使って解説がなされており、これを読めば分からないことはない（かもしれない）。
    - JPUGによって全文日本語訳されている。
  - **ウェブ資料**
    - 歴史的に日本ではコアなユーザが多く、資料が充実。
- **公開サービス向きの機能**
  - **豊富なクライアントインターフェース**
    - C、Java、JavaScript、Perl、PHP、Python、Ruby他、多くの言語からアクセス可能。
  - **商用版のPowerGresが存在**
    - ソフトウェアが出荷されてから7年間サポートが有るため、長期間同じ環境を維持したい場合などに有用（PostgreSQLは5年間）。

❖天文データ検索向きの機能

- **強力なユーザ定義関数**
  - 複数の戻り値（行、列、行列）を返す関数を定義可能。
    - > ORDBMSであるPostgreSQLの大きな特徴。
    - > 通常RDBMSの関数は戻り値として1つの値しか返せない。
  - 座標に対する複雑な計算を頻繁におこなう天文データ検索に有用。
- **強力なユーザ定義関数の開発環境**
  - SQL言語、PL/pgSQL、C言語、PL/Tcl、PL/Perl、PL/Python他、多くの言語でユーザ定義関数を作成可能。
- **巨大なテーブルに対応する機能**
  - 式インデックスや部分インデックスが使用可能。
  - テーブルパーティショニング（テーブルの分割化）が可能。

3.4. 対応するプラットフォーム

❖以下のプラットフォームで動作する

- **CPUアーキテクチャ**
  - X86、X86_64、IA64、PowerPC、PowerPC 64、S/390、S/390x、Sparc、Sparc 64、ARM、MIPS、MIPSEL、PA-RISC等。
- **OS**
  - Linux（最近のディストリビューション全て）。
  - Windows（Win2000 SP4以降）。
  - FreeBSD、OpenBSD、NetBSD。
  - OS X。
  - AIX、HP/UX、Solaris 等。

4. PostgreSQLのインストール

4.1. 概要

❖ LinuxへのPostgreSQLのインストール方法を紹介

- 本講習会の実施環境へはインストール済み。

❖ インストール環境

- OS：CentOS Stream release 8
  - ソフトウェアの選択：最小限のインストール + 標準
- おこなった設定
  - # dnf update
  - # dnf install gcc
  - # dnf install emacs
  - # localectl set-locale LANG=ja_JP.UTF-8; source /etc/locale.conf

❖ 4章目次

- 4.2. PostgreSQLのインストール
- 4.3. データベースクラスタの作成
- 4.4. PostgreSQLの起動と終了

4.2. PostgreSQLのインストール

❖本家ウェブサイトからPostgreSQLをダウンロード

1. 適当な環境でウェブブラウザを起動。
2. PostgreSQLの本家ウェブサイトにアクセス。

```
https://www.postgresql.org
```

3. ページ上部の「Download」をクリック。
4. 「Packages and Installers」で「Linux」をクリック。続いて「Red Hat/Rocky/CentOS」をクリック。
5. 「PostgreSQL Yum Repository」の選択欄を次のように選択。

```
1. Select version:13
2. Select platform:Red Hat Enterprise, Rocky, CentOS or
   Oracle Version 8
3. Select architecture:x86_64
```

PostgreSQL Yum Repository

The **PostgreSQL Yum Repository** will integrate with your normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the support **lifetime** of PostgreSQL.

The PostgreSQL Yum Repository currently supports:

- Red Hat Enterprise Linux
- Rocky Linux
- CentOS
- Scientific Linux
- Oracle Linux
- Fedora*

***Note**: due to the shorter support cycle on Fedora, all supported versions of PostgreSQL are not available on this platform. We do not recommend using Fedora for server deployments.

To use the PostgreSQL Yum Repository, follow these steps:

1. Select version:
   13
2. Select platform:
   Red Hat Enterprise, Rocky, CentOS or Oracle version 8
3. Select architecture:
   x86_64
4. Copy, paste and run the relevant parts of the setup script:

Copy Script

```
# Install the repository RPM:
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm

# Disable the built-in PostgreSQL module:
sudo dnf -qy module disable postgresql

# Install PostgreSQL:
sudo dnf install -y postgresql13-server

# Optionally initialize the database and enable automatic start:
sudo /usr/pgsql-13/bin/postgresql-13-setup initdb
sudo systemctl enable postgresql-13
sudo systemctl start postgresql-13
```

❖PostgreSQLのインストール

1. PostgreSQLをインストールする環境にリポジトリを登録。

```
[root]# dnf install https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-ppc64le/pgdg-redhat-repo-latest.noarch.rpm
```

2. dnfのPostgreSQLのモジュールの無効化。

```
[root]# dnf module disable postgresql
```

3. PostgreSQLのパッケージをインストール。

```
[root]# dnf install postgresql13
                    postgresql13-server
                    postgresql13-devel
                    postgresql13-contrib
                    postgresql13-libs
```

❖インストールすべきパッケージ

- **postgresql13**
  - データベースサーバにアクセスするために必要なクライアントソフトウェア。
- **postgresql13-server**
  - データベースサーバを構築・管理するために必要なソフトウェア。
- **postgresql13-devel**
  - アプリケーション開発用ヘッダファイルやライブラリ。
- **postgresql13-contrib**
  - OSコマンドや関数など。
- **postgresql13-libs**
  - PostgreSQLのクライアントプログラム用のライブラリやインターフェース。

4.3. データベースクラスタの作成

❖データベースクラスタ

- データベースを格納するディレクトリ。
- PostgreSQLサーバ用Linuxコマンド「initdb」で作成。
- 未作成だとPostgreSQL起動不可。

❖データベースクラスタの作成

1. コマンド「initdb」でデータベースクラスタを作成。

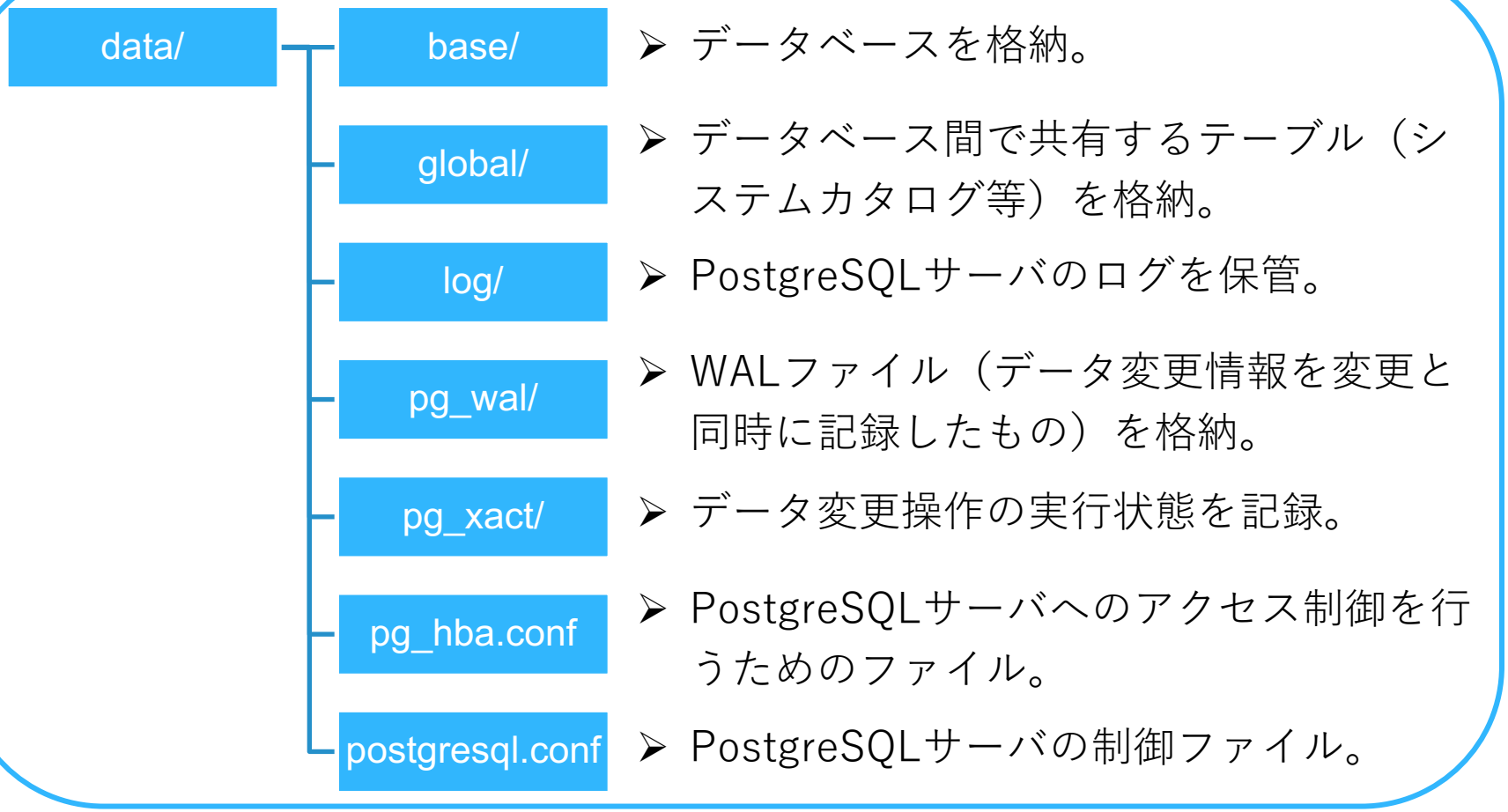
```
[root]# su postgres
bash-4.4$ /usr/pgsql-13/bin/initdb
          --no-locale
          --encoding=UTF8
          -D /var/lib/pgsql/13/data
```

- **Linuxユーザ「postgres」**
  - PostgreSQL13-serverインストール時に自動的に作成されるLinuxユーザ。
  - ホームディレクトリ：「/var/lib/pgsql」。
- **--no-locale**
  - PostgreSQLサーバの地域化を解除。
  - デフォルトはOSで設定されているロケール（講習会環境ではja_JP）。並び替え、日付や通貨の表示、エラーメッセージの言語などが地域化。
  - LIKE句（パターンマッチング）でインデックスが使用されず検索速度が悪化。
- **データベースの初期化**
  - データベースクラスタ（/var/lib/pgsql/13/data）を削除することでデータベースを完全に初期化可能。

❖データベースクラスタの構造

- 主要なディレクトリとファイルについて。

```
[root]# ls -la /var/lib/pgsql/13/data
```



4.4. PostgreSQLの起動と終了

❖PostgreSQLの起動

1. PostgreSQLの起動。

```
[root]# systemctl start postgresql-13.service
```

2. 状態確認。

```
[root]# systemctl status postgresql-13.service
```

3. 自動起動設定。

```
[root]# systemctl enable postgresql-13.service
```

- CentOSでは自動起動設定をおこなわないと、OS起動時に自動的にPostgreSQLが立ち上がらない。
- **systemctl**
  - CentOSにおけるサービス管理コマンド。
  - stop：サービス終了。
  - disable：自動起動設定解除。

5. PostgreSQLの初期設定

作業環境の確認

❖初期設定方法を紹介する前に、本講習会の作業環境の確認をおこないます

❖【実習】 作業環境の確認

1. 事前にお知らせした手順に従い、SSH接続で講習会用サーバに接続。
2. 端末にコマンドを入力し環境を確認。

```
[schoolXX]$ ls DB
```

DB ┬ catalogue
　 └ sql

- catalogue：天文カタログを収めたディレクトリ。
- sql：作成したsqlファイルを収めるディレクトリ。

```
[schoolXX]$ su postgres
bash-4.4$ whoami
bash-4.4$ exit
```

- デフォルトではアカウントがロックされているため一般ユーザからLinuxユーザ「postgres」にはスイッチできないが、本講習会環境ではコマンド「# passwd -u -f postgres」でロックを解除している。

5.1. 概要

❖PostgreSQLの初期設定方法を紹介

- 本講習会の実施環境へは設定済み。

❖5章目次

- 5.2. データベースロールの設定
- 5.3. PostgreSQLサーバ制御ファイルの設定
- 5.4. アクセス制御ファイルの設定

5.2. データベースロールの設定

❖ データベースロール

- PostgreSQLサーバ用のユーザのようなもの。
- 閲覧専用等、役割に応じて作成するのが一般的。

❖ 設定するロール

- **postgres**
  - データベースクラスタ作成時に作成される特権ロール。
  - 初期状態ではパスワードが設定されていないので設定。
- **dbr**
  - 一般権限のデータベースロールを作成。

❖ 5.2節目次

5.2.1. ロール「postgres」のパスワード設定

❖ロール「postgres」のパスワードの設定

1. データベース「postgres」に接続。

```
[schoolXX]$ psql -U postgres postgres
```

- psql：PostgreSQLの対話型インターフェース。7章で詳しく説明。
- -U postgres：特権ロール「postgres」で接続。
- postgres：データベースクラスタ作成時に自動的に作成されるデータベース。主に特権が必要な操作を行うときに利用する。

2. 「ALTER ROLE」コマンドでパスワードを変更。

```
postgres=# ALTER ROLE postgres WITH PASSWORD 'postgres';
```

- ALTER ROLE：ロールの属性を変更するためのSQLコマンド。

3. 「¥q」コマンドでpsqlを終了。

```
postgres=# ¥q
```

- 「¥」はバックスラッシュ「＼」。
- アクセス制御ファイルが初期状態の場合無条件にデータベースへの接続を許可しているため、この時点ではパスワードを求められない。

5.2.2. ロール「dbr」の作成

❖ ロール「dbr」の作成

1. データベース「postgres」に接続。

```
[schoolXX]$ psql -U postgres
```

   - 接続先を省略するとロールと同名のデータベースへ接続される。

2. 「CREATE ROLE」コマンドでロールを作成。

```
postgres=# CREATE ROLE dbr LOGIN PASSWORD 'dbr';
```

   - CREATE ROLE：ロールを新規作成するためのSQLコマンド。

3. ロールが作成できたか確認

```
postgres=# ¥du
```

   - ¥du：ロールの一覧を表示するpsqlメタコマンド。

4. 「¥q」コマンドでpsqlを終了。

```
postgres=# ¥q
```

❖【実習】データベースへのログイン

1. データベース「postgres」にロール「postgres」で接続してみましょう。

```
[schoolXX]$ psql –U postgres
postgres=# ¥x
postgres=# SELECT * FROM pg_roles
           WHERE rolname='postgres';
postgres=# SELECT * FROM pg_roles
           WHERE rolname='dbr';
postgres=# ¥q
```

- ¥x：表示形式を変更するpsqlメタコマンド。
- コマンドと任意値を区別するため「SELECT~」で大文字を使っているが、psqlでは大文字小文字の区別がないため小文字入力でOK。
- 上矢印キーでコマンドの履歴を表示可能。

2. データベース「postgres」にロール「dbr」で接続してみましょう。

```
[schoolXX]$ psql –U dbr postgres
postgres=> CREATE ROLE tukurenaiyo PASSWORD 'hoge';
postgres=> ¥q
```

- 本講習会の実施環境ではアクセス制御ファイルの設定が完了しているためパスワードの入力を求められる。

5.3. PostgreSQLサーバ制御ファイルの設定

❖postgresql.conf

- 在り処：「/var/lib/pgsql/13/data/postgresql.conf」。

❖設定項目

- FILE LOCATIONS
- CONNECTIONS AND AUTHENTICATION
- RESOURCE USAGE
- WRITE-AHEAD LOG
- REPLICATION
- QUERY TUNING
- REPORTING AND LOGGING
- PROCESS TITLE
- STATISTICS
- AUTOVACUUM
- CLIENT CONNECTION DEFAULTS
- LOCK MANAGEMENT
- VERSION AND PLATFORM COMPATIBILITY
- ERROR HANDLING
- CONFIG FILE INCLUDES

❖5.3節目次

5.3.1. 設定変更すべき項目

❖ 設定変更すべき項目

- ほとんどの項目は初期設定のままで問題ない。
- デフォルトのメモリ使用量は保守的で本来の性能を発揮できないため、現在のマシンスペックに合わせた設定変更が必要。
  - 本講習会環境のメモリ量は4GB。

❖ RESOURCE USAGE

- **shared_buffers（初期値：128MB => 1024MB）**
  - テーブルやインデックスのデータをキャッシュする領域。
  - 膨大なデータから少数のデータを読み書きする処理が頻繁におこなわれる処理に影響。
  - カーネルがキャッシュしたデータをPostgreSQLのshared_buffresが読み込むため、shared_buffersに大きな値を割り当てるとキャッシュするデータのほとんどが重複し、メモリの利用効率が悪化する。
  - 経験的にshared_buffersのサイズは「実メモリの20~30%程度、もしくは頻繁にアクセスするテーブルのデータがshared_buffersに載る程度」とされる。

- **work_mem（初期値：4MB => ~~1024MB~~）30MB**
  - 検索時にテーブル結合や並び替え操作をおこなうための領域。
  - 多くのレコードにアクセスする統計処理や、多くのレコードからデータを取り出す処理に影響。
  - work_memの値は最大でも「（実メモリ-shared_buffers）/max_connections（初期値：100）」程度とし、スワップが発生しない様に設定する。
- **maintenance_work_mem（初期値：64MB => 1024MB）**
  - バキュームやインデックス作成等のメンテナンス時に使用される。
- **effective_cache_size（初期値：4GB => 2GB）**
  - ディスクキャッシュに利用できるメモリ量を予想するために用いられる（実メモリは消費しない）。
  - 実メモリの50%程度が良いとされる。

