データベース講習会（初級編）

於　国立天文台三鷹キャンパス
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
2019年12月5—6日

目次

1. はじめに

自己紹介

名前：小澤 武揚
おざわ たけあき

生年月日：昭和63年9月1日

所属：天文データセンター

趣味：写真

データベース歴：4年

- 多波長データ解析システムのシステム情報、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 本講習会の内容

- 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、データベースの構築方法、データベースの問い合わせ方法を、Linux端末を操作して実践的に学び習得する。
- 本講習会では次の天文カタログをデータベース化する。
  - PGC2003カタログ：983261天体
  - NVSSカタログ：1773484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 11.5文書（https://www.postgresql.jp/document/11/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、検索システム、Web通販の商品情報・アカウント情報、銀行の勘定系システム・・・
- 観測データ、天体カタログ、文献データベース・・・

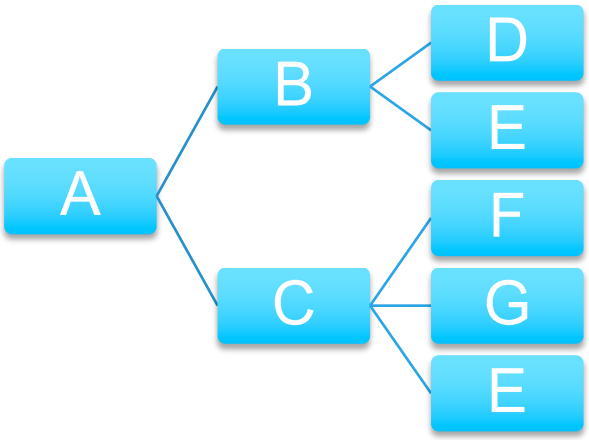
2.2 データベースの種類

- データベースは主に「階層型」、「ネットワーク型」、「リレーショナル型」に分類できる
- 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。
    - 組織図やファイル管理のような仕組み。
  - 欠点
    - データが冗長的になる。
      - 親データは複数の子データを持てるが、子データは1つの親データしか持てない。子データが複数の親データに属する場合、それぞれの親データの下に同一の子データが配置される。
    - データの構造が変化した場合、プログラムの修正が必要。
      - ファイルのパスを変更した時、そのファイルを使用しているプログラムも修正しなければならない。

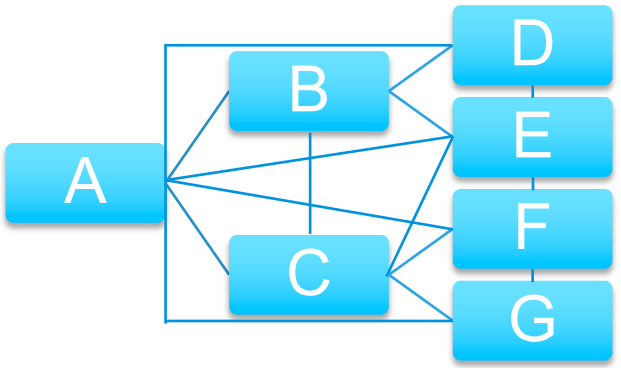
❖ネットワーク型データベース

- データを網の目構造で管理するデータベース。
- 利点
  - データの冗長化が起こらない。
- 欠点
  - データ同士の関係が複雑になる。
  - データ構造が変化した場合、プログラムの修正が必要。

階層型データベース



ネットワーク型データベース



❖リレーショナル型データベース(RDB)

- データを行と列からなる表で管理するデータベース。
  - データの関係を表と表の関係で表す。
    - 表と表に共通するデータをもたせることで関連付けを行う。
  - SQLによりデータを自由に取り出せる。
- 利点
  - プログラムとデータの分離。
    - データの独立性が高く、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。