5.3.2. postgresql.confの設定

❖postgresql.confの設定

1. 「su」コマンドでLinuxユーザ「postgres」にスイッチ。

```
[root]# su postgres
```

2. テキストエディタで「postgresql.conf」を開く。

```
bash-4.4$ emacs /var/lib/pgsql/13/data/postgresql.conf
```

3. 「postgresql.conf」を以下のように編集（行頭の「#」は消す）。

```
shared_buffers = 1024MB
work_mem = 1024MB 30MB
maintenance_work_mem = 1024MB
effective_cache_size = 2GB
```

4. Linuxユーザ「root」でPostgreSQLを再起動。

```
bash-4.4$ exit
[root]# systemctl restart postgresql-13.service
[root]# exit
```

5.4. アクセス制御ファイルの設定

❖**pg_hba.conf**（hbaは「Host Based Authentication」の略）

- 在り処：「/var/lib/pgsql/13/data/pg_hba.conf」。

❖**個人環境で利用することを想定した設定**

- **外部からの接続は不許可**
  - PostgreSQLサーバをインストールしたサーバにログインしないとデータベースに接続できない
- **ローカルからの接続に対するパスワード認証を実施**
  - サーバにログインしないとデータベースに接続できないので本来パスワード認証は不要だが、練習のため設定。

❖**5.4節目次**

5.4.1. pg_hba.confの書式

❖pg_hba.confの書式

# TYPE	DATABASE	USER	ADDRESS	METHOD
local	all	all		trust
host	db01	dbr	192.168.0.0/24	md5

- **TYPE**
  - local：ローカルホストに対するアクセス制御。
  - host：リモートホストに対するアクセス制御。
- **DATABASE**
  - データベース名: 指定したデータベースのみ接続可能（all指定可能）。
- **USER**
  - ロール名: 指定したロールのみ接続可能（all指定可能）。
- **ADDRESS**
  - 指定したネットワークアドレスの機器のみ接続可能（localの場合は不要）。
- **METHOD**
  - trust：無条件に接続を許可。
  - reject：無条件に接続を拒否。
  - md5：md5で暗号化されたパスワードによる認証。

5.4.2. pg_hba.confの設定

❖pg_hba.confの設定

1. 「su」コマンドでLinuxユーザ「postgres」にスイッチ。

```
[root]# su postgres
```

2. テキストエディタで「pg_hba.conf」を開く。

```
bash-4.4$ emacs /var/lib/pgsql/13/data/pg_hba.conf
```

3. 「pg_hba.conf」を以下のように編集。

```
local all             all                     md5
host  all             all 127.0.0.1/32        md5
host  all             all ::1/128             md5
local replication     all                     reject
host  replication     all 127.0.0.1/32        reject
host  replication     all ::1/128             reject
```

- 初期状態ではMETHODが全てtrust。

4. PostgreSQLを再起動。

```
bash-4.4$ exit
[root]# systemctl restart postgresql-13.service
[root]# exit
```

5. パスワード認証が有効化されたか確認。

```
[schoolXX]$ psql –U postgres
ユーザー postgres のパスワード: postgres
postgres=> ¥q

[schoolXX]$ psql -U dbr postgres
ユーザー dbr のパスワード: dbr
postgres=> ¥q
```

6. データベースの作成

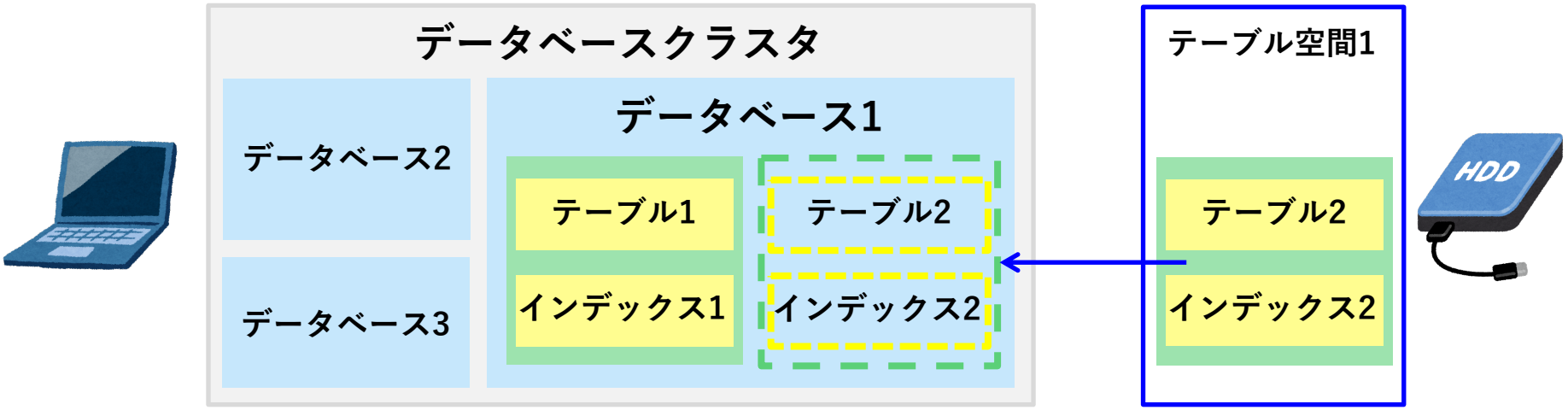
6.1. 概要

❖天体カタログ「PGC2003」と「NVSS」のデータベース化を実施

❖6章目次

- 6.2. テーブル空間の作成
- 6.3. データベースの作成
- 6.4. テーブルの作成
- 6.5. インデックスの作成
- 6.6. 演習問題：NVSSカタログのデータベース化

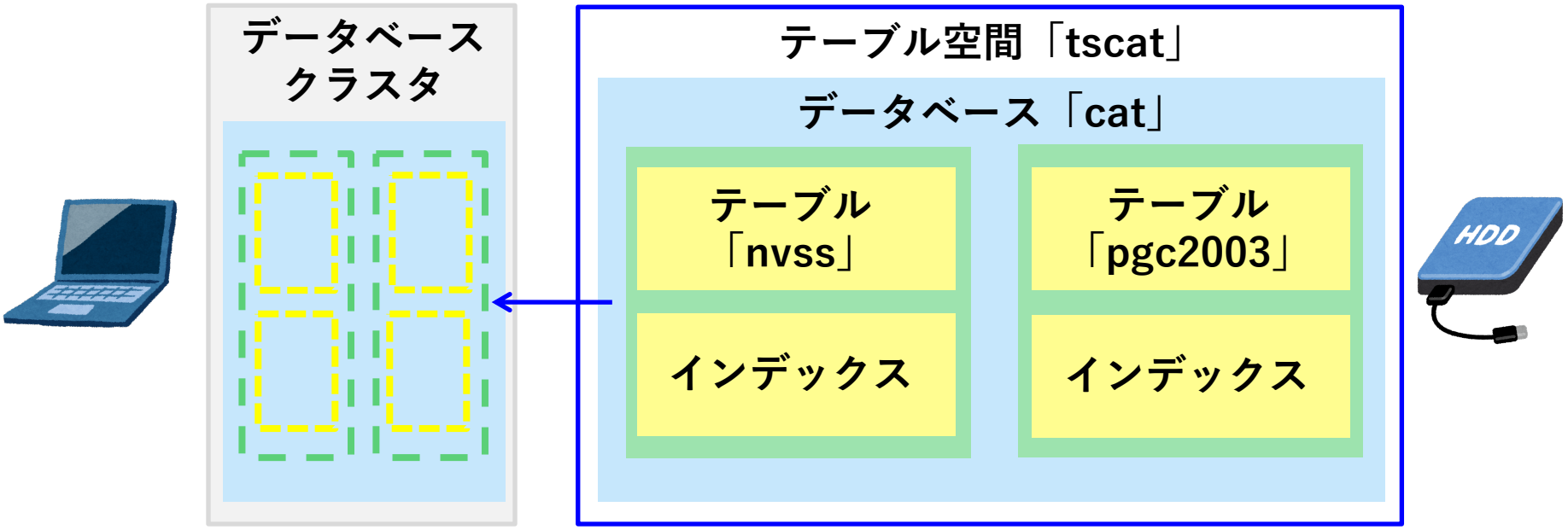
❖データベースの構造



- **データベースクラスタ**
  - PostgreSQLサーバの全情報が格納されたディレクトリ。
- **データベース**
  - オブジェクト（テーブル、インデックス等）の集合に名前を付けたもの。
  - 物理的にはオブジェクトファイルが格納されたディレクトリ（data/base/）。
  - データベースを跨いでのデータのやりとりはできない。
- **テーブル**
  - データを格納する表。
  - 物理的にはデータが記述されたファイル群。
- **テーブル空間**
  - データベースやテーブルを格納できるディレクトリ。
  - データベースクラスタ外にデータベースやテーブルを配置可能。

どのDBに対応しているのかは
SELECT * FROM pg_database;
で確認可能

❖作成するデータベースの構造



- PCに外付けHDDを備えた個人環境にデータベースを構築することを想定。
- ディレクトリ「/hdd/」以下にテーブル空間を作成し、テーブル空間内にデータベースを作成。

6.2. テーブル空間の作成

❖CREATE TABLESPACE文の書式

```
CREATE TABLESPACE <テーブル空間名>
OWNER <ロール名>
LOCATION <テーブル空間を作成する場所のパス>;
```

❖【実習】 テーブル空間の作成

■ ディレクトリ「/hdd/」以下にロール「dbr」が所有者であるテーブル空間「tscat」を作成

1. ディレクトリ「/hdd/」の確認。

```
[schoolXX]$ ls -ld /hdd
drwx------.  2 postgres postgres    6 11月 24 16:41 hdd
```

- テーブル空間用ディレクトリは、Linuxユーザ「postgres」が所有していなければならない。

2. ロール「postgres」でデータベース「postgres」に接続。

```
[schoolXX]$ psql -U postgres
```

- ロール「dbr」にはテーブル空間の作成権限がない。

3. テーブル空間を作成。

```
postgres=# CREATE TABLESPACE tscat
           OWNER dbr
           LOCATION '/hdd';
```

4. 作成したテーブル空間をpsqlメタコマンド「¥db」で確認。

```
postgres=# ¥db
postgres=# ¥q
```

❖ テーブル空間の削除方法

```
postgres=# DROP TABLESPACE tscat;
```

- テーブル空間を所有するロールでもテーブル空間を削除可能。
- テーブル空間内にデータベースオブジェクトが存在する場合は削除できない。

6.3. データベースの作成

❖CREATE DATABASE文の書式

```
CREATE DATABASE <データベース名>
OWNER <ロール名>
[TABLESPACE <テーブル空間名>];
```

❖【実習】 データベースの作成

- **ロール「dbr」が所有するデータベース「cat」をテーブル空間「tscat」内に作成**

1. ロール「postgres」でデータベース「postgres」に接続。

```
[schoolXX]$ psql -U postgres
```

- ロール「dbr」にはデータベースの作成権限がない。

2. データベースを作成。

```
postgres=# CREATE DATABASE cat
           OWNER dbr
           TABLESPACE tscat;
```

3. 作成したデータベースをpsqlメタコマンド「¥l」で確認。

```
postgres=# ¥l
postgres=# ¥q
```

4. ロール「dbr」によるデータベースへの接続確認。

```
[schoolXX]$ psql -U dbr cat
cat=> ¥q
```

5. ディレクトリ「/hdd/」の確認。

```
[schoolXX]$ su postgres
bash-4.4$ ls /hdd
```

❖ データベースの削除方法

```
postgres=# DROP DATABASE cat;
```

- データベースを所有するロールでもデータベースを削除可能。

6.4. テーブルの作成

- ❖天体カタログ「PGC2003」を収めるテーブルの作成とデータの登録を実施
- ❖6.4節目次
  - 6.4.1. テーブルの構造
  - 6.4.2. PostgreSQLがサポートするデータ型
  - 6.4.3. PGC2003
  - 6.4.4. テーブルの作成
  - 6.4.5. データの登録

6.4.1. テーブルの構造

- **リレーショナルデータベースのテーブルは行（row）と列（column）から構成される**
  - **行（row）**
    - データベースにおけるデータそのもの。
  - **列（column）**
    - 列毎にデータ型が定義され、データ型と異なる値は入力できない。
    - 列毎に制約を定義でき、制約に違反した値は入力できない。
  - **行と列をRecordやFieldと言う場合もあるが定義が曖昧なので個人的に非推奨**

			column	
	id	ra	dc	mag
	1	83.625	+22.016	8.4
row	2	323.375	-0.816	6.3
	3	205.550	+31.716	6.3

6.4.2. PostgreSQLがサポートするデータ型

❖ **PostgreSQLは以下のデータ型をサポート**

- **数値データ型**
- **文字データ型**
- **日付・時刻データ型**
- ブーリアン型
- 幾何データ型
- ネットワークアドレスデータ型
- ビット列データ型
- 通貨データ型
- 疑似データ型
- バイナリ列データ型
- 文字列検索型
- XML型
- Range型

❖数値データ型

データ型名	格納サイズ	説明	範囲
SMALLINT	2バイト	半精度整数型	-32768から+32767
INTEGER	4バイト	単精度整数型	-2147483648から+2147483647
BIGINT	8バイト	倍精度整数型	-9223372036854775808から +9223372036854775807
REAL	4バイト	単精度浮動小数点数型	6桁精度 -1E37から+1E+37
DOUBLE PRECISION	8バイト	倍精度浮動小数点数型	15桁精度 -1E308から+1E308
NUMERIC、 DECIMAL	可変長	任意精度固定小数点数型	小数点より上は131072桁まで、 小数点より下は16383桁まで

- **SMALLINT、INTEGER、BIGINT**
  - 必要に応じて使い分ける。
- **REAL、DOUBLE PRECISION**
  - 浮動小数点数型であるため丸め誤差が発生する。
  - 正確な記録と計算が必要な時はNUMERICかDECIMALを使用する。
- **NUMERIC[(p,s)]、DECIMAL[(p,s)]**
  - DECIMALはNUMERICの別名。
  - 精度（全体の桁数）と小数部の桁数が指定可能で、正確な計算が可能。
  - 整数型及び浮動小数点数型に対し、計算が非常に遅い。

❖文字データ型

データ型名	格納文字数	説明
TEXT	無制限	文字数制限の無い可変長文字列型。
CHARACTER VARYING(N)、VARCHAR(N)	N文字以下	最大N文字の可変長文字列型。
CHARACTER(N)、CHAR(N)	N文字	N文字の固定長文字列型

- **TEXT**
  - 長さに制限なく文字列を扱える（ただし、PostgreSQLが1行に収めることのできる最大バイト数は1ギガバイト）。
  - TEXT型は標準SQLにはないが、多くのRDBMSがサポート。
- **CHARACTER VARYING(N)、VARCHAR(N)**
  - VARCHARはCHARACTER VARYINGの別名。
  - 最大n文字まで文字列を格納できる。
  - PostgreSQL内部ではTEXT型と区別がなく、データ入力時に文字列長を確認する点だけが異なる。
- **CHARACTER(N)、CHAR(N)**
  - CHARはCHARACTERの別名。
  - 最大n文字までの文字列を格納できる。
  - 入力文字数がN文字に満たない場合は、残りの部分が空白文字で埋められる。
  - 上記のデータ型よりも処理速度が遅く、理由がない限り使う必要はない。

❖日付・時刻データ型

データ型名	格納サイズ	説明	精度
TIMESTAMP [(P)] [WITH TIME ZONE]	8バイト	YYYY-MM-DD hh:mm:ss [+/-hh]	1マイクロ秒
DATE	4バイト	YYYY-MM-DD	1日
TIME [(P)]	8バイト	hh:mm:ss	1マイクロ秒

※(P)：Pは0から6までの整数。秒単位以下の表示精度を指定。

- **TIMESTAMP、DATE、TIME**
  - それぞれ「年月日時分秒」、「年月日」、「時分秒」のデータを格納可能。
- **WITH TIME ZONE オプション**
  - 日時をUTCで保存。
  - 日時の入出力時に時間帯を指定可能。
  - 時間帯を指定せずに日時を入力した場合、postgresql.confで設定された時間帯を参照して入力値をUTCに変換して格納。
  - 時間帯を指定せずに日時を出力した場合、格納されているUTC日時をpostgresql.confで設定された時間帯に変換して出力

6.4.3. PGC2003

❖ Principal Galaxy Catalog 2003（Paturel+2003）

- HYPERLEDAデータベースの基となっているカタログ。
- 18 B-mag以上の銀河を約100万収録。
- 各銀河の座標、直径、楕円率、位置角、50個のカタログにおける名称を掲載。

❖ 講習会用PGC2003

- 在り処：「/home/schoolXX/DB/catalogue/pgc2003.dat」。
- データベース化のため以下の加工を施している。（これが一番大変な作業）
  - 赤経を時角から度に変換。
  - 赤緯を度分秒から度に変換。
  - 直径、楕円率、位置角が不明の場合の値「9.99」及び「999」を空文字に置換。
  - 区切り文字を','に変更。

❖講習会用PGC2003諸元

列名	単位	データ型	Null値	説明
id		TEXT	無	PGC番号（PGC0000001-PGC3099300）
ra	degree	DOUBLE PRECISION	無	赤経（J2000）
dc	degree	DOUBLE PRECISION	無	赤緯（J2000）
otype		TEXT	無	G: galaxy; M: multiple system; GM: galaxy in multiple system.
mtype		TEXT	有	LEDA（Lyon-Meudon Extragalactic Database）における暫定的な形態分類
logd25	0.1arcmin	REAL	有	天体の直径
e_logd25	0.1arcmin	REAL	有	天体の直径の誤差
logr25		REAL	有	天体の楕円率
e_logr25		REAL	有	天体の楕円率の誤差
pa	degree	REAL	有	天体の位置角
e_pa	degree	REAL	有	天体の位置角の誤差
o_anames		SMALLINT	無	他のカタログにおける天体の別名の数
anames		TEXT	有	他のカタログにおける天体の別名

- 「dec」はデータ型「decimal」の略なので使わない。

6.4.4. テーブルの作成

❖CREATE TABLE文の書式

```
CREATE TABLE <テーブル名> (
    <列名1>    <データ型>    [NOT NULL],
    <列名2>    <データ型>    [NOT NULL],
     ⋮
    CONSTRAINT <制約名> PRIMARY KEY (<列名>)
);
```

- **列名に大文字と「+*/-.':」は使わない**
  - 大文字は小文字として解釈されてしまう。
  - 列名を二重引用符で囲めば大文字と記号を使えるが、検索時も二重引用符をつけて列名を指定しなければならない。
- **主キー制約（PRIMARY KEY）**
  - テーブル内の各行を一意に識別できる列に対し設定する制約。
  - 重複データの入力を拒否。
  - 1テーブルに1つしか設定できない。
  - 主キー制約を課した列にはインデックスと非NULL制約が設定される。
- **非NULL制約（NOT NULL）**
  - NULL値が存在するデータの入力を拒否。

❖【実習】テーブル「pgc2003」の作成

■ SQLを記述したファイルを作成しファイルをPostgreSQLに読み込ませる

1. SQLを記述したファイルを「/home/schoolXX/DB/sql」以下に作成。

```
[schoolXX]$ cd /home/shoolXX/DB/sql
[schoolXX]$ emacs CreateTablePgc2003.sql
```

2. テーブル作成用のSQLを記述。

```
CREATE TABLE pgc2003 (
    id          TEXT                NOT NULL,
    ra          DOUBLE PRECISION    NOT NULL,
    dc          DOUBLE PRECISION    NOT NULL,
    otype       TEXT                NOT NULL,
    mtype       TEXT,
    logd25      REAL,
    e_logd25    REAL,
    logr25      REAL,
    e_logr25    REAL,
    pa          REAL,
    e_pa        REAL,
    o_anames    SMALLINT            NOT NULL,
    anames      TEXT,
    CONSTRAINT pgc2003_pkey PRIMARY KEY (id)
);
```

3. 作成したSQLを実行。

```
[schoolXX]$ psql -U dbr cat -f CreateTablePgc2003.sql
```

- -f：ファイルに記述されたSQLを実行するオプション。

4. 作成したテーブルをpsqlメタコマンド「¥d」で確認。

```
[shoolXX]$ psql -U dbr cat
cat=> ¥d
cat=> ¥d pgc2003
cat=> ¥q
```

❖ テーブルの削除方法

```
cat=# DROP TABLE pgc2003;
```

- データベースを所有するロールでもテーブルを削除可能。

6.4.5. データの登録

❖¥copyコマンドの書式

```
¥copy <テーブル名> FROM <ファイル名> [DELIMITER ‘<区切り文字>’] [NULL AS ‘<NULL値を表す文字>’]
```

- ファイルの内容をテーブルへコピーするpsqlメタコマンド。
- デフォルトの区切り文字は「¥t」（タブスペース）。
- デフォルトのNULL値を表す文字は「¥N」。

❖【実習】 データの登録

1. ロール「dbr」でデータベース「cat」に接続。

```
[schoolXX]$ psql -U dbr cat
```

2. /home/schoolXX/DB/catalogue/pgc2003.dat」をコピー。

```
cat=> ¥copy pgc2003 FROM /home/schoolXX/DB/catalogue/pgc2003.dat DELIMITER ‘,’ NULL AS ‘’
```

- 983261行コピーされる。

3. テーブルの確認。

```
cat=> SELECT COUNT(*) FROM pgc2003;
cat=> SELECT * FROM pgc2003 LIMIT 10;
cat=> ¥q
```

❖ データの削除方法

```
cat=# DELETE FROM pgc2003;
```

- データベースを所有するロールでもデータを削除可能。

❖【コラム】「¥copy」と「COPY」

- **¥copy**
  - psqlメタコマンド。
  - クライアント側にあるファイルをデータベースにコピー。
- **COPY**
  - SQLコマンド。
  - サーバ側にあるファイルをデータベースにコピー。
  - サーバ側のファイルはLinuxユーザ「postgres」が読み込み可能である必要がある。
  - ファイルは絶対パスで指定しなければならない。

6.5. インデックスの作成

❖テーブル「pgc2003」の列にインデックスを設定

❖6.5節目次

- 6.5.1. インデックスの仕組み
- 6.5.2. インデックスの作成と検索、速度の比較

6.5.1. インデックスの仕組み

❖インデックス（索引）

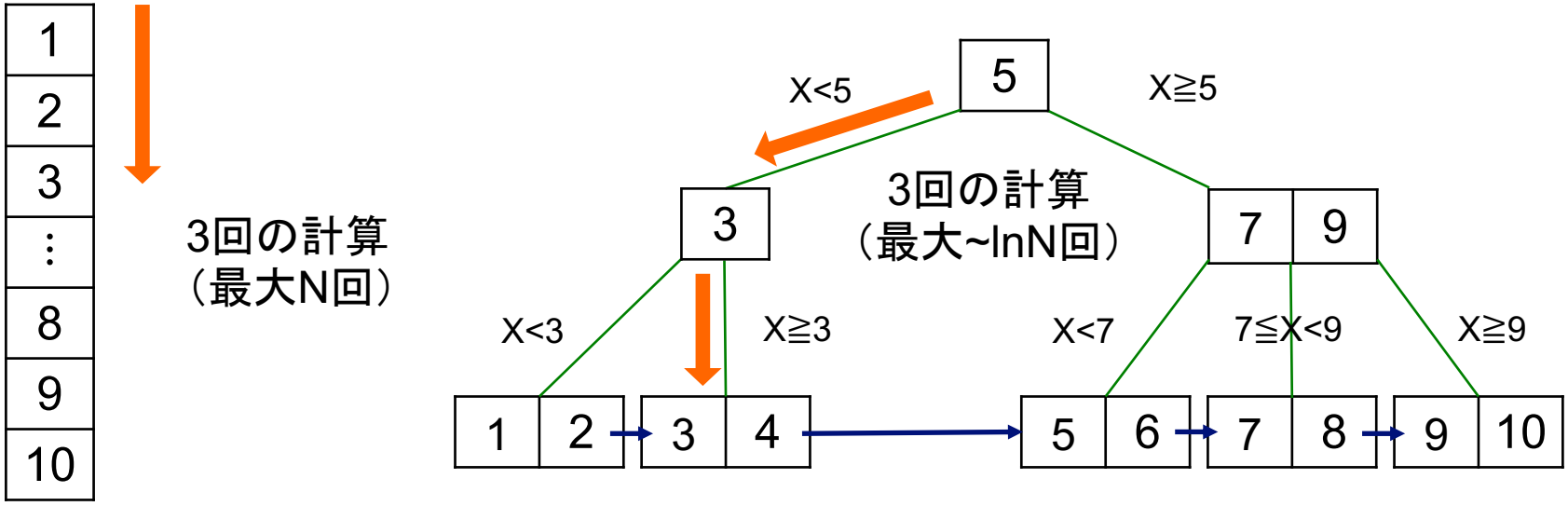
- 目的のデータまでの検索手順を最小化するためのデータ構造とその仕組み。
- インデックスを作成することでRDBMSは本来の性能を発揮し、インデックスがない場合と比べ格段に高速の検索を実現。
- 主キーを設定した列には自動的にインデックスが作成されるが、その他の列は必要に応じて作成する必要がある。
- PostgreSQLではデフォルトでB+TREEインデックスという方式が使用される。

❖B+TREEインデックス

- 平衡探索木型のデータ構造を持つインデックス。
- ある順番でソート可能なデータに対する等価性や範囲の問い合わせを扱うことができる。

❖例：1から10までのデータから3を探す場合。

シーケンシャルスキャン　　B+TREEインデックススキャン



6.5.2. インデックスの作成と検索、速度の比較

❖CREATE INDEX文の書式

```
CREATE INDEX [<インデックス名>] ON <テーブル名(列名)>;
```

- インデックス名を省略すると自動的に名前が設定される。

❖【実習】インデックスがない時の検索時間の測定

■ **テーブル「pgc2003」から赤緯-5度から+5度の天体を検索してその検索時間を測定**

1. ロール「dbr」でデータベース「cat」に接続。

```
[schoolXX]$ psql -U dbr cat
```

2. psqlメタコマンド「¥timing」を実行し、実行時間が計測されるようにする。

```
cat=> ¥timing
タイミングは on です。
```

3. 以下のSQLを3回以上実行し、実行時間の平均値を調べる。

```
cat=> SELECT count(*)
      FROM pgc2003
      WHERE dc BETWEEN -5 and 5;
```

❖【実習】インデックスの設定

■ テーブル「pgc2003」の列「ra」と「dc」にインデックスを設定

1. インデックスを作成。

```
cat=> CREATE INDEX ON pgc2003(ra);
cat=> CREATE INDEX ON pgc2003(dc);
```

2. インデックスの確認。

```
cat=> ¥d pgc2003
```

3. 以下のSQLを3回以上実行し、実行時間の平均値を調べる。

```
cat=> SELECT count(*)
      FROM pgc2003
      WHERE dc BETWEEN -5 and 5;
cat=> ¥q
```

❖【実習】インデックスの追加

■ テーブル「pgc2003」の残りのカラムにインデックスを設定

1. インデックスを作成。

```
cat=> CREATE INDEX ON pgc2003(otype);
cat=> CREATE INDEX ON pgc2003(mtype);
cat=> CREATE INDEX ON pgc2003(logd25);
cat=> CREATE INDEX ON pgc2003(e_logd25);
cat=> CREATE INDEX ON pgc2003(logr25);
cat=> CREATE INDEX ON pgc2003(e_logr25);
cat=> CREATE INDEX ON pgc2003(pa);
cat=> CREATE INDEX ON pgc2003(e_pa);
cat=> CREATE INDEX ON pgc2003(o_anames);
cat=> CREATE INDEX ON pgc2003(anames);
```

- インデックスを設定するとデータの更新処理が遅くなるため、本来は検索に使用する列のみに設定するのが正しい。

6.6. 演習問題：NVSSカタログのデータベース化

❖ The NRAO VLA Sky Survey (Condon+1998)

- Very Large Array（VLA）を使って、赤緯-40度以北の天域を周波数1.4GHzでサーベイした結果をまとめたカタログ。
- 2.5 mJy 以上の電波源を約177万の収録。
- 各電波源の座標、フラックス密度、長軸長、短軸長、位置角、偏波強度等を掲載。

❖ 講習会用NVSSカタログ

- 在り処：/home/schoolXX/DB/catalogue/nvss.dat
- データベース化のため、以下の加工を行っている。
  - 赤経を時角から度に変換。
  - 赤緯を度分秒角から度に変換。
  - Field列、Xpos列、Ypos列の削除。
    - サーベイデータのイメージ番号と、天体のXYピクセル座標。
  - 区切り文字を’,’に変更。

❖講習会用NVSSカタログ諸元

列名	単位	データ型	Null値	説明
id		TEXT	無	NVSSカタログにおける電波源名（HHMMSS+/-DDMMSS[a]）。
ra	degree	DOUBLE PRECISION	無	赤経（J2000）。
dc	degree	DOUBLE PRECISION	無	赤緯（J2000）。
e_ra	degree	DOUBLE PRECISION	無	赤経の誤差。
e_dc	degree	DOUBLE PRECISION	無	赤緯の誤差。
flux	mJy	DOUBLE PRECISION	無	電波源の1.4GHz帯の積分フラックス密度（最大値：858423.0）。
e_flux	mJy	DOUBLE PRECISION	無	電波源の1.4GHz帯の積分フラックス密度の誤差（最大値：32372.3）。
l_majaxis		TEXT	有	電波源の長軸長が上限値であることを示す記号”<“。
majaxis	arcsec	REAL	無	電波源の長軸長。
l_minaxis		TEXT	有	電波源の短軸長が上限値で有ることを示す記号”<“。
minaxis	arcsec	REAL	無	電波源の短軸長。
pa	arcdeg	REAL	有	電波源の長軸の位置角。
e_majaxis	arcsec	REAL	有	電波源の長軸長の誤差。
e_minaxis	arcsec	REAL	有	電波源の短軸長の誤差。
e_pa	arcdeg	REAL	有	電波源の長軸の位置角の誤差。
f_resflux		TEXT	有	電波源のフィッティング時の残差が大きいことを[PS*]で示す。ピークフラックス密度の残差が大きい時”S”、積分フラックス密度の残差が大きい時”P”を表示。
resflux	mJy/beam	REAL	有	電波源のフィッティング時の輝度の残差のピーク値（最大値：9999）。
polflux	mJy	REAL	有	電波源の1.4GHzでの（面積）積分直線偏波フラックス密度（最大値：999.9）。
polpa	degree	REAL	有	偏波角（-90度から90度）。
e_polflux	mJy	REAL	有	電波源の1.4GHz帯の積分直線偏波フラックス密度の誤差。
e_polpa	degree	REAL	有	偏波角の誤差。

❖【演習】NVSSカタログのデータベース化

1. **テーブル「nvss」を作成せよ。**
   - 参：P66。
   - ファイル「~/DB/sql/CreateTableNvss.sql」を作成。
2. **テーブル「nvss」にNVSSカタログをコピーせよ。**
   - 参：P68。
3. **テーブル「nvss」の全列にインデックスを設定せよ。**
   - 参：P75。

❖ 【解答例】テーブル「nvss」の作成とデータの登録

1. テーブル「nvss」を作成せよ。

```
[schoolXX]$ emacs /home/schoolXX/DB/sql/CreateTableNvss.sq
l
CREATE TABLE nvss (
    id         TEXT                NOT NULL,
    ra         DOUBLE PRECISION    NOT NULL,
    dc         DOUBLE PRECISION    NOT NULL,
    e_ra       DOUBLE PRECISION    NOT NULL,
    e_dc       DOUBLE PRECISION    NOT NULL,
    flux       DOUBLE PRECISION    NOT NULL,
    e_flux     DOUBLE PRECISION    NOT NULL,
    l_majaxis TEXT,
    majaxis    REAL                NOT NULL,
    l_minaxis TEXT,
    minaxis    REAL                NOT NULL,
    pa         REAL,
    e_majaxis REAL,
    e_minaxis REAL,
    e_pa       REAL,
    f_resflux TEXT,
    resflux    REAL,
    polflux    REAL,
    polpa      REAL,
    e_polflux REAL,
    e_polpa    REAL,
    CONSTRAINT nvss_pkey PRIMARY KEY (id)
);
```

2. テーブル「nvss」にNVSSカタログをコピーせよ。

```
cat=> ¥copy nvss FROM /home/schoolXX/DB/catalogue/nvss.d
at DELIMITER ‘,’ NULL AS ‘’
```

- 1773484行コピーされる。

3. テーブル「nvss」の全列にインデックスを設定せよ。

```
cat=> CREATE INDEX ON nvss(hoge);
```

- psqlメタコマンド「¥d nvss」でインデックスを作成できたか確認する。

❖【コラム】データ操作コマンド

- **PostgreSQLには3つのデータ操作コマンドが存在**
- **¥copy (COPY)**
  - ファイルからデータをテーブルにコピーするコマンド。
  - 大量のデータを高速にコピーできる。
  - ファイルからしかコピーできない。
- **INSERT**
  - テーブルに行を挿入するコマンド。
  - 値を明示して挿入、或いは他テーブルの値をSQLで取得して挿入可能。
  - 1行ずつ処理を行うため遅い。
- **UPDATE**
  - テーブルのデータを更新（変更）するためのコマンド。

7. psqlの使用方法

7.1. 概要

❖psql

- PostgreSQLの対話型インターフェース。
- 入力されたSQLをPostgreSQLサーバに送信し、その実行結果を受け取り表示。

❖7章目次

- 7.2. Linuxコマンド「psql」
- 7.3. SQL
- 7.4. psqlメタコマンド
- 7.5. ログインパスワードの入力省略設定

7.2. Linuxコマンド「psql」

❖psqlコマンドの書式

```
$ psql [<オプション>] -U <ロール名> <データベース名>
```

- **接続関連オプション**
  - -h <ホスト名|IPアドレス>：接続先サーバを指定。
  - -p <ポート番号>：PostgreSQLサーバのポート番号を指定。
- **問い合わせ結果の出力形式変更オプション**
  - -t：列名や結果行数の出力を抑制。
  - -A：位置揃えなしの出力モードに切り替え。
  - -F 区切り文字：フィールドの区切り文字を指定。「-A -F,」でCSV形式。
  - -H：HTML形式で出力。
  - -P format=latex：LaTeX形式で出力。
- **コマンドラインからのSQL実行時に使用するオプション**
  - -c “<SQL>”：指定したSQLを実行。
  - -f “<ファイル名>”：ファイルに記述されたSQLを実行。
  - -o “<ファイル名>”：SQLの実行結果をファイルに出力。
  - -L “<ファイル名>”：SQLの実行結果をファイルと標準出力の両方に出力。