社員データベース

id	名字	名前
1	山田	一郎
2	山田	二郎
3	山田	三郎

所属部署

id	部署
1	A
2	B
3	A
3	B

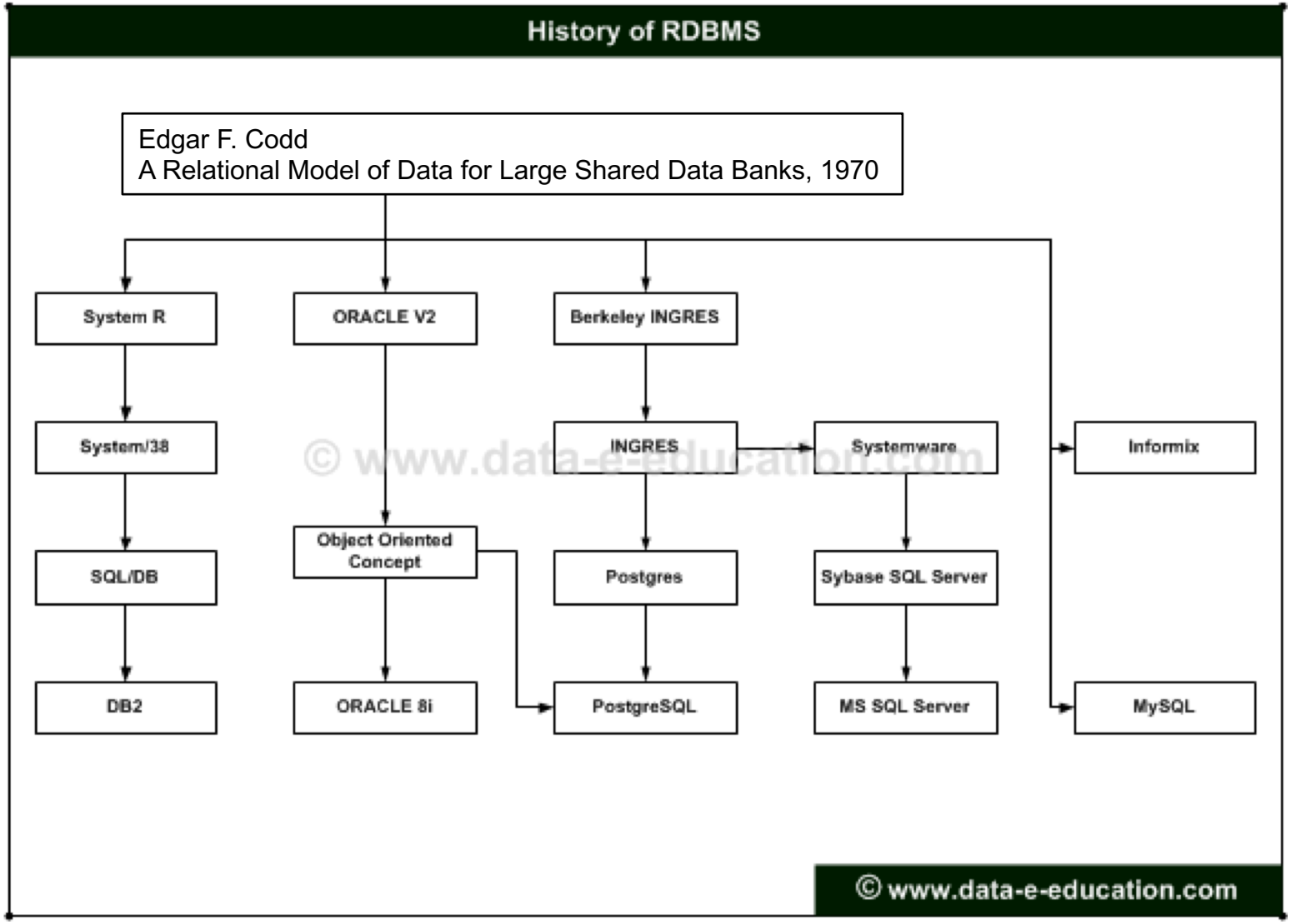
2.3 リレーショナルデータベース管理システム

❖リレーショナルデータベース管理システム（RDBMS）

- RDB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する。
    - 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する。
- データベースへの検索要求はSQL言語で行われる。