❖実行例

■ ホスト名とポート名を明示してデータベースに接続

```
[schoolXX]$ psql -h localhost -p 5432 -U dbr cat
cat=> ¥q
```

- 5432：PostgreSQLサーバのデフォルトのポート番号。

■ コマンドラインからSQLを実行し結果をCSV形式でファイルに出力

```
[schoolXX]$ psql -t -A -F, -U dbr cat
            -o ~/DB/psql_result.txt
            -c “SELECT * FROM pgc2003 LIMIT 10;”
[schoolXX]$ cat ~/DB/psql_result.txt
```

7.3. SQL

❖SQLの書式

- セミコロン「;」がコマンドの終端。
- 文字列を除き大文字と小文字は区別されない。
- 文字列は単一引用符「'」で囲んで入力。
- 数値計算が可能。
- 単一行のコメントアウトは二重ダッシュ記号「--」。
- 複数行のコメントアウトは「/* コメント */」。

```
cat=> SELECT 'Hello world!'; --文字列入力の例
   ?column?
--------------
 Hello world!

cat=> SELECT 1+2*(3+4); /*数値計算の例*/
   ?column?
-------------
          15
```

7.4. psqlメタコマンド

❖psqlメタコマンド

- psql自身が実行するコマンド。
- バックスラッシュから始まり、改行がコマンドの終端。
- ここではよく使われるメタコマンドを紹介。

❖バッチファイル実行用メタコマンド

メタコマンド	説明
¥i “<ファイル名>”	ファイルに記述されたSQLを実行。
¥g “<ファイル名>”	「SQL ¥g ”<ファイル名>”」でSQLの実行結果をファイルに出力。
¥o “<ファイル名>”	以降のSQLの実行結果をファイルに出力。「/o」で書き込みを終了。

❖情報表示用メタコマンド

メタコマンド	説明
¥db	テーブル空間の一覧を表示。
¥du	データベースロールの一覧を表示。
¥l	データベースの一覧を表示。
¥d	テーブルとビューの一覧を表示。
¥d テーブル名	テーブルの情報を表示。
¥dS	全テーブルとビューの一覧を表示。
¥di	インデックスの一覧を表示。
¥df	ユーザが作成した関数の一覧を表示。
¥dfS	全関数の一覧を表示。
¥dT	ユーザが作成したデータ型の一覧を表示。
¥dTS	全データ型の一覧を表示

- 「¥db+」等、“+”をつけると詳細な情報を表示可能。

❖表示変更用メタコマンド

メタコマンド	説明
¥x	拡張形式モードへ切り替え。
¥t	結果のみ表示へ切り替え。
¥a	位置揃えなしの出力モードへ切り替え。
¥f ‘<区切り文字>’	出力区切り文字を設定。¥aを設定しないと無効。
¥H	HTML形式の出力モードへ切り替え。
¥pset format ‘latex’	LaTeX形式の出力モードへ切り替え。

❖その他のメタコマンド

メタコマンド	説明
¥h	ヘルプが存在するSQLコマンドの一覧を表示。
¥h <SQLコマンド>	SQLコマンドの説明を表示。
¥s	入力履歴を表示。
¥timing	SQLの実行時間を計測。
¥cd ‘<ディレクトリ名>’	現在の作業ディレクトリを変更。
¥! <LINUXコマンド>	LINUXコマンドを実行。例：¥! ls

7.5. ログインパスワードの入力省略設定

❖PostgreSQLサーバ接続時に求められるパスワードの入力省略設定を実施

- データベースへの接続情報を記述したファイルをホームディレクトリに配置。
- 入力省略設定はクライアント側の環境でおこなう。
  - すなわちパスワード認証が有効な外部のデータベースに接続する場合でもパスワード入力を省略できる。

❖「.pgpass」の書式

```
<ホスト名>:<ポート番号>:<データベース名>:<ロール名>:<パスワード>
```

- アクセス権は「600」（所有者のみ読み書き可）。

❖【実習】 パスワードの省略設定

1. ホームディレクトリに「.pgpass」を作成。

```
[schoolXX]$ cd
[schoolXX]$ emacs .pgpass
localhost:5432:cat:dbr:dbr
```

2. アクセス権の変更。

```
[schooolXX]$ chmod 600 .pgpass
```

3. 接続の確認。

```
[schoolXX]$ psql -U dbr cat
cat=> ¥q
```

8. テーブルへの問い合わせ

8.1. 概要

❖テーブルへの問い合わせコマンドであるSELECT文の使用方法を学習

❖8章目次

- 8.2. SELECT文の基本形
- 8.3. 検索条件の指定
- 8.4. 列に対する演算
- 8.5. 副問い合わせ
- 8.6. テーブルの結合
- 8.7. 遅くならないWHERE句の書き方

8.2. SELECT文の基本形

❖SELECT文

- テーブルから行を検索し、結果を表示するSQLコマンド。
- SELECT文には様々な「句」が存在し、「句」を組み合わせることで様々な検索実現。

❖8.2節目次

8.2.1. 基本形

❖SELECT文の書式

```
SELECT <列名1,列名2,...,列名N]|*>
FROM <テーブル名>;
```

❖実行例

■ テーブル「pgc2003」から列「id, ra, dc」を検索

```
cat=> SELECT id,ra,dc
      FROM pgc2003;
```

■ テーブル「pgc2003」から全列を検索

```
cat=> SELECT *
      FROM pgc2003;
```

- **検索結果画面の操作**
  - 1行送り：「Enter」。
  - ページ単位送り：「Space」。
  - 検索結果の表示終了：「q」。

8.2.2. 検索結果の出力件数の指定

❖LIMIT句の書式

```
SELECT <列名>
FROM <テーブル名>
LIMIT <出力件数>;
```

- LIMIT句の位置はSELECT文の文末。

❖実行例

■ テーブルの先頭から20行を出力

```
cat=> SELECT id,ra,dc
      FROM pgc2003
      LIMIT 20;
```

8.2.3. 検索結果の並び替え

❖ORDER BY句の書式

```
SELECT <列名>
FROM <テーブル名>
ORDER BY <列名> [ASC|DESC];
```

- ASC：昇順で並び替え（デフォルト）。
- DESC：降順で並び替え。

❖実行例

■ **検索結果を列「id」が小さい順に並び替えて出力**

```
cat=> SELECT id,ra,dc
      FROM pgc2003
      ORDER BY id
      LIMIT 20;
```

8.2.4. 列名・テーブル名の別名の定義

❖別名の定義方法

```
SELECT <列名> AS <別名>
FROM <テーブル名> AS <別名>;
```

- 列の別名はORDER BY句とGROUP BY句でのみ参照可能。

❖実行例

■ テーブル「pgc2003」と列「id, ra, dc」に別名を定義

```
cat=> SELECT p.id AS pgc_number,
             p.ra AS sekkei,
             p.dc AS sekii
      FROM pgc2003 AS p
      LIMIT 20;
```

- 「テーブル名.列名」は列名の厳密な指定方法。
- テーブルに別名を定義した場合は、列名を厳密に指定しなければならない場合が多い。

8.2.5. 演習問題

❖ 【演習】 以下の問い合わせを実施せよ

1. 【s825ex1.sql】テーブル「nvss」の列「id, ra, dc, flux」を、列「flux」が大きい順に20件表示せよ。

❖【解答例】以下の問い合わせを実施せよ

1. 【s825ex1.sql】テーブル「nvss」の列「id, ra, dc ,flux」を、列「flux」が大きい順に20件表示せよ。

```
cat=> SELECT id,ra,dc,flux
      FROM nvss
      ORDER BY flux DESC
      LIMIT 20;
```

8.3. 検索条件の指定

❖ WHERE句

- 指定した条件式を満たす行のみを返すことができる句。
- WHERE句で使われる演算子を紹介。

```
SELECT <列名>
FROM <テーブル名>
WHERE <条件式>;
```

❖ 8.3節目次

8.3.1. 比較演算子

❖比較演算子一覧

演算子	説明
<	小なり
>	大なり
<=	以下
>=	以上
=	等しい
<> or !=	等しくない（「!=」は内部で「<>」として処理される）

❖実行例

■ 列「id」がPGC0077777の天体を検索

```
cat=> SELECT id,ra,dc
      FROM pgc2003
      WHERE id='PGC0077777';
```

- 文字列はシングルクォーテーションで囲む。

8.3.2. IS演算子

❖IS演算子の書式

```
SELECT <列名>
FROM <テーブル名>
WHERE <列名> IS [NOT] NULL;
```

- NULL値を含む/含まない行を検索。
- 比較演算子でNULL値の比較演算をおこなうと戻り値として「NULL」を返すためNULL値の検索ができない。

❖実行例

■ 列「mtype」がNULLである行を比較演算子「=」を使って検索

```
cat=> SELECT id,ra,dc,mtype
      FROM pgc2003
      WHERE mtype=NULL
      LIMIT 20;
```

■ 列「mtype」がNULLである行をIS演算子を使って検索。

```
cat=> SELECT id,ra,dc,mtype
      FROM pgc2003
      WHERE mtype IS NULL
      LIMIT 20;
```

8.3.3. 論理演算子

❖論理演算子一覧

演算子	説明
AND	論理積
OR	論理和
NOT	否定

❖実行例

- ■ 赤経が359度から1度かつ列「mtype」がNULLではない天体を検索

```
cat=> SELECT id,ra,dc,mtype
      FROM pgc2003
      WHERE (ra>359 OR ra<1) AND
            (mtype IS NOT NULL)
      ORDER BY ra;
```

8.3.4. BETWEEN句

❖BETWEEN句の書式

```
SELECT <列名>
FROM <テーブル名>
WHERE <列名> [NOT] BETWEEN <a> AND <b>;
```

- 列の値がa以上b以下の行を検索。

❖実行例

■ **赤緯が-0.1度以上+0.1度以下の天体を検索**

```
cat=> SELECT id,ra,dc
      FROM pgc2003
      WHERE dc BETWEEN -0.1 AND 0.1
      ORDER BY dc;
```

■ **赤経が359度から1度かつ列「mtype」がNULLではない天体を検索**

```
cat=> SELECT id,ra,dc,mtype
      FROM pgc2003
      WHERE (ra NOT BETWEEN 1 AND 359) AND
            (mtype IS NOT NULL)
      ORDER BY ra;
```

8.3.5. LIKE/SIMILAR TO演算子

❖LIKE/SIMILAR TO演算子の書式

```
SELECT <列名>
FROM <テーブル名>
WHERE <列名> [NOT] LIKE|SIMILAR TO '<パターン>';
```

❖使用可能な正規表現

演算子	説明	LIKE	SIMILAR TO
%	0文字以上の文字に一致。	○	○
_	任意の1文字に一致。	○	○
*	直前の文字の0回以上の繰り返し。	×	○
+	直前の文字の1回以上の繰り返す。	×	○
[文字列]	指定した文字列、文字範囲に含まれるあらゆる文字に一致。	×	○
[^文字列]	指定した文字列、文字範囲に含まれないあらゆる文字に一致。	×	○
|	二者択一。	×	○
(パターン)	パターンのグループ化。	×	○

- SIMILAR TOはPOSIX準拠の正規表現を使用できるが、正規表現に対する脆弱性攻撃に気を付ける必要がある。

❖実行例

■ LIKE演算子を使って「NGC5194」を検索

```
cat=> SELECT id,ra,dc,anames
      FROM pgc2003
      WHERE anames LIKE '%NGC5194%';
```

- 列「anames」にはスペース区切りで複数の別名が羅列されているため、「%」で「NGC5194」前後の文字列を表現。

■ SIMILAR TO演算子を使って「NGC1~9」を検索

```
cat=> SELECT id,ra,dc,anames
      FROM pgc2003
      WHERE anames SIMILAR TO '%NGC[1-9]_%';
```

- _：空白。

8.3.6. 演習問題

- 【演習】 以下の問い合わせを実施せよ
  1. 【s836ex1.sql】テーブル「pgc2003」から赤経が（202<=ra<=204）かつ赤緯が（46<=dc<=48）である天体を検索し、列「id,ra,dc,anames」を「anames」がアルファベット順となるように表示せよ。
  2. 【s836ex2.sql】上記の問い合わせに列「anames」がNULLでないという条件を追加し検索せよ。

❖【解答例】 以下の問い合わせを実施せよ

1. 【s836ex1.sql】テーブル「pgc2003」から赤経が（202<=ra<=204）かつ赤緯が（46<=dc<=48）である天体を検索し、列「id,ra,dc,anames」を「anames」がアルファベット順となるように表示せよ。

```
cat=> SELECT id,ra,dc,anames
      FROM pgc2003
      WHERE (ra BETWEEN 202 AND 204) AND
            (dc BETWEEN 46 AND 48)
      ORDER BY anames;
```

2. 【s836ex2.sql】上記の問い合わせに「anames」がNULLでないという条件を追加し検索せよ。

```
cat=> SELECT id,ra,dc,anames
      FROM pgc2003
      WHERE (ra BETWEEN 202 AND 204) AND
            (dc BETWEEN 46 AND 48) AND
            (anames IS NOT NULL)
      ORDER BY anames;
```

8.4. 列に対する演算

❖演算子と関数

- PostgreSQLには様々な演算子と関数が用意されている。
- SELECT文中で列に対し演算を行うことで、演算結果を検索結果として得ることができる。

❖8.4節目次

8.4.1. 型変換関数

❖型変換関数一覧

関数	説明	例	結果
CAST(データ AS データ型)	データを指定したデータ型に変換。	CAST(1.2 AS INTEGER)	1
データ :: データ型	PostgreSQL独自の記法(演算子?)。	1.2 :: INTEGER	1

❖実行例

■ 異なるデータ型の比較演算

```
cat=> SELECT id,polflux
      FROM nvss
      WHERE polflux=CAST(999.9 AS REAL);
cat=> SELECT id,polflux
      FROM nvss
      WHERE polflux=999.9;
cat=> SELECT pg_typeof(999.9);
cat=> SELECT 999.9::REAL=999.9::NUMERIC AS logical;
```

- REAL型とNUMERIC型の999.9は異なる値。

❖実行例

■ 入力値のデータ型

```
cat=> SELECT 3, 3.0, 3^3, 3.0^3;
3 | 3.0 | 27 | 27.000000000000000

cat=> SELECT pg_typeof(3),
             pg_typeof(3.0),
             pg_typeof(3^3),
             pg_typeof(3.0^3);
integer | numeric | double precision | numeric
```

- 字句解析をおこない入力引数にデータ型を割り当て（数値はINTEGERかNUMERIC）。
- 演算子ごとに決められたデータ型に型変換して演算。
- 演算子「^」の引数型と戻り値型：
  - DOUBLE PRECISION^DOUBLE PRECISION=>DOUBLE PRECISION
  - NUMERIC^NUMERIC=>NUMERIC

8.4.2. 算術演算子

❖算術演算子一覧

演算子	説明	例	結果
+	和	1+2	3
-	差	1-2	-1
*	積	3*4	12
/	商（整数同士の割り算はの場合は余りを切り捨てる）	7/3	2
%	剰余	7%2	1
^	べき乗	2^3	8
|/	平方根	|/ 144	12
||/	立方根	||/ 27	3
!	階乗	3!	6
@	絶対値	@(-10)	10

❖実行例

■ 列「logd25」のlogを外してarcminに変換したものを値が大きい順に表示

```
cat=> SELECT id,(10^logd25)*0.1 AS d,anames
      FROM pgc2003
      WHERE logd25 IS NOT NULL
      ORDER BY d DESC
      LIMIT 20;
```

- 演算結果列に別名「d」を付与し、ORDER BY句で指定。
- 演算結果列は無名「?column?」なため、別名を付与しないとORDER BY句で列名を指定ができない。

8.4.3. 算術関数

❖算術関数一覧

関数	戻り値型	説明	例	結果
ABS(X)	Xと同じ	絶対値	ABS(-2.0)	2.0
CBRT(DP)	DP	立方根	CBRT(27.0)	3
CEIL(DP or NUMERIC)	入力型と同一	切り上げ	CEIL(-3.2)	-3
DEGREES(DP)	DP	ラジアンに対応する度	DEGREES(3.14)	179.908747671078
EXP(DP or NUMERIC)	入力型と同一	指数	EXP(1.0)	2.7182818284590452
FLOOR(DP or NUMERIC)	入力型と同一	切り下げ	FLOOR(-3.2)	-4
LN(DP or NUMERIC)	入力型と同一	自然対数	LN(2.7)	0.9932517730102834
LOG(DP or NUMERIC)	入力型と同一	常用対数	LOG(100)	2
LOG(B NUMERIC,X NUMERIC)	入力型と同一	Bを底としたXの対数	LOG(2,256)	8.0000000000000000
MOD(Y,X)	引数型と同一	Y/Xの剰余	MOD(9.3,3)	0.1
PI()	DP	円周率	PI()	3.14159265358979
POWER(A DP, B DP)	DP	AのB乗	POWER(3.0,2.0)	9.0000000000000000
POWER(A NUMERIC,B NUMERIC)	NUMERIC	AのB乗	POWER(3.0,2.0)	9
RADIANS(DP)	DP	度に対応するラジアン	RADIANS(180)	3.14159265358979
RANDOM()	DP	0.0-1.0の乱数値	RANDOM()	0.529124391730875
ROUND(DP or NUMERIC)	入力型と同一	四捨五入	ROUND(11.1)	11
ROUND(V NUMERIC,S INTEGER)	NUMERIC	Sの桁で四捨五入	ROUND(12.345,1)	12.3
SETSEED(DP)	INTEGER	RANDOM()で使用する種を設定	---	---
SIGN(DP or NUMERIC)	入力型と同一	引数の符号	SIGN(-2.4)	-1
SQRT(DP or NUMERIC)	入力型と同一	平方根	SQRT(3.0)	1.732050807568877
TRUNC(DP or NUMERIC)	入力型と同一	切り捨て	TRUNC(17.5)	17
TRUNC(V NUMERIC,S INTEGER)	NUMERIC	Vをsの桁で切り捨て	TRUNC(12.345,1)	12.3

 ※DP：DOUBLE PRECISION

❖三角関数一覧

関数	戻り値型	説明
ACOS(DP)	DOUBLE PRECISION	逆余弦関数
ASIN(DP)	DOUBLE PRECISION	逆正弦関数
ATAN(DP)	DOUBLE PRECISION	逆正接関数
ATAN2(Y DP,X DP)	DOUBLE PRECISION	Y/Xの逆正接関数
COS(DP)	DOUBLE PRECISION	余弦関数
COT(DP)	DOUBLE PRECISION	余接関数
SIN(DP)	DOUBLE PRECISION	正弦関数
TAN(DP)	DOUBLE PRECISION	正接関数

❖実行例

■ 赤経と赤緯の小数点以下を切り捨て

```
cat=> SELECT id,
             TRUNC(CAST(ra AS NUMERIC),0) AS tra,
             TRUNC(CAST(dc AS NUMERIC),0) AS tdc
      FROM pgc2003
      LIMIT 20;
```

- TRUNC(V NUMERIC,S INTEGER)：VをSの桁で切り捨て。
- 列「ra」と「dc」のデータ型はDOUBLE PRECISION型であるため、CAST関数でNUMERICに変更しないとエラーが発生する。

8.4.4. 集約・統計関数

❖統計関数一覧

関数	引数型	戻り値型	説明
CORR(Y,X)	DOUBLE PRECISION	DOUBLE PRECISION	相関係数
COVAR_POP(Y,X)	DOUBLE PRECISION	DOUBLE PRECISION	母共分散
COVAR_SAMP(Y,X)	DOUBLE PRECISION	DOUBLE PRECISION	標本共分散
REGR_AVGX(Y,X)	DOUBLE PRECISION	DOUBLE PRECISION	独立変数の平均値（SUM(X)/N)
REGR_AVGY(Y,X)	DOUBLE PRECISION	DOUBLE PRECISION	従属変数の平均値（SUM(Y)/N)
REGR_COUNT(Y,X)	DOUBLE PRECISION	BIGINT	両式が非NULLとなる入力行の個数
REGR_INTERCEPT(Y,X)	DOUBLE PRECISION	DOUBLE PRECISION	(X,Y)の組み合わせで決まる最小二乗法による線形方程式のY切片
REGR_R2(Y,X)	DOUBLE PRECISION	DOUBLE PRECISION	相関係数の二乗値
REGR_SLOPE(Y,X)	DOUBLE PRECISION	DOUBLE PRECISION	(X,Y)の組み合わせで決まる最小二乗法による線形方程式の傾き
REGR_SXX(Y,X)	DOUBLE PRECISION	DOUBLE PRECISION	SUM(X^2)-SUM(X)^2/N（従属変数の二乗和）
REGR_SXY(Y,X)	DOUBLE PRECISION	DOUBLE PRECISION	SUM(X*Y)-SUM(X)*SUM(Y)/N（従属変数と独立変数の積の和）
REGR_SYY(Y,X)	DOUBLE PRECISION	DOUBLE PRECISION	SUM(Y^2)-SUM(Y)^2/N（独立変数の二乗和）
STDDEV_POP(式)	SMALLINT、INT、BIGINT、REAL、DOUBLE PRECISION、NUMERIC	浮動小数点数型引数はDOUBLE PRECISION、その他はNUMERIC	入力値に対する母標準偏差
STDDEV_SAMP(式)			入力値に対する標本標準偏差
VAR_POP(式)			入力値に対する母分散
VAR_SAMP（式）			入力値に対する標本分散

❖集約関数一覧

関数	戻り値型	説明
AVG(X)	整数型引数はNUMERIC、 浮動小数点数型引数はDOUBLE PRECISION、 その他は入力型と同一	全ての入力値の平均値を返す
COUNT(*)	---	入力行の数
MAX(X)	入力型と同一	全ての入力値の中から最大値を返す
MIN(X)	入力型と同一	全ての入力値の中から最小値を返す
SUM(X)	SMALLINTとINT型引数はBIGINT、 BIGINT型引数はNUMERIC、 その他は入力型と同一	全ての入力値の和を返す

- 複数の入力値から1つの結果を返す。
- WHERE句では使用できない。

❖実行例

■ テーブルのレコード数を算出

```
cat=> SELECT COUNT(*) FROM pgc2003;
```

■ 列「logd25」の最大値を検索

```
cat=> SELECT MAX(logd25) FROM pgc2003;
```

■ 列「logd25」の最大値がどのIDなのか知りたいとき（間違い例）

```
cat=> SELECT id FROM pgc2003 WHERE logd25=MAX(logd25);
```

❖GROUP BY句とHAVING句の書式

```
SELECT <列名> FROM <テーブル名>
GROUP BY <列名>
[HAVING <条件式>];
```

- GROUP BY句：同じ値を持つ行をグループ化（して集約・統計）。
- HAVING句：グループの中から条件式を満たす行を返す。

❖実行例

■ 列「mtype」をグループ化してタイプが「SB%」の天体数と各グループ内で最大の列「logd25」の値を表示

```
cat=> SELECT mtype,COUNT(*),max(logd25)
      FROM pgc2003
      GROUP BY mtype
      HAVING mtype LIKE 'SB%';
```

■ 赤経10度ごとにグループ化して天体数を表示

```
cat=>  SELECT TRUNC(ra::NUMERIC,-1) AS tra,COUNT(*)
       FROM pgc2003
       WHERE mtype IS NOT NULL
       GROUP BY tra
       ORDER BY tra;
```

8.4.5. 文字列演算子・関数

❖文字列演算子・関数一覧

演算子・関数	説明	例	結果
||	文字列の結合	ho || ge	hoge
CHAR_LENGTH(‘text’)	文字数を返す	CHAR_LENGTH(’hoge’)	4
OVERLAY(‘text1’ PLACING ‘text2’ FROM i)	text1のi番目以降の文字をtext2で上書き	OVERLAY(‘hoge’ PLACING ‘pp’ FOR 2)	hppe
POSITION(‘text1’ IN ‘text2’)	text2からtext1の文字列が始まる位置を返す	POSITION(‘hoge’ IN ‘og’)	2
SUBSTRING(‘text’ FROM i FOR j)	textのi番目の文字からj文字分取り出す	SUBSTRING(‘hoge’ FROM 3 FOR 2)	ge
SUBSTRING(‘test’ FROM ‘pattern’)	正規表現にマッチする部分文字列を返す。	SUBSTRING(‘hoge’ FROM ‘..$’)	ge
TRIM(‘text1’ FROM ‘text2’)	text2からtext1の文字を取り除く	TRIM(‘he’ FROM ’hoge’)	og

❖実行例

■ 列「id」から「PGC」を削除

```
cat=> SELECT TRIM('PGC' FROM id)
      FROM pgc2003
      LIMIT 20;
```

8.4.6. 演習問題

❖ 【演習】 以下の問い合わせを実行せよ

1. 【s846ex1.sql】テーブル「nvss」から「flux」が1000mJy以上の電波源を検索しその総数を表示せよ。
2. 【s846ex2.sql】テーブル「nvss」を検索し、列「id,x,y,z」を表示せよ。
   - x = cos(ra) * cos(dc)
   - y = sin(ra) * cos(dc)
   - z = sin(dc)

❖【解答例】 以下の問い合わせを実施せよ

1. 【s846ex1.sql】テーブル「nvss」から「flux」が1000mJy以上の電波源を検索しその総数を表示せよ。

```
cat => SELECT COUNT(*)
       FROM nvss
       WHERE flux>=1000;
```

2. 【s846ex2.sql】テーブル「nvss」を検索し、列「id,x,y,z」を表示せよ。

```
cat => SELECT id,
              COS(RADIANS(ra))*COS(RADIANS(dc)) AS x,
              SIN(RADIANS(ra))*COS(RADIANS(dc)) AS y,
              SIN(RADIANS(dc)) AS z
       FROM nvss
       LIMIT 20;
```

❖【コラム】SELECT分の評価順序

- SELECT文は以下の順序で評価がおこなわれる。

1. WITH句
2. FROM句
3. WHERE句
4. GROUP BY句
5. HAVING句
6. SELECT句
7. UNION・INTERSECT・EXCEPT句
8. ORDER BY句
9. DISTINCT句
10. LIMIT句

❖例

```
cat => SELECT TRUNC(ra::NUMERIC,-1) AS tra,COUNT(*)
       FROM pgc2003
       WHERE mtype IS NOT NULL
       GROUP BY tra
       ORDER BY tra;
```

1. FROM：テーブル「pgc2003」から、
2. WHERE：「mtype」がNULL値の行を除外し、
3. GROUP BY：赤経が同じ値の行をグループ化し、
4. SELECT：グループ毎にCOUNT(*)を計算し、
5. ORDER BY：赤経の値が小さい順に並び替える。

8.5. 副問い合わせ

❖副問い合わせ

- SELECT文中でSELECT文を用いることで、より複雑な検索が可能になる。
- SELECT句中、FROM句中、WHERE句中でSELECT文を用いることができる。

❖8.5節目次

8.5.1. SELECT句での副問い合わせ

❖SELECT句での副問い合わせ書式

```
SELECT (
   SELECT <列名>
   FROM <テーブル名>
   WHERE <条件式>
) AS <別名>
FROM <テーブル名>;
```

- 副問い合わせが返した一行一列の値を問い合わせ結果に結合して表示。

❖実行例

■ 各電波源のFluxが平均値よりも大きいか検査

```
cat=> SELECT id,flux,flux > (
         SELECT AVG(flux)
         FROM nvss
       ) AS logical
       FROM nvss
       LIMIT 20;
```

8.5.2. FROM句での副問い合わせ

❖FROM句での副問い合わせ書式

```
SELECT <副問い合わせの別名.列名>
FROM (
   SELECT <列名>
   FROM <テーブル名>
   WHERE <条件式>
) AS <副問い合わせの別名>;
```

- 副問い合わせが返した複数行複数列の結果に対して問い合わせをおこなう。
- 副問い合わせの結果に別名を指定しなければならない。
- 主SELECT文ではインデックス検索ができない。

❖実行例

■ **列「mtype」が「SB%」の天体の中から赤緯が±0.1度内の天体を検索**

```
cat => SELECT t.id,t.ra,t.dc,t.mtype
       FROM (
          SELECT *
          FROM pgc2003
          WHERE mtype LIKE 'SB%'
       ) AS t
       WHERE t.dc BETWEEN -0.1 AND 0.1;
```

8.5.3. WHERE句での副問い合わせ

❖WHERE句での副問い合わせ

- 一行一列の値を返す副問い合わせ
- 複数行複数列の値を返す副問い合わせ
- 相関副問い合わせ

❖一行一列の値を返す副問い合わせ書式

```
SELECT <列名>
FROM <テーブル名>
WHERE <列名> 比較演算子 (
   SELECT <列名>
   FROM <テーブル名>
   WHERE <条件式>
);
```

- 副問い合わせが返した一行一列の値を比較条件として問い合わせをおこなう。

❖実行例

■ 列「logd25」の最大値がどのIDなのか知りたいとき

```
cat=> SELECT id,logd25
      FROM pgc2003
      WHERE logd25 = (
        SELECT MAX(logd25)
        FROM pgc2003
      );
```

■ Fluxが平均値よりも大きい電波源を検索

```
cat=> SELECT id,flux,(SELECT AVG(flux) FROM NVSS) AS a
      FROM nvss
      WHERE flux > (
        SELECT AVG(flux)
        FROM nvss
      )
      LIMIT 20;
```

❖複数行複数列の値を返す副問い合わせ書式

```
SELECT <列名>
FROM <テーブル名>
WHERE (<列名1>,<列名2>,...<列名n>) [NOT] IN (
  SELECT <列名1,>,<列名2>,...,<列名n>
  FROM <テーブル名>
  WHERE <条件式>
);
```

- 副問い合わせが返した複数行複数列の値がIN句で指定した列にあるか評価。

❖実行例

■ **テーブル「nvss」から赤経赤緯の値がテーブル「pgc2003」のそれと完全に一致する行を検索**

```
cat=> SELECT id,ra,dc
      FROM nvss
      WHERE (ra,dc) IN (
        SELECT ra,dc
        FROM pgc2003
      );
```

❖相関副問い合わせ書式

```
SELECT <列名>
FROM <テーブル名>
WHERE [NOT] EXISTS (
   SELECT <列名>
   FROM <テーブル名>
   WHERE <条件式>
);
```

- 副問い合わせの結果が真である場合のみ主SELECT文の結果を返す。
- 主SELECT文のテーブルを副問い合わせが参照する為「相関副問い合わせ」。

❖実行例

■ **テーブル「nvss」から赤経赤緯の値がテーブル「pgc2003」のそれと完全に一致するレコードを検索**

```
cat=> SELECT id,ra,dc
      FROM pgc2003
      WHERE EXISTS (
        SELECT *
        FROM nvss
        WHERE (pgc2003.ra=nvss.ra) AND
              (pgc2003.dc=nvss.dc)
      );
```

- pgc2003のレコードを一行ずつ副問い合わせで評価し、真の場合結果を返す。

8.5.4. 共通テーブル式

❖WITH句の書式

```
WITH <別名1> AS (
   SELECT <列名>
   FROM <テーブル名>
   WHERE <条件式>
) [,<別名N> AS (...)]
```

- WITH句に記述した問い合わせの結果を一時的にテーブル化。
- 共通テーブル式は主SELECT文内でのみ有効（他の問い合わせには無影響）。

❖実行例

■ 列「mtype」が「SB%」の天体の中から赤緯が±0.1度内の天体を検索

```
cat=> WITH cte AS (
         SELECT *
         FROM pgc2003
         WHERE mtype LIKE 'SB%'
       )
       SELECT id,ra,dc,mtype
       FROM cte
       WHERE dc BETWEEN -0.1 AND 0.1;
```

8.5.5. 演習問題

- 【演習】 以下の問い合わせを実行せよ。
  1. 【s855ex1.sql】副問い合わせを使ってテーブル「nvss」から赤経が（202<=ra<=204）かつ赤緯が（46<=dc<=48）である電波源を検索し、その中から列「flux」が100mJy以上の電波源を検索せよ。
  2. 【s855ex2.sql】副問い合わせを使ってテーブル「nvss」から列「flux」が最も大きい電波源の「id,ra,dc,flux,最も大きいflux」を表示せよ。

❖【解答例】 以下の問い合わせを実施せよ。

1. 【s855ex1.sql】FROM句での副問い合わせを使ってテーブル「nvss」から赤経が（202<=ra<=204）かつ赤緯が（46<=dc<=48）である電波源を検索し、その中から「flux」が100mJy以上の電波源を検索せよ。

```
cat => SELECT *
       FROM (
         SELECT *
         FROM nvss
         WHERE (ra BETWEEN 202 AND 204) AND
               (dc BETWEEN 46 AND 48)
       ) AS t
       WHERE t.flux>=100;
```

2. 【s855ex2.sql】副問い合わせを使ってテーブル「nvss」から列「flux」が最も大きい電波源の「id,ra,dc,flux,最も大きいflux」を表示せよ。

```
cat => SELECT id,ra,dc,flux,
              (SELECT MAX(flux) FROM nvss)
       FROM nvss
       WHERE flux = (SELECT MAX(flux) FROM nvss);
```

■ 共通テーブル式を使った場合

```
cat => WITH cte AS (
         SELECT MAX(flux)
         FROM nvss
       )
       SELECT id,ra,dc,flux,(SELECT * FROM cte)
       FROM nvss
       WHERE flux = (SELECT * FROM cte);
```

- 共通テーブル式はテーブルなので、FROM句以外の場所で使う場合はSELECT文で値を引き出さなければならない。

8.6. テーブルの結合

❖テーブルの結合

- テーブル同士を結合することで、複数のテーブルに対して同時に検索ができる。
- 結合条件を課すことでテーブル同士を関連づけられる。
- 交差結合、内部結合、外部結合という結合方法が存在。