❖RDBMSの系譜

– http://data-e-education.com/E107_History_of_RDBMS.html より



2.4 SQL言語

❖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規格の作成が進められている（最新： SQL:2016）。
- RDBMSは基本的に標準SQL規格に対応するように開発が進められており、異なるRDBMSを使っても使用するSQLが”大きく”変わることはない。
  - 関数に関してはRDBMSによって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）によって開発が行われている。
  - PGDGは、5人のコアメンバー、29人の主要開発者（日本人2名）、数十名の貢献者から構成される（2019年10月現在）。
- 「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。
- OSSのRDBMSとしては、日本ではPostgreSQLのシェアがMySQLを上回っていたが、2009年前後を境にMySQLのシェアがPostgreSQLを上回っている。
  - 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として知られるIngresに端を発している。1986年にカリフォルニア大学バークレー校（UCB）でIngresの後継としてオブジェクト指向を取り入れたRDBMSである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した。
- 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が公開された。
- 1997年には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が公開された。同時期にPGDGが結成され、インターネットを通じた開発体制が確立された。また1999年には日本のユーザコミュニティである日本PostgreSQLユーザ会（JPUG）が設立された。
- 2019年現在ではPostgreSQL 12が公開されている。

3.3 PostgreSQLの特徴

❖ 天文学の研究や業務にとって嬉しい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、資料が充実している。
- 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、ODBC、Perl、PHP、Python、Ruby他、多くの言語からアクセス可能。
  - レプリケーション（データベースの複製同期機能）をサポート
  - 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。
  - SSL対応

- 天文データ検索向きの機能
  - 強力なユーザ定義関数
    - 通常RDBMSの関数は返り値として1つの値しか返せないが、PostgreSQLではテーブル全体を返すことができる。この機能はオブジェクトRDBMSであるPostgreSQLの大きな特徴であり、複雑な処理も単純な記述で実現できる。
  - 強力なユーザ定義関数の開発環境
    - SQL言語、C言語、PL/pgSQL、PL/Tcl、PL/Perl、PL/Python他、多くの言語を使ってユーザ定義関数を作成することができる。
  - 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。

❖ PostgreSQLが科学用途に向く理由

- 天文データベースでは座標に対する検索が頻繁に行われるが、座標に対する検索はRDBMSにとってはやや特殊な処理であることが多く、何らかのアルゴリズムの実装と最適化が必要となる場合が多い。
- 例えば本講習会で行う「ある座標を中心とした任意角度範囲内の天体の検索」では、天球面上の2点間の角度を高速で計算するアルゴリズムを実装しなければならない。
- PostgreSQLは強力なユーザ定義関数とその開発環境を備えており、目的のアルゴリズムの実装と最適化を簡単に行うことができる。

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（ことが期待される）

※PostgreSQL 11 時点

- 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

本講習会ではCentOS 7へのインストールを実施

4. PostgreSQLのインストール

4.1 概要

❖ CentOS 7へのPostgreSQLのインストールを実施

- 本講習会では端末「new-rXX」上の仮想マシンにインストールされたCentOS 7を使用する。

❖ 作業環境

- 端末名：new-r[01-13]
  - CPU：Intel Xeon W-2123(3.6GHz, 4コア)
  - メモリ：16GB DDR4 SDRAM
  - OS：CentOS 7(64bit)
- 仮想マシン：Oracle VM VirtualBox
  - CPU：2コア
  - メモリ：8GB
  - OS：CentOS 7
    - インストール時のベース環境：サーバ（GUI使用）

❖ 4章目次

4.2 作業準備

❖【実習】new-rXXへのログイン

1. ターミナルへアカウントとパスワードを入力。

アカウント：ホワイトボード参照
パスワード：ホワイトボード参照

❖【実習】ゲストOSへのログイン

1. デスクトップ画面左上の「Applications」をクリックし、「System Tools」にカーソルを合わせ一覧から「Oracle VM VirtualBox」をクリックして起動。
2. 一覧から「CentOS7-DB」をダブルクリックし起動。
3. ログイン画面で「lecture」をクリックし、ゲストOS用のパスワードを入力。

アカウント：`lecture`
パスワード：`lecdb1912`

- ゲストOSの画面が小さい（余白が大きい）場合は、VirtualBoxのウィンドウを拡大縮小してください。ゲストOSの画面がVirtualBoxのウィンドウにフィットします。

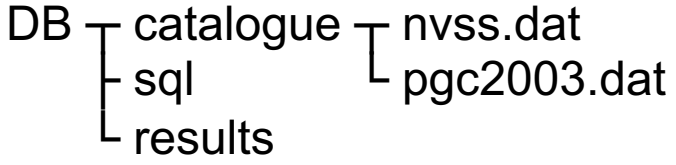
❖ゲストOS（CentOS7）の設定内容

- 画面ロック機能の解除
- [root]# yum update
- [root]# yum install kernel-devel
  - VirtualBoxのGuest Additionsを有効にするために必要だった。
- [root]# yum install gcc
- [root]# yum install emacs
- [root]# yum install ipa-gothic-fonts ipa-mincho-fonts ipa-pgothic-fonts ipa-pmincho-fonts
- [root]# yum install screen
- [lecture]$ LANG=C xdg-user-dirs-gtk-update
  - /home以下の日本語名のディレクトリを英語に変更。
- bashrcの設定
  - [root & dbr]#$ emacs ~/.bashrc
    - 設定内容は.bashrcを参照。
- emacsの設定
  - [root & dbr]#$ mkdir -p ~/.emacs.d/backup
  - [root & dbr]#$ emacs ~/.emacs.d/.init.el
    - 設定内容は.init.elを参照。

❖【実習】ゲストOS上での作業環境の確認

1. デスクトップ画面左上の「アプリケーション」をクリックし、「お気に入り」にカーソルを合わせ一覧から「端末」を選択。
2. 作業用ディレクトリの確認。

```
$ ls
DB  Documents Music Public Videos
Desktop Downloads Pictures Templates
$ ls DB
```



- 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
- results：出力結果を収めるディレクトリ

❖ゲストOS上のファイルのUSBメモリへの移動方法

1. new-rXXにUSBメモリを挿す。
2. 「デバイス」>「USB」>「USBメモリ名」> を選択し、ゲストOSにUSBメモリをマウント。
3. データを移動。
4. 2と同手順でUSBメモリをアンマウント。
5. new-rXXからUSBメモリをアンマウント。

4.3 PostgreSQLのインストール

❖【実習】本家ウェブサイトからPostgreSQLをダウンロードしCentOS 7にインストール

1. デスクトップ画面左上の「アプリケーション」をクリックし、「お気に入り」にカーソルを合わせ一覧から「Firefox」を選択し起動。
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. 「Binary packages」欄の「Red Hat」をクリック。
5. 「To use the PostgreSQL Yum Repository, follow these steps」: 以降の項目を、以下のように選択。

```
1. Select version:11
2. Select platform:CentOS 7
3. Select architecture:x86_64
```

■ PostgreSQLダウンロードページ入力結果

To use the PostgreSQL Yum Repository, follow these steps:

1. Select version:

   11

2. Select platform:

   RedHat Enterprise, CentOS, Scientific or Oracle version 7

3. Select architecture:

   x86_64

4. Install the repository RPM:

```
yum install https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

5. Install the client packages:

```
yum install postgresql11
```

6. Optionally install the server packages:

```
yum install postgresql11-server
```

7. Optionally initialize the database and enable automatic start:

```
/usr/pgsql-11/bin/postgresql-11-setup initdb
systemctl enable postgresql-11
systemctl start postgresql-11
```

- PostgreSQL12はリリースされたばかりで検証していないため11を使います。

6. PostgreSQLのリポジトリを登録。

```
[lecture]$ su
パスワード:lecdb1912
[root]# yum install
https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-
x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

7. PostgreSQLのパッケージをインストール。

```
[root]# yum install postgresql11
                    postgresql11-server
                    postgresql11-devel
                    postgresql11-contrib
                    postgresql11-libs
```

- postgresql11
  - 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。
- postgresql11-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。
- postgresql11-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。
- postgresql11-contrib
  - OSコマンドや関数など。
- postgresql11-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。

8. データベースクラスタの作成。

作成方法

```
[root]# su postgres
bash-4.2$ /usr/pgsql-11/bin/initdb
          --no-locale
          --encoding=UTF8
          -D /var/lib/pgsql/11/data
bash-4.2$ exit
```

- データベースクラスタ：データベースを格納する領域。
- initdb：データベースクラスタを作成するコマンド。
  - --no-locale：デフォルトではOSで設定されているロケール（ここではja_JP）が設定される。ロケールの設定を行うと、データベース内での文字列処理、日付や通貨の表示、メッセージの言語などを地域化することができる。地域化を行うとPostgreSQLの一部の機能が使えなくなるため、ロケールは設定しない。
  - --encording=UTF8：データベースのデフォルトの符号化方式を指定。
  - -D：データベースクラスタを作成するディレクトリを指定。