❖8.6節目次

8.6.1. 交差結合

❖交差結合の書式1

```
SELECT <列名>
FROM <テーブル1> CROSS JOIN <テーブル2>;
```

- テーブル1の各行にテーブル2の全行を結合。全ての組み合わせが求まる。
- 内部・外部結合の結果は交差結合に結合条件を課したもの。

❖図解

■ テーブルAとBを交差結合した場合

A

ID	主食	種別
1	ごはん	和
2	パン	洋
3	炒飯	中

X

B

ID	おかず	種別
1	納豆	和
2	ハム	洋

=

ID	主食	種別	ID	おかず	種別
1	ごはん	和	1	納豆	和
1	ごはん	和	2	ハム	洋
2	パン	洋	1	納豆	和
2	パン	洋	2	ハム	洋
3	炒飯	中	1	納豆	和
3	炒飯	中	2	ハム	洋

❖実行例

■ テーブル「pgc2003」と「nvss」を交差結合し、赤経赤緯の値が完全に一致する行を検索

```
cat=> SELECT a.id,
             a.ra,
             a.dc,
             b.id,
             b.ra,
             b.dc
      FROM pgc2003 AS a CROSS JOIN nvss AS b
      WHERE a.ra=b.ra AND
            a.dc=b.dc
      ORDER BY a.id;
```

- 内部結合と同等の結果を得られる。

❖交差結合の書式2

```
SELECT <列>
FROM <テーブル1>,<テーブル2>,...,<テーブルn>
```

❖実行例

- テーブル「pgc2003」と「nvss」を交差結合し、赤経赤緯の値が完全に一致する行を検索

```
cat=> SELECT a.id,
             a.ra,
             a.dc,
             b.id,
             b.ra,
             b.dc
      FROM pgc2003 AS a,
           nvss    AS b
      WHERE a.ra=b.ra AND
            a.dc=b.dc
      ORDER BY a.id;
```

- 内部結合と同等の結果を得られる。
- よく使う。

8.6.2. 内部結合

❖内部結合の書式

```
SELECT <列名>
FROM <テーブル1> INNER JOIN <テーブル2> ON <結合条件>;
```

- テーブル1の各行に結合条件を満たすテーブル2の行を結合。
- 結合条件は論理値が求まる式（テーブル1.id=テーブル2.idなど）。

❖図解

■ テーブルAとBを「A.種別=B.種別」という条件で内部結合した場合

A

ID	主食	種別
1	ごはん	和
2	パン	洋
3	炒飯	中

X

B

ID	おかず	種別
1	納豆	和
2	ハム	洋

=

ID	主食	種別	ID	おかず	種別
1	ごはん	和	1	納豆	和
2	パン	洋	2	ハム	洋

❖実行例

- テーブル「pgc2003」と「nvss」を内部結合し、赤経赤緯の値が完全に一致する行を検索

```
cat=> SELECT a.id,
             a.ra,
             a.dc,
             b.id,
             b.ra,
             b.dc
      FROM pgc2003 AS a INNER JOIN nvss AS b
           ON a.ra=b.ra AND
              a.dc=b.dc
      ORDER BY a.id;
```

❖複数のテーブルを結合する場合の例

```
SELECT t1.id,
       t2.id,
       t3.id
FROM (t1 INNER JOIN t2 ON t1.id = t2.id)
     INNER JOIN ON t1.id = t3.id;
```

8.6.3. 外部結合

❖外部結合の書式

```
SELECT <列名>
FROM <テーブル1> LEFT|RIGHT|FULL OUTER JOIN <テーブル2>
ON <結合条件>;
```

- 結合条件に真偽に拘らず結合。偽の場合NULL値が入力された行を結合。
- 左外部結合、右外部結合、完全外部結合が存在。

143

❖図解（左外部結合）

■ テーブルAとBを「A.種別=B.種別」という条件で左外部結合した場合

A

ID	主食	種別
1	ごはん	和
2	パン	洋
3	炒飯	中

X

B

ID	おかず	種別
1	納豆	和
2	ハム	洋

=

ID	主食	種別	ID	おかず	種別
1	ごはん	和	1	納豆	和
2	パン	洋	2	ハム	洋
3	炒飯	中			

- JOIN句の左側のテーブルを基準に右側のテーブルを結合。

❖図解（右外部結合）

■ テーブルAとBを「A.種別=B.種別」という条件で右外部結合した場合

A

ID	主食	種別
1	ごはん	和
2	パン	洋
3	炒飯	中

X

B

ID	おかず	種別
1	納豆	和
2	ハム	洋

=

ID	主食	種別	ID	おかず	種別
1	ごはん	和	1	納豆	和
2	パン	洋	2	ハム	洋

- JOIN句の右側のテーブルを基準に左側のテーブルを結合。

❖図解（完全外部結合）

– テーブルAとBを「A.種別=B.種別」という条件で完全外部結合した場合

A

ID	主食	種別
1	ごはん	和
2	炒飯	中

X

B

ID	おかず	種別
1	納豆	和
2	ハム	洋

=

ID	主食	種別	ID	おかず	種別
1	ごはん	和	1	納豆	和
2	炒飯	中			
			2	ハム	洋

- 両テーブルの行を全て保持して結合。

❖実行例

■ テーブル「pgc2003」から赤緯が（10.0<=dc<=10.1）の天体を抽出したものとテーブル「nvss」を左外部結合し、赤経赤緯の値が完全に一致する行を検索

```
cat=> WITH a AS (
        SELECT *
        FROM pgc2003
        WHERE dc BETWEEN 10.0 AND 10.1
      )
      SELECT a.id,
             a.ra,
             a.dc,
             b.id,
             b.ra,
             b.dc
      FROM a LEFT OUTER JOIN nvss AS b
           ON a.ra=b.ra AND
              a.dc=b.dc
      ORDER BY a.id;
```

- 一方（あるいは両方）のテーブルの行を残したい場合に使う。
- 例えば日時を主キーとした温度、湿度、気圧テーブルを日時を結合条件として結合した場合、内部結合ではあるテーブルである日時にデータ欠損が発生するとその行が結合条件を満たさない為表示されなくなる。

8.6.4. 演習問題

- 【演習】以下の問い合わせを実行せよ。
  1. 【s864ex1.sql】テーブル「pgc2003」と「nvss」を赤経赤緯の小数第三位までが一致するという条件で内部結合し、赤緯が（-1<=dc<=1）である天体の数を調べよ。

❖【解答例】 以下の問い合わせを実施せよ。

1. テーブル「pgc2003」と「nvss」を赤経赤緯の小数第三位までが一致するという条件で内部結合し、赤緯が（-1<=dc<=1）である天体の数を調べよ。

```
cat => SELECT COUNT(*)
       FROM pgc2003 AS a INNER JOIN nvss AS b
       ON
          (
            TRUNC(CAST(a.ra AS numeric),4)=
            TRUNC(CAST(b.ra AS numeric),4)
          )AND(
            TRUNC(CAST(a.dc AS numeric),4)=
            TRUNC(CAST(b.dc AS numeric),4)
          )
       WHERE a.dc BETWEEN -1 AND 1;
```

- なぜ検索に数秒かかるのか？

8.7. 遅くならないSELECT文の書き方

❖RDBMSは問い合わせの結果が直ちに返ってくるように利用すべき

- 以下の場合、結果がすぐに返ってこない。
  - WHERE句・ON句の書き方が悪い。
  - インデックスを張っていない列を検索している。
  - 検索結果が膨大で、ディスクI/Oがボトルネックに。
- 質の悪いSQLは作業効率の低下を招くだけでなく、サーバに無用な負担をかけてしまう。

❖8.7節目次

8.7.1. 問い合わせの実行計画の表示

❖EXPLAIN句の書式

```
EXPLAIN [ANALYZE]
SELECT ...;
```

- 問い合わせの実行計画を表示。
- ANALYZE：問い合わせを実行して実際の実行時間も表示。

❖実行例

■ PGC0077777を検索した場合の実行計画

```
cat=> EXPLAIN ANALYZE
      SELECT * FROM pgc2003 WHERE id='PGC0077777';
```

- 検索方式：Seq Scan、Index Scan、Bitmap Scan等が存在。
- cost：オプティマイザが最も効率的な問い合わせ方法を見つけるための指標。1コスト=1ページ（テーブルファイルを構成する8192byteの固定長領域）のデータをシーケンシャルに読み込むためにかかる時間。
- rows：推定検索結果行数。
- width：1行あたりの推定平均幅（byte）。

❖【実習】 実行結果は同じだがWHERE句の書き方が異なるSQLの実行時間の比較

1. 以下のSQLは「logd25」が1arcmin以上の天体の数を計測するものである。SQLを3回以上実行し、実行時間の平均時間を調べよ。

■ A

```
cat=> ¥timing
cat=> SELECT COUNT(*)
      FROM pgc2003
      WHERE (10^logd25)*0.1 >= 1;
```

■ B

```
cat=> SELECT COUNT(*)
      FROM pgc2003
      WHERE logd25 >= LOG(1*10);
```

2. EXPLAIN ANALYZE句を使って実行計画を確認せよ。

❖列に対する演算

- WHERE句・ON句で列に対する演算をおこなうと、RDBMSは列の演算結果に対して比較演算を実行する。
- 列の演算結果にはインデックスが存在しないため、列の演算結果全行に対する比較演算（シーケンシャルスキャン）がおこなわれる。
- WHERE句・ON句内での列に対する演算には要注意。

8.7.2. 「8.6.4. 演習問題」の高速化

❖ON句で列に対する演算をしているため遅い

```
cat => SELECT COUNT(*)
       FROM pgc2003 AS a INNER JOIN nvss AS b
       ON
          (
            TRUNC(CAST(a.ra AS numeric),4)=
            TRUNC(CAST(b.ra AS numeric),4)
          )AND(
            TRUNC(CAST(a.dc AS numeric),4)=
            TRUNC(CAST(b.dc AS numeric),4)
          )
       WHERE a.dc BETWEEN -1 AND 1;
```

❖解決方法

- **絞り込み検索**
  - インデックス検索で絞り込んだ後シーケンシャル検索。
- **式インデックス**
  - 計算式（の演算結果）に対してインデックスを設定。
- **あらかじめ計算結果を表に格納してインデックスを設定**
  - よく使われる値は計算しておくと効率的。
  - DBの設計理念的にはX。行を更新したときに演算し忘れないよう注意が必要。

❖絞り込み検索の実施


```
cat => WITH a AS (
         SELECT *
         FROM pgc2003
         WHERE dc between -1 and 1
       ),   b AS (
         SELECT *
         FROM nvss
         WHERE dc between -1 and 1
       )
       SELECT COUNT(*)
       FROM a INNER JOIN b ON
         (
           TRUNC(CAST(a.ra AS numeric),1)=
           TRUNC(CAST(b.ra AS numeric),1)
         )AND(
           TRUNC(CAST(a.dc AS numeric),1)=
           TRUNC(CAST(b.dc AS numeric),1)
         );
```

- 実行時間：~0.3秒
- FROM句での副問い合わせを使って検索範囲を絞り込んだ後に結合。

❖【コラム】列名の重複

■ **以下のSQLは「FROM句にテーブルaが存在しない」というエラーが出て失敗する。**

```
cat => SELECT a.id
       FROM (
         SELECT *
         FROM pgc2003 AS a INNER JOIN nvss AS b ON
         a.ra=b.ra AND a.dc=b.dc
       ) AS t;
```

- FROM句での副問い合わせ内のテーブル名は主SELECT文の参照範囲外。

■ **以下のSQLも「列idの参照があいまいである」というエラーが出て失敗する。**

```
cat => SELECT t.id
       FROM (
         SELECT *
         FROM pgc2003 AS a INNER JOIN nvss AS b ON
         a.ra=b.ra AND a.dc=b.dc
       ) AS t;
```

- FROM句での副問い合わせ結果「t」内に列「id」が2列存在している。

■ 以下のようにSQLを修正すると成功する。

```
cat => SELECT aid
       FROM (
         SELECT a.id AS aid
         FROM pgc2003 AS a INNER JOIN nvss AS b ON
         a.ra=b.ra AND a.dc=b.dc
       ) AS t;
```

- FROM句内での副問い合わせ結果「t」内の列「a.id」と「b.id」を区別するためにそれぞれに別名を定義すれば、主SELECT文で列を特定できる。
- 全列指定「*」を使っているとよく発生するので注意が必要。

9. ユーザ定義関数

9.1. 概要

❖SQL言語とPL/pgSQLを使ったユーザ定義関数の作成方法を紹介

- ユーザ定義関数の作成にはSQL言語、C言語、手続き型言語が使用できる。
- 標準で使用できる手続き型言語は以下の通り。
  - PL/pgSQL、PL/Tcl、PL/Perl、PL/Python
  - ただし、PL/pgSQL以外は要インストール。

❖9章目次

- 9.2. SQL関数
- 9.3. PL/pgSQL関数
- 9.4. SQL関数とPL/pgSQL関数の使い分け

9.2. SQL関数

❖SQL関数

- SQL文で構成される関数。
- 関数内に記述されたSQL文を実行していき、最後に記述されたSQL文の実行結果を返す。

❖9.2節目次

9.2.1. 一行一列の値を返すSQL関数

❖一行一列の値を返すSQL関数の書式

```
CREATE FUNCTION <関数名>(<引数のデータ型>[,･･･])
RETURNS <戻り値のデータ型> AS $$
  <SQL>;
$$ <IMMUTABLE|STABLE|VOLATILE> LANGUAGE SQL;
```

❖実行例

■ 【s921eg1.sql】「Hello, World!」を表示するSQL関数

```
CREATE FUNCTION s921eg1()
RETURNS TEXT AS $$
  SELECT 'Hello, World!';
$$ IMMUTABLE LANGUAGE SQL;

cat=> SELECT s921eg1();
```

- ユーザ定義関数の一覧はメタコマンド「¥df」で確認可能。

■ 【s921eg2.sql】 引数を用いるSQL関数

```
CREATE FUNCTION s921eg2(INTEGER,INTEGER)
RETURNS INTEGER AS $$
  SELECT $1 + $2;
$$ IMMUTABLE LANGUAGE SQL;

cat=> SELECT s921eg2(2000,21);
```

■ 【s921eg3.sql】条件式の値を変数としたSQL関数

```
CREATE FUNCTION s921eg3(TEXT)
RETURNS TEXT AS $$
  SELECT anames FROM pgc2003 WHERE id=$1;
$$ IMMUTABLE LANGUAGE SQL;

cat=> SELECT s921eg3('PGC0077777');
```

❖SQL関数の削除方法

```
cat=> DROP FUNCTION <関数名[(引数のデータ型)]>;
```

- 同じ名前の関数が複数存在する場合は引数のデータ型も指定。

❖一行一列の値を返すSQL関数の解説


```
CREATE FUNCTION <関数名>(<引数のデータ型>[,･･･])
RETURNS <戻り値のデータ型> AS $$
  <SQL>;
$$ <IMMUTABLE|STABLE|VOLATILE> LANGUAGE SQL;
```

- **二重ドル引用符（$$）**
  - SQLコマンド「CREATE FUNCTION」にとって「<SQL>」は文字列であるため単一引用符で囲む必要がある。
  - しかし単一引用符で囲んでしまうと「<SQL>」内で文字列を記述できなくなってしまうため、同じ働きをする二重ドル引用符で囲む。
- **関数の変動性分類（IMMUTABLE,STABLE,VOLATILE）**
  - 関数の振る舞い方をオプティマイザに教えるためのオプション。
  - WHERE句で関数を呼び出した時IMMUTALBLEのみインデックス検索が可能。
  - IMMUTALBE：同一引数に対し常に同一の結果を返す関数。
    - 例：SIN()
  - STABLE：同一問い合わせ内で同一引数に対し同一の結果を返す関数。
    - 例：NOW()
  - VOLATILE：同一引数に対し異なる結果を返す関数。デフォルト設定。
    - 例：RANDOM()
- **関数の引数**
  - 「<SQL>」内の列名や条件式の値に引数「$<引数番号>」を使用できる。
  - テーブル名には引数を使用できない。

9.2.2. 一行複数列の値を返すSQL関数

❖一行複数列の値を返すSQL関数の書式

```
CREATE FUNCTION <関数名>(<引数データ型[,･･･])
RETURNS <複合型の戻り値データ型> AS $$
  <SQL>;
$$ IMMUTABLE|STABLE|VOLATILE LANGUAGE SQL;
```

- 戻り値のデータ型として複合型を指定する必要がある。

❖複合型の作成方法

```
CREATE TYPE <複合型名> AS (
  <要素名1>  <データ型名1>,
  <要素名2>  <データ型名2>,
  ･･･
);
```

- 複合型：複数のデータ型を組み合わせて1つのデータ型として扱うデータ型。
- 列名とデータ型を単に列挙したもの。

❖実行例

■ 【s922eg1.sql】 3つのデータ型要素を持つ複合型の定義

```
CREATE TYPE s922eg1 AS (
  a TEXT,
  b DOUBLE PRECISION,
  c DOUBLE PRECISION
);
```