削除方法

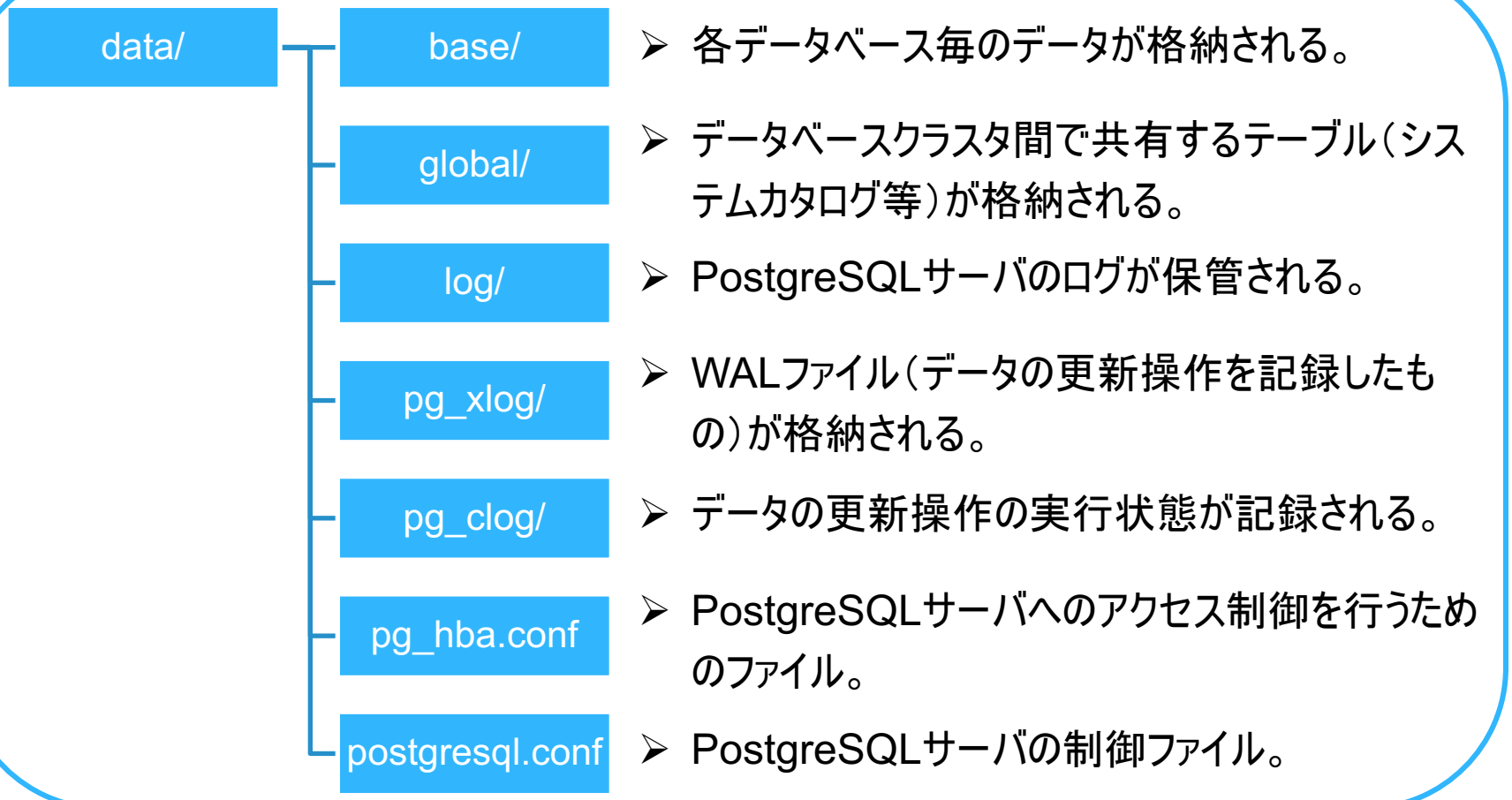
```
[lecture]$ su
パスワード:lecdb1912
[root]# systemctl stop postgresql-11.service
[root]# su postgres
bash-4.2$ rm –r /var/lib/pgsql/11/data
```

- データベースクラスタを削除することで、データベースを完全に初期化できる。

■ データベースクラスタ概略

- 主要なディレクトリとファイルについて。

```
[root]# ls -la /var/lib/pgsql/11/data
```



9. PostgreSQLの起動方法。

起動

```
[root]# systemctl start postgresql-11.service
```

状態の確認

```
[root]# systemctl status postgresql-11.service
```

終了

```
[root]# systemctl stop postgresql-11.service
```

10. PostgreSQLの自動起動設定。

設定

```
[root]# systemctl enable postgresql-11.service
```

設定解除

```
[root]# systemctl disable postgresql-11.service
```

- CentOS 7では自動起動を設定しないと、OSを起動した時に自動的にPostgreSQLが起動しない。

5. PostgreSQLの初期設定

5.1 概要

❖PostgreSQLインストール後に必要な初期設定を実施

- Linuxユーザとデータベースロールの設定作業
- PostgreSQLのパフォーマンスに関わる設定作業
- アクセス制御
  - 個人環境にPostgreSQLをインストールすることを想定。
  - PostgreSQLへの接続はローカルからの接続のみ許可。端末を直接操作してPostgreSQLを使用する環境を想定。

❖5章目次

5.2 Linuxユーザと データベースロールの設定

- **PostgreSQLを管理するために使用するLinuxユーザとPostgreSQLへの接続時に使用するデータベースロールの設定をそれぞれ実施**
  - Linuxユーザ「postgres」のパスワード設定
    - PostgreSQLサーバを管理するためのLinuxユーザ。
    - PostgreSQLインストール時に作成される。
    - 初期状態ではパスワードが設定されていない。
  - データベースロール「postgres」のパスワード設定
    - データベースロール：データベース接続時に使うユーザのようなもの。
    - データベースクラスタ作成時に、スーパーユーザ「postgres」が作成される。
    - 初期状態ではパスワードが設定されていない。
  - データベースロール「dbr」の作成
    - 一般権限のデータベースロール「dbr」を作成。
    - データベースへの接続、テーブルの作成、問い合わせ等に利用。
- **5.2節目次**
  - 5.2.1 Linuxユーザ「postgres」のパスワード設定
  - 5.2.2 データベースロール「postgres」のパスワード設定
  - 5.2.3 データベースロール「dbr」の作成

5.2.1 Linuxユーザ「postgres」のパスワード設定

❖【実習】ユーザ「postgres」のパスワードの設定

1. 「su」コマンドでルートユーザにスイッチ。

```
[lecture]$ su
パスワード:ledcb1912
```

2. 「passwd」コマンドでパスワードを設定。

```
[root]# passwd postgres
ユーザー postgres のパスワードを変更。
新しいパスワード:lecdb1912
新しいパスワードを再入力して下さい:lecdb1912
passwd: すべての認証トークンが正しく更新できました。
[root]# exit
```

3. 設定確認

```
[lecture]$ su postgres
パスワード:lecdb1912
bash-4.2$ exit
```

5.2.2 データベースロール「postgres」のパスワード設定

❖【実習】ロール「postgres」のパスワードの設定

1. 「psql」コマンドを使いロール「postgres」でデータベース「postgres」に接続。

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

- psql：PostgreSQLの対話型インターフェース。8章で詳しく説明。
- データベース「postgres」：データベースクラスタ作成時に作成されるデータベース。主にロール「postgres」でスーパーユーザ権限を要する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 'lecdb1912';
```

- 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「pg_hba.conf」が初期状態ではデータベースへのログインを無条件に許可しているため、現時点ではデータベースにログインしてもパスワードを求められない。

5.2.3 データベースロール「dbr」の作成

❖【実習】ロール「dbr」の作成

1. 「psql」コマンドを使いロール「postgres」でデータベース「postgres」に接続。

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

- 接続先データベースを省略した場合、接続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 'lecdb1912';
```

- 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のメタコマンド。

4. 「¥q」コマンドでpsqlを終了。

```
postgres=# ¥q
```

5.3 PostgreSQLサーバ制御ファイル（postgresql.conf）の設定

❖ postgresql.conf

- PostgreSQLサーバ制御用の設定パラメータを記述するファイル。
- 在り処：/var/lib/pgsql/11/data/postgresql.conf
- ファイルの所有者：postgres（Linuxユーザ）

```
[lecture]$ su postgres
パスワード:lecdb1912
bash-4.2$ cat /var/lib/pgsql/11/data/postgresql.conf
```