- ユーザー定義データ型は「¥dT」「¥d データ型」で確認可能。

■ 【s922eg2.sql】 複数列の結果を返すSQL関数

```
CREATE FUNCTION s922eg2()
RETURNS s922eg1 AS $$
  SELECT id,ra,dc FROM pgc2003 LIMIT 20;
$$ IMMUTABLE LANGUAGE SQL;

cat=> SELECT s922eg2();
cat=> SELECT * FROM s922eg2();
```

- 関数の出力結果の列名は複合型の要素名になる。
- この方法だと1行しか返ってこない。

❖ユーザ定義データ型の削除方法

```
cat=> DROP TYPE <データ型名>;
```

9.2.3. 複数行複数列の値を返すSQL関数

❖複数行複数列の値を返すSQL関数の書式

```
CREATE FUNCTION <関数名>(<引数のデータ型>)
RETURNS SETOF <複合型の戻り値のデータ型> AS $$
  <SQL>;
$$ IMMUTABLE|STABLE|VOLATILE LANGUAGE SQL;
```

- SETOF：関数内の本文に記述された複数のSQLのうち、最も最後に記述されたSQLをSQLが終了するまで実行。。

❖実行例

■ 【s923eg1.sql】 複数行複数列の値を返すSQL関数

```
CREATE FUNCTION s923eg1()
RETURNS SETOF pgc2003 AS $$
  SELECT * FROM pgc2003 LIMIT 20;
$$ IMMUTABLE LANGUAGE SQL;

cat=> SELECT * FROM s923eg1();
```

- 戻り値のデータ型としてテーブル名を指定可能。

9.2.4. 演習問題

❖ **【演習】次のSQL関数を作成せよ。**

1. **【s924ex1.sql】テーブル「pgc2003」の列「anames」から任意の天体名の行を検索するSQL関数。**
   - ヒント：結合演算子||を使う。

❖【解答例】

1. 【s924ex1.sql】テーブル「pgc2003」の列「anames」から任意の天体名の行を検索するSQL関数。

■ 部分一致検索

```
CREATE FUNCTION s924ex1(text)
RETURNS SETOF pgc2003 AS $$
  SELECT * FROM PGC2003 WHERE anames LIKE '%'||$1||'%'
$$ IMMUTABLE LANGUAGE SQL;

cat=> SELECT * FROM s924ex1('NGC1');
```

■ 完全一致検索

```
CREATE FUNCTION s924ex1b(text)
RETURNS SETOF pgc2003 AS $$
  SELECT * FROM PGC2003 WHERE anames
  SIMILAR TO '(^'||$1||'%|%_'||$1||'_%|'||$1||'$)'
$$ IMMUTABLE LANGUAGE SQL;
```

- _：スペース。

9.3. PL/pgSQL関数

❖PL/pgSQL

- 非手続き型言語であるSQLを拡張した、PostgreSQLで読み込み可能な手続き型言語。
- SQL関数よりも複雑な演算を行うことができる。
- PostgreSQL9.0以降ではデフォルトで使用可能。

❖9.3節目次

9.3.1. 一行一列の値を返すPL/pgSQL関数

❖一行一列の値を返すPL/pgSQL関数の書式

```
CREATE FUNCTION <関数名>(<仮引数> <データ型>[,･･･])
RETURNS <戻り値のデータ型> AS $$
  DECLARE
    <変数名> <データ型>[:=<値>];
  BEGIN
    RETURN <式>;
  END
$$ IMMUTABLE|STABLE|VOLATILE LANGUAGE PLPGSQL;
```

- DECLARE：変数の宣言をおこなえる。
- RETURN：式の内容を結果として返す。
- :=：変数へ値を代入。PostgreSQL9.4以降は「=」も使用可能。

❖実行例

■ 【s931eg1.sql】「Hello, World!」を表示するPL/pgSQL関数

```
CREATE FUNCTION s931eg1() RETURNS TEXT AS $$
  DECLARE
    x TEXT;
  BEGIN
    x := 'Hello, World!';
    RETURN x;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;

cat=> SELECT s931eg1();
```

■ 【s931eg2.sql】 引数を用いるPL/pgSQL関数

```
CREATE FUNCTION s931eg2(x INTEGER,y INTEGER)
RETURNS INTEGER AS $$
  DECLARE
    z INTEGER;
  BEGIN
    z := x + y;
    RETURN z;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;

cat=> SELECT s931eg2(2000,21);
```

■ 【s931eg3.sql】条件式の値を変数としたSQL関数

```
CREATE FUNCTION s931eg3(x TEXT) RETURNS TEXT AS $$
  DECLARE
    z TEXT;
  BEGIN
    SELECT anames INTO z FROM pgc2003 WHERE id=x;
    RETURN z;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;

cat=> SELECT s931eg3('PGC0077777');
```

- SELECT <列名> INTO <変数>：指定した列の値を変数に代入。

9.3.2. 一行複数列の値を返すPL/pgSQL関数

❖一行複数列の値を返すPL/pgSQL関数の書式

```
CREATE FUNCTION <関数名>(<引数> <データ型>[,･･･])
RETURNS <複合型の戻り値データ型> AS $$
  DECLARE
    <変数名> <複合型>[:=(<値>[,...])];
  BEGIN
    RETURN 式;
  END
$$ IMMUTABLE|STABLE|VOLATILE LANGUAGE PLPGSQL;
```

- 行変数：変数のデータ型として複合型を指定したもの。

■ 【s932eg1.sql】複数列の結果を返すPL/pgSQL関数

```
CREATE FUNCTION s932eg1()
RETURNS s922eg1 AS $$
  DECLARE
    row s922eg1; -- 行変数rowを宣言。
  BEGIN
    SELECT id,ra,dc INTO row FROM pgc2003 LIMIT 20;
    RETURN row;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;

cat=> SELECT * FROM s932eg1();
```

- s922eg1：(a TEXT, b DP, c DP）。
- この方法では1行しか返ってこない。

9.3.3. 複数行複数列の値を返すPL/pgSQL関数

❖複数行複数列の値を返すPL/pgSQL関数の書式

```
CREATE FUNCTION <関数名>(<引数1> <データ型>[,...])
RETURNS SETOF <複合型の戻り値のデータ型> AS $$
  DECLARE
    <変数名> <複合型>[:=(<値>[,...])];
  BEGIN
    RETURN QUERY <SQL>; | RETURN NEXT <変数名>;
    RETURN;
  END
$$ IMMUTABLE|STABLE|VOLATILE LANGUAGE PLPGSQL;
```

- 「SETOF」使用時には「RETURN QUERY」か「RETURN NEXT」が必要。
- RETURN QUERY <SQL>：<SQL>の実行結果を戻り値テーブルに追加。
- RETURN NEXT <変数名>：<変数名>の値を戻り値テーブルに追加。
- RETURN：関数が終了したことを表す。

❖実行例

■ 【s933eg1.sql】複数列の結果を返すPL/pgSQL関数（失敗例）

```
CREATE FUNCTION s933eg1()
RETURNS SETOF s922eg1 AS $$
  DECLARE
    row s922eg1; -- 行変数rowを宣言。
  BEGIN
    SELECT id,ra,dc INTO row FROM pgc2003 LIMIT 20;
    RETURN row;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;

ERROR: RETURN cannot have a parameter in function
returning set
行 7: RETURN row;
```

- 「SETOF」使用時は「RETURN QUERY」か「RETURN NEXT」を使わないと関数を定義できない。

■ 【s933eg2.sql】RETURN QUERYの使用例

```
CREATE FUNCTION s933eg2() RETURNS
SETOF s922eg1 AS $$
  BEGIN
    RETURN QUERY SELECT id,ra,dc FROM pgc2003 LIMIT 20;
    RETURN;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;

cat=> SELECT * FROM s933eg2();
```

■ 【s933eg3.sql】RETURN NEXTの使用例

```
CREATE FUNCTION s933eg3()
RETURNS SETOF pgc2003 AS $$
  DECLARE
    row pgc2003;
  BEGIN
    FOR i IN 1..20 LOOP
      SELECT * INTO row FROM pgc2003 LIMIT 1 OFFSET i-1;
      RETURN NEXT row;
    END LOOP;
    RETURN;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;

cat=> SELECT * FROM s933eg3();
```

- OFFSET：指定した行を飛ばして表示。
- 検索結果を1行ずつ戻り値テーブルに追加している。

9.3.4. PL/pgSQLの制御構文

❖IF文

```
IF (<条件式>) THEN
  <実行文>;
ELSEIF (<条件式>) THEN
  <実行文>;
ELSE
  <実行文>;
END IF;
```

❖CASE文

```
CASE <評価対象>
  WHEN <値> then <実行文>;
  ELSE <実行文>;
END CASE;
```

❖整数FORループ文

```
FOR <変数名> IN <整数値..整数値> LOOP
  <実行文>;
END LOOP;
```

- <変数>はINTEGER型として自動的に定義される。

❖問い合わせ結果FORループ文

```
FOR <行変数|カンマ区切り変数リスト> IN <SQL> LOOP
  <実行文>;
END LOOP;
```

- 問い合わせ結果の各行を<実行文>で処理できるループ文。

❖WHILE文

```
WHILE <条件式> LOOP
  <実行文>;
END LOOP;
```

- 条件式の評価が真である限りループを繰り返す。

9.3.5. 演習問題

- 【演習】次のPL/pgSQL関数を作成せよ。
  1. 【s935ex1.sql】入力された度分秒を度に変換するPL/pgSQL関数。
     - 入力引数の形式は「'12d34m56s'」とする。
     - SUBSTRING関数で文字列の抽出が可能。
       - SUBSTRING('hoge' FROM 3 FOR 2) => ge

❖【解答例】次のPL/pgSQL関数を作成せよ。

1. 【s935ex1.sql】入力された度分秒を度に変換するPL/pgSQL関数。

```
CREATE FUNCTION s935ex1(dms TEXT)
RETURNS DOUBLE PRECISION AS $$
  DECLARE
    d DOUBLE PRECISION;
    m DOUBLE PRECISION;
    s DOUBLE PRECISION;
    r DOUBLE PRECISION;
  BEGIN
    d:=SUBSTRING(dms FROM 1 FOR 2);
    m:=SUBSTRING(dms FROM 4 FOR 2);
    s:=SUBSTRING(dms FROM 7 FOR 2);
    r:=d+(m*1/60)+(s*1/3600);
    RETURN r;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;

cat=> SELECT s935ex1('12d34m56s');
12.58222...
```

- 変数の型と代入値の型が異なる場合、代入値を変数の型に変換する型変換が自動的に試みられる。

9.4. SQL関数とPL/pgSQL関数の使い分け

❖ SQL関数

- SQLの知識だけで書くことができる。
- 静的なSQLしか書くことができない。
  - テーブル名を仮引数とすることができない。
- PL/pgSQLよりもパフォーマンスが良好。

❖ PL/pgSQL関数

- 言語仕様の知識が必要。
- 手続き型言語であるため、テーブルとは無関係な何らかの処理を行なって1つの値を返す場合に使うのが基本。
- SQL関数と比較して極めて複雑な処理を行うことができる。

❖ 使い分け

- テーブルを扱わない関数 => PL/pgSQL関数
- テーブルを扱う関数 => SQL関数 or PL/pgSQL関数

10. 指定座標を中心とした任意角度範囲内の天体の検索

10.1. 概要

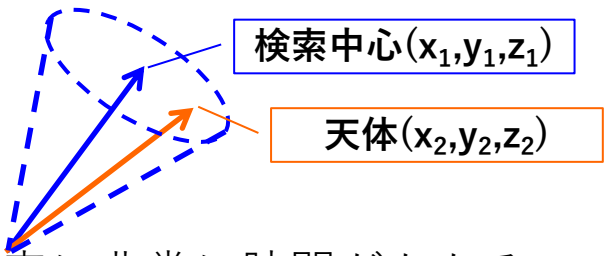
❖ コーンサーチ・ラジアルサーチ

- 天球面上のある座標を中心とした任意角度範囲内にある天体を検索する方法。
- 天文業界ではコーンサーチの実装手法がいくつか確立されているが、本講習会では直交座標系を利用したコーンサーチの実装手法を紹介。

❖ 直交座標系を利用したコーンサーチ

- 天体の位置を単位ベクトルで表し、検索中心と天体との2点間の角度を内積で求め、求めた角度が任意角度以下となる天体を検索。

```
SELECT *
FROM <テーブル名>
WHERE ACOS(x1x2+y1y2+z1z2) < 任意角度;
```



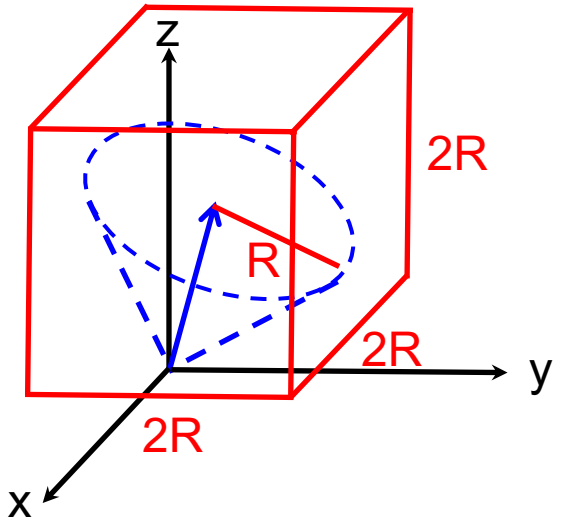
- WHERE句内で列に対する演算をおこなうため検索に非常に時間がかかる。
- 絞り込み検索をおこなう必要がある。

❖絞り込み検索を用いたコーンサーチの実装手法

1. 天体の赤道座標を直交座標に変換。
2. 検索半径がRの時、検索中心を中心とした1辺2Rの立方体の中にある天体をインデックス検索で抽出。
3. 2で取り出した天体に対して検索中心からの角度を計算し、検索半径内にある天体を取り出す。

❖10章目次

- 10.2. 直交座標への変換
- 10.3. コーンサーチの実装
- 10.4. クロスマッチの実装



10.2 直交座標への変換

❖天体の赤道座標を直交座標に変換しビューに格納。

❖10.2節目次

- 10.2.1 座標変換用関数の作成
- 10.2.2 直交座標系ビューの作成
- 10.2.4 直交座標系ビューに対する検索の高速化

10.2.1 座標変換用関数の作成

❖ **【演習】 座標変換用関数の作成**

1. **【fEq2(X|Y|Z).sql】赤道座標を直交座標に変換するPL/pgSQL関数「fEq2X, fEq2Y, fEq2Z」をそれぞれ作成せよ。**
   - 引数と戻り値の型：DOUBLE PRECISION
   - 参：「8.4.6. 演習問題」
2. **次のSQLを実行し値が~1になることを確認せよ。**

```
cat=> SELECT SQRT(
                fEq2X(ra,dc)^2 +
                fEq2Y(ra,dc)^2 +
                fEq2Z(dc)^2
              )
      FROM pgc2003
      LIMIT 20;
```

❖【解答例】座標変換用関数の作成

1. 【fEq2(X|Y|Z).sql】赤道座標を直交座標に変換するPL/pgSQL関数「fEq2X, fEq2Y, fEq2Z」をそれぞれ作成せよ。

■ fEq2X.sql

```
CREATE FUNCTION fEq2X(ra DOUBLE PRECISION,dc DOUBLE PRECISION)
RETURNS DOUBLE PRECISION AS $$
  BEGIN
    RETURN COS(RADIANS(ra))*COS(RADIANS(dc));
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;
```

■【fEq2Y.sql】

```
CREATE FUNCTION fEq2Y(ra DOUBLE PRECISION,dc DOUBLE PRECISION)
RETURNS DOUBLE PRECISION AS $$
  BEGIN
    RETURN SIN(RADIANS(ra))*COS(RADIANS(dc));
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;
```

■【fEq2Z.sql】

```
CREATE FUNCTION fEq2Z(dc DOUBLE PRECISION)
RETURNS DOUBLE PRECISION AS $$
  BEGIN
    RETURN SIN(RADIANS(dc));
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;
```

10.2.2. 直交座標系ビューの作成

❖CREATE VIEW文の書式

```
CREATE VIEW <ビュー名> (<列名>[,...]) AS
SELECT <列名>[,...] FROM <テーブル名>;
```

❖ビュー

- テーブルから任意の列を取り出して仮想的なテーブルを作成する機能。
- SELECT文内でテーブルと同様に扱える。
- 内部的には「CREATE VIEW」で定義したSQLを問い合わせのたびに実行。
- 使用例：
  - 列から計算した値をテーブルのように扱いたい時。
  - ユーザーに対しテーブルの一部の列を隠蔽したい時。

❖【実習】直交座標系ビューの作成

- 【CreateViewPgc2003_xyz.sql】テーブル「pgc2003」の赤道座標を直交座標に変換するビュー「pgc2003_xyz」を作成せよ。

```
cat=> CREATE VIEW pgc2003_xyz(id,x,y,z) AS
      SELECT id,
             fEq2X(ra,dc),
             fEq2Y(ra,dc),
             fEq2Z(dc)
      FROM pgc2003;
```

- 【CreateViewNvss_xyz.sql】テーブル「nvss」の赤道座標を直交座標に変換するビュー「nvss_xyz」を作成せよ。

```
cat=> CREATE VIEW nvss_xyz(id,x,y,z) AS
      SELECT id,
             fEq2X(ra,dc),
             fEq2Y(ra,dc),
             fEq2Z(dc)
      FROM nvss;
```

- 【上記ビューが機能しているか確認せよ。

```
cat=> ¥d
cat=> SELECT * FROM pgc2003_xyz LIMIT 20;
cat=> SELECT id,SQRT(x^2+y^2+z^2)
      FROM pgc2003_xyz LIMIT 20;
```

10.2.4. 直交座標系ビューに対する検索の高速化

❖【実習】直交座標系ビューの検索時間の測定

■ 【s1024eg1.sql】ビュー「nvss_xyz」から適当な立方体領域にある天体を検索

```
cat=> EXPLAIN ANALYZE
      SELECT COUNT(*)
      FROM nvss_xyz
      WHERE (x BETWEEN 0.8 AND 0.9) AND
            (y BETWEEN 0 AND 0.1) AND
            (z BETWEEN -0.6 AND -0.5);
```

■ 上記SQLの実態

```
SELECT COUNT(*)
FROM nvss_xyz
WHERE
  (COS(RADIANS(ra))*COS(RADIANS(dc)) BETWEEN 0.8 AND 0.9) AND
  (SIN(RADIANS(ra))*COS(RADIANS(dc)) BETWEEN 0 AND 0.1) AND
  (COS(RADIANS(dc)) BETWEEN -0.6 AND -0.5);
```

- 列「ra,dc」はインデックスが設定されているが、WHERE句内で列に対する演算をおこなっているためシーケンシャル検索となり遅い。
- だからといって赤経赤緯で立方体内天球面の範囲を表すわけにもいかない。
- **式インデックスと複合インデックスを活用する。**

❖式インデックス

- 列を用いた式や関数に対して設定するインデックス。
- 演算結果に基づいた高速な検索が可能。
- 行が挿入・更新される度に式の計算が必要なため、通常のインデックスよりも運用コストが高い。

❖実行例

■ **列「id」から'PGC'を削除した値にインデックスを設定**

```
cat=> CREATE INDEX ON pgc2003(TRIM('PGC' FROM id));
```

■ **検索速度の比較**

```
cat=> SELECT *
      FROM pgc2003
      WHERE TRIM('PGC' FROM id)='0077777';
```

- 式インデックス無し：~150 ms
- 式インデックス有り：~0.5 ms

❖複合インデックス

- 複数の列の組み合わせに設定するインデックス。
- 複数で一つの情報を表す列に設定。
  - 赤経と赤緯、苗字と名前など。
- 通常のインデックスよりも検索速度が向上する可能性がある。

❖実行例

■ **列「ra,dc」に複合インデックスを設定**

```
cat=> CREATE INDEX ON pgc2003(ra,dc);
```

■ **検索速度の比較**

```
cat=> SELECT *
      FROM pgc2003
      WHERE (ra BETWEEN 90 AND 100) AND
            (dc BETWEEN -5 and 5);
```

- インデックス有り：~8 ms
- 複合インデックス有り：~1.5 ms

❖複合インデックスの注意点

- インデックスの定義順序が検索性能に影響。
  - 列1、列2の順番で絞り込むのが最適な場合は、この順番で複合インデックスを定義すべき。
- 1つの列に対してのみ検索を行なった時、1番目に定義された列以外はインデックス検索を行えない。
  - pgc2003(ra,dc)設定時、dcのみの検索はシーケンシャルに。

❖【実行例】

■ 検索速度の比較

```
cat=> SELECT *
      FROM pgc2003
      WHERE dc BETWEEN -5 and 5;
```

- インデックス有り：~40ms
- 複合インデックス有り：~120 ms
- **検索方法が定型化されていない限り、複合インデックスの採用は慎重に。**

❖【実習】複合インデックスの作成

■ テーブル「pgc2003」に複合インデックスを設定せよ。

```
cat=> CREATE INDEX ON pgc2003(
        fEq2X(ra,dc),
        fEq2Y(ra,dc),
        fEq2Z(dc)
      );
```

- なおx、y、zの検索優先度は同一であるため列の定義順序は何でも良い。

■ テーブル「nvss」に複合インデックスを設定せよ。

```
cat=> CREATE INDEX ON nvss(
        fEq2X(ra,dc),
        fEq2Y(ra,dc),
        fEq2Z(dc)
      );
```

■ 【s1024eg1.sql 】SQLの実行時間が短くなったか確認せよ。

```
cat=> EXPLAIN ANALYZE
      SELECT COUNT(*)
      FROM nvss_xyz
      WHERE (x BETWEEN 0.8 AND 0.9) AND
            (y BETWEEN 0 AND 0.1)   AND
            (z BETWEEN -0.6 AND -0.5);
```

10.3. コーンサーチの実装

❖コーンサーチの実装手法

1. 天体の赤道座標を直交座標に変換（**済**）。
2. 検索半径がRの時、検索中心を中心とした1辺2Rの立方体の中にある天体をインデックス検索でとりだす。
3. 2で取り出した天体に対して検索中心からの角度を計算し、検索半径内にある天体を取り出す。

❖10.3節目次

10.3.1. コーンサーチに必要な関数の作成

❖ 必要な関数

- 天球面上の2点間の角度（分角）を求める関数
- 分角をラジアンに変換する関数

❖ 【演習】2点間の角度を求める関数

1. 【fDistanceArcminXyz.sql】天球面上の点1（x1, y1, z1）と点2（x2, y2, z2）の間の角度（分角）を求めるPL/pgSQL関数を作成せよ。
   - 引数と戻り値の型：DOUBLE PRECISION
   - 関数の内容：

```
a := x1*x2+y1*y2+z1*z2;
IF (a>1) THEN
  a := 1;
ELSEIF (a<-1) THEN
  a := -1;
END IF;
RETURN DEGREES(ACOS(a))*60;
```

- 丸め誤差を警戒してACOSの定義域を制限。

2. 次のSQLを実行し、結果が「5400」になることを確認せよ。

```
cat=> SELECT fDistanceArcminXyz(1,0,0,0,1,0);
```

3. 【fArcmin2Rad.sql】分角をラジアンに変換するPL/pgSQL関数を作成せよ。
   - 引数と戻り値の型：DOUBLE PRECISION
   - 度をラジアンに変換する関数：RADIANS(DP)
4. 次のSQLを実行し、結果が「3.14...」になることを確認せよ。

```
cat=> SELECT fArcmin2Rad(180*60);
```

❖【解答例】

1. 【fDistanceArcminXyz.sql】天球面上の点1（x1, y1, z1）と点2（x2, y2, z2）の間の角度（分角）を求めるPL/pgSQL関数を作成せよ。

```
CREATE FUNCTION fDistanceArcminXyz (
  x1 DOUBLE PRECISION,
  y1 DOUBLE PRECISION,
  z1 DOUBLE PRECISION,
  x2 DOUBLE PRECISION,
  y2 DOUBLE PRECISION,
  z2 DOUBLE PRECISION
) RETURNS DOUBLE PRECISION AS $$
  DECLARE
    a DOUBLE PRECISION;
  BEGIN
    a := x1*x2+y1*y2+z1*z2;
    IF (a>1) THEN
      a := 1;
    ELSEIF (a<-1) THEN
      a := -1;
    END IF;
    RETURN DEGREES(ACOS(a))*60;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;
```

3. 【fArcmin2Rad.sql】分角をラジアンに変換するPL/pgSQL関数「fArcmin2Rad」を作成せよ。

```
CREATE FUNCTION fArcmin2Rad (m DOUBLE PRECISION)
RETURNS DOUBLE PRECISION AS $$
  BEGIN
    RETURN RADIANS(m/60);
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;
```

10.3.2. コーンサーチ用SQLの作成

❖【実習】赤経200度、赤緯5度を中心とした半径60分内にある天体を検索する場合のSQLの作成

■【s1032eg1.sql】1辺2Rの立方体内にある天体のインデックス検索

```
cat=> SELECT t1.id,t1.ra,t1.dc
      FROM pgc2003 AS t1 INNER JOIN pgc2003_xyz AS t2
                         ON t1.id=t2.id
      WHERE (x BETWEEN fEq2X(200,5)-fArcmin2Rad(60)
                   AND fEq2X(200,5)+fArcmin2Rad(60)
      ) AND (y BETWEEN fEq2Y(200,5)-fArcmin2Rad(60)
                   AND fEq2Y(200,5)+fArcmin2Rad(60)
      ) AND (z BETWEEN fEq2Z(5)-fArcmin2Rad(60)
                   AND fEq2Z(5)+fArcmin2Rad(60)
      );
```

- fEq2(X|Y|Z)+/-farcmin2Rad(60)で検索中心+/-Rを表現。
- pgc2003の情報を得るためにpgc2003とpgc2003_xyzを内部結合。

■ 【s1032eg2.sql】1辺2Rの立方体内にある天体に対して検索中心との離角を計算し指定した半径（60度）以内にあるものを表示

```
SELECT t.id,t.ra,t.dc,t.distance
FROM (
  SELECT t1.id,t1.ra,t1.dc,
         fDistanceArcminXyz(
           fEq2X(200,5),
           fEq2Y(200,5),
           fEq2Z(5),
           x,y,z
         ) AS distance
  FROM pgc2003 AS t1 INNER JOIN pgc2003_xyz AS t2
                       ON t1.id=t2.id
  WHERE (x BETWEEN fEq2X(200,5)-fArcmin2Rad(60)
               AND fEq2X(200,5)+fArcmin2Rad(60)
  ) AND (y BETWEEN fEq2Y(200,5)-fArcmin2Rad(60)
               AND fEq2Y(200,5)+fArcmin2Rad(60)
  ) AND (z BETWEEN fEq2Z(5)-fArcmin2Rad(60)
               AND fEq2Z(5)+fArcmin2Rad(60)
  )
) AS t
WHERE t.distance <= 60;
```

- f DestanceArcminXyzで離角を測定。
- 主SELECT分のWHERE句で離角が条件を満たすか判定。

10.3.3. コーンサーチの実装

❖【演習】 コーンサーチ用SQL関数の作成

1. **【TypeCone.sql】 コーンサーチ用ユーザ定義関数の実行結果を返すために必要な複合データ型「TypeCone」を作成せよ。**
   - 複合型の要素
     - id TEXT
     - ra DOUBLE PRECISION
     - dc DOUBLE PRECISION
     - distance DOUBLE PRECISION
2. **【fConesearchPgc2003.sql】 テーブル「pgc2003」に対しコーンサーチをおこなうSQL関数を作成せよ。**
   - 引数は「ra,dc,r」
   - 引数の型：DOUBLE PRECISION
   - 戻り値の型：SETOF TypeCone

3. 以下のSQLの検索件数がSQL「s1032eg2.sql」と同じ結果になるか確認せよ。

```
cat=> SELECT * FROM fConesearchPgc2003(200,5,60);
```

4. 【fConesearchNvss.sql】 テーブル「nvss」用のコーンサーチ用SQL関数を作成せよ。

❖【解答例】 コーンサーチ用SQL関数の作成

1. 【TypeCone.sql】コーンサーチ用ユーザ定義関数の実行結果を返すために必要な複合データ型「TypeCone」を作成せよ。

```
CREATE TYPE TypeCone AS (
  id          TEXT,
  ra          DOUBLE PRECISION,
  dc          DOUBLE PRECISION,
  distance    DOUBLE PRECISION
);
```

2. 【fConesearchPgc2003.sql】テーブル「pgc2003」に対しコーンサーチをおこなうSQL関数を作成せよ。

```
CREATE FUNCTION fConesearchPgc2003 (
                  DOUBLE PRECISION,
                  DOUBLE PRECISION,
                  DOUBLE PRECISION
                ) RETURNS SETOF TypeCone AS $$
  SELECT t.id,t.ra,t.dc,t.distance
  FROM (
    SELECT t1.id,t1.ra,t1.dc,
           fDistanceArcminXyz(
             fEq2X($1,$2),
             fEq2Y($1,$2),
             fEq2Z($2),
             x,y,z
           ) AS distance
    FROM pgc2003 AS t1 INNER JOIN pgc2003_xyz AS t2
                         ON t1.id=t2.id
    WHERE (x BETWEEN fEq2X($1,$2)-fArcmin2Rad($3)
                 AND fEq2X($1,$2)+fArcmin2Rad($3)
    ) AND (y BETWEEN fEq2Y($1,$2)-fArcmin2Rad($3)
                 AND fEq2Y($1,$2)+fArcmin2Rad($3)
    ) AND (z BETWEEN fEq2Z($2)-fArcmin2Rad($3)
                 AND fEq2Z($2)+fArcmin2Rad($3)
    )
  ) AS t
  WHERE t.distance <= $3;
$$ IMMUTABLE LANGUAGE SQL;
```

❖【コラム】 コーンサーチ用PL/pgSQL関数の例

- 関数中のFROM句で仮引数や変数を使うことはできない。
- テーブル名を引数で指定したい場合はPL/pgSQL関数でSQL文を文字列として動的に生成し、EXECUTE文で文字列の内容を実行する。

```
CREATE FUNCTION fConesearchPl (
                  ra  DOUBLE PRECISION,
                  dc  DOUBLE PRECISION,
                  r   DOUBLE PRECISION,
                  tbl TEXT
                ) RETURNS SETOF TypeCone AS $$
  DECLARE
    sql  TEXT;
  BEGIN
    sql:='
      SELECT t.id,t.ra,t.dc,t.distance
      FROM (
        SELECT t1.id,t1.ra,t1.dc,
               fDistanceArcminXyz(
                 fEq2X('||ra||','||dc||'),
                 fEq2Y('||ra||','||dc||'),
                 fEq2Z('||dc||'),
                 x,y,z
               ) AS distance
        FROM '||tbl||' AS t1 INNER JOIN '||tbl||'_xyz AS t2 ON t1.id=t2.id
        WHERE (x BETWEEN fEq2X('||ra||','||dc||')-fArcmin2Rad('||r||')
                     AND fEq2X('||ra||','||dc||')+fArcmin2Rad('||r||')
        ) AND (y BETWEEN fEq2Y('||ra||','||dc||')-fArcmin2Rad('||r||')
                     AND fEq2Y('||ra||','||dc||')+fArcmin2Rad('||r||')
        ) AND (z BETWEEN fEq2Z('||dc||')-fArcmin2Rad('||r||')
                     AND fEq2Z('||dc||')+fArcmin2Rad('||r||')
        )
      ) AS t
      WHERE t.distance <= '||r ;
    RETURN QUERY EXECUTE sql;
    RETURN;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;
```

10.4. クロスマッチの実装

❖クロスマッチの実装

- コーンサーチを利用すれば、複数のカタログ間での天体のクロスマッチを簡単に実装できる。
- 実装手法
  1. カタログAから天体の赤経赤緯を取り出す。
  2. カタログBから1で取り出した赤経赤緯を中心にコーンサーチをおこない、最も近傍の天体を取り出す。
- 必要なもの
  - 検索半径内で最も検索中心に近い天体を取り出すクロスマッチ用関数

❖10.4節目次

10.4.1. クロスマッチの実装

❖【実習】クロスマッチ用関数の作成

- 【fGetNearestNvssObjID.sql】テーブル「nvss」に対してコーンサーチをおこない、検索中心に最も近い天体の「id,ra,dc,distance」を返す関数

```
CREATE FUNCTION fGetNearestNvssObjID (
                  DOUBLE PRECISION,
                  DOUBLE PRECISION,
                  DOUBLE PRECISION
                ) RETURNS SETOF TypeCone AS $$
  SELECT t.id,t.ra,t.dc,t.distance
  FROM (
    SELECT t1.id,t1.ra,t1.dc,
           fDistanceArcminXyz(
             fEq2X($1,$2),
             fEq2Y($1,$2),
             fEq2Z($2),
             x,y,z
           ) AS distance
    FROM nvss AS t1 INNER JOIN nvss_xyz AS t2 ON t1.id=t2.id
    WHERE (x BETWEEN fEq2X($1,$2)-fArcmin2Rad($3)
                 AND fEq2X($1,$2)+fArcmin2Rad($3)
    ) AND (y BETWEEN fEq2Y($1,$2)-fArcmin2Rad($3)
                 AND fEq2Y($1,$2)+fArcmin2Rad($3)
    ) AND (z BETWEEN fEq2Z($2)-fArcmin2Rad($3)
                 AND fEq2Z($2)+fArcmin2Rad($3)
    )
  ) AS t
  WHERE t.distance <= $3
  ORDER BY t.distance
  LIMIT 1;
$$ IMMUTABLE LANGUAGE SQL;
```

10.4.2. クロスマッチの実行

❖【実習】 クロスマッチの実行

- 【s1042eg1.sql】テーブル「pgc2003」から「NGC5194」を検索し、テーブル「NVSS」から1分角でクロスマッチ

```
WITH t1 AS (
  SELECT id,ra,dc
  FROM pgc2003
  WHERE anames LIKE '%NGC5194%'
)
SELECT * FROM t1 CROSS JOIN
              fGetNearestNvssObjID(t1.ra,t1.dc,1) AS t2;
```

おわりに

- **ADCの講習会ウェブページには過去のSQL講習会の資料も公開されています。**
- **特に「2016年度SQL講習会[中級編：データベース構築編]資料（山内 千里）」では本講習会の内容に加え、以下のような高度な手法も紹介されています。**
  - C言語でのユーザー定義関数の作成方法
  - テーブルパーティショニングの方法
  - 画像ファイルの検索方法