❖ 設定項目

- 設定項目は下の12項目に別れている。
  - ファイルの場所
  - 接続と認証
  - 資源の配分
  - WAL
  - レプリケーション
  - 問い合わせ計画
  - エラー報告とログ取得
  - 稼働統計情報
  - Auto vacuum
  - クライアント接続デフォルト
  - ロック管理
  - バージョン/プラットフォーム互換性

5.3.1 変更すべき項目

❖ 資源の配分（RESOURCE USAGE）

– デフォルトのメモリ使用量は保守的であり、現在のマシンでは本来の性能を発揮できない。以下の項目は値を変更すべき。

shared_buffers（デフォルト：128MB => 2048MB）

- 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行われる処理に影響。
- カーネルがキャッシュしたデータをPostgreSQLのshared_buffers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（デフォルト：4MB => 2048MB）

- 検索時にテーブル結合や並び替え操作を行うための領域。
- 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」程度とし、スワップが発生しないように設定する。

maintenance_work_mem（デフォルト：64MB => 4096MB）

- 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（デフォルト：4GB => そのまま）

- ディスクアクセス時のキャッシュヒット率を予想するために用いられる。
- 実メモリの50%程度が良いとされる。

5.3.2 postgresql.confの設定

❖【実習】postgresql.confの設定

1. 「su」コマンドでLinuxユーザ「postgres」にスイッチ。

```
[lecdb]$ su postgres
パスワード:lecdb1912
```

2. テキストエディタで「postgresql.conf」を開く。

```
bash-4.2$ emacs /var/lib/pgsql/11/data/postgresql.conf
```

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

```
shared_buffers = 2048MB
work_mem = 2048MB
maintenance_work_mem = 4096MB
```

4. ルートユーザでPostgreSQLを再起動。

```
bash-4.2$ exit
[lecture]$ su
パスワード:lecdb1912
[root]# systemctl restart postgresql-11.service
[root]# exit
```

5.4 アクセス制御ファイル（pg_hba.conf）の設定

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

- PostgreSQLサーバへのアクセスを制御するためのファイル
- 在り処：/var/lib/pgsql/11/data/pg_hba.conf
- ファイルの所有者：postgres（Linuxユーザ）

```
$[lecture] su postgres
パスワード:lecdb1912
bash-4.2$ cat /var/lib/pgsql/11/data/pg_hba.conf
```

❖ 設定方針

- ローカルからの接続に対し、パスワードによる認証を実施。
- 他の接続は許可しない（設定しない）。

5.4.1 pg_hba.confの書式

❖書式

タイプ	データベース	ロール	ネットワークアドレス	認証方式
local	all	all		trust

- タイプ
  - 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。
- データベース
  - データベース名：指定したデータベースのみアクセス可能。
  - all：全てのデータベースにアクセス可能。
- ロール
  - ロール名：指定したロールのみアクセス可能。
  - all：全てのロールがアクセス可能。
- ネットワークアドレス
  - 接続元のネットワークアドレスとネットマスクを記述。
  - タイプが「local」の場合は記述しない。
- 認証方式
  - 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

❖【実習】pg_hba.confの設定

1. 「su」コマンドでLinuxユーザ「postgres」にスイッチ。

```
[lecture]$ su postgres
パスワード:lecdb1912
```

2. テキストエディタで「pg_hba.conf」を開く。

```
bash-4.2$ emacs /var/lib/pgsql/11/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

# 参考(外部からのアクセスをパスワード付きで許可):
# host  all             all 0.0.0.0/0     md5
```

4. PostgreSQLを再起動。

```
bash-4.2$ exit
[lecture]$ su
パスワード:lecdb1912
[root]# systemctl restart postgresql-11.service
[root]# exit
```

5.4.3 pg_hba.confの設定確認

❖【実習】ロール「postgres」の接続確認

```
[lecture]$ psql -U postgres
ユーザー postgres のパスワード:lecdb1912
postgres=# ¥q
```

❖【実習】ロール「dbr」の接続確認

```
[lecture]$ psql -U dbr postgres
ユーザー dbr のパスワード:lecdb1912
postgres=> ¥q
```

- データベース名「postgres」を省略した場合はデータベース「dbr」への接続が試行されるが、データベース「dbr」は存在しないため接続に失敗する。

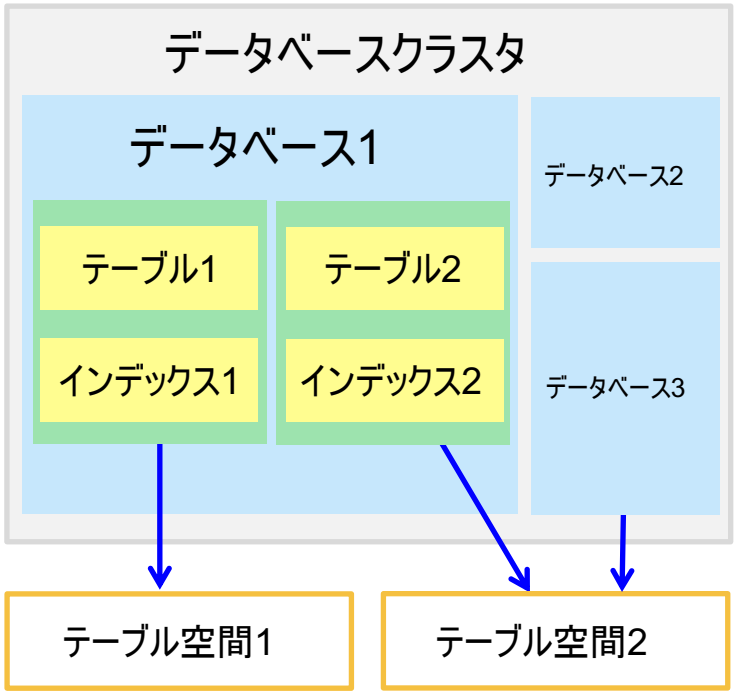
6. データベースの作成

6.1 概要

❖ データベースの作成とデータの登録を実施

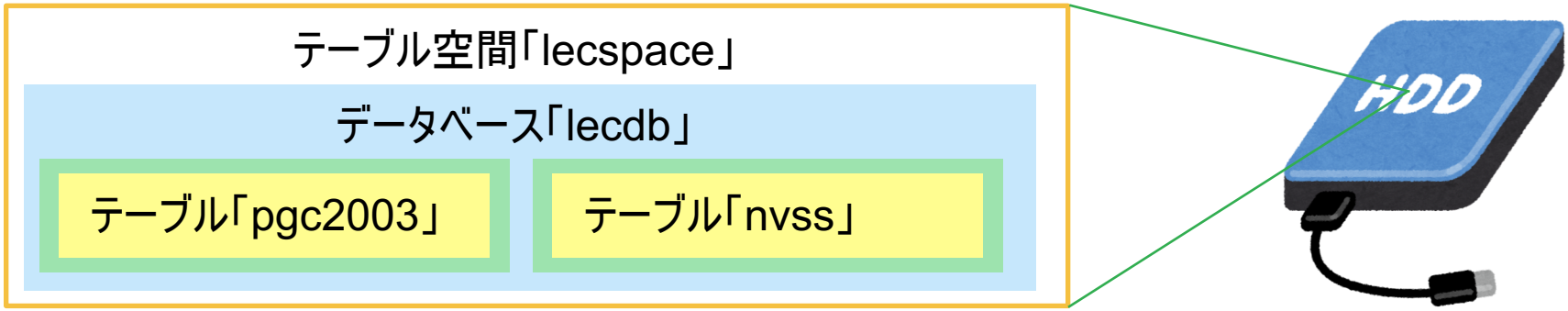
- 天体カタログ「PGC2003」と「NVSS」をデータベース化。
- 外付けHDD上にデータを収めることを想定し、データベースクラスタ外に作成したテーブル空間にデータを保存。

❖ データベースの構造



- データベースクラスタ
  - データベースの全情報が保存される領域。
- データベース
  - オブジェクト（テーブル、インデックス等）が格納される箱。
  - 物理的にはオブジェクトファイルが収まるディレクトリ。
  - データベースを跨いでのデータのアクセスは不可。
- テーブル
  - データが収まる表。
  - 物理的にはデータが収まるファイル郡。
- テーブル空間
  - データベースやテーブル等を保存できる領域。
  - データベースクラスタ外に作成できる。

❖作成するデータベースの構造



❖6章目次

- 6.1 概要
- 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

6.2 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;
```

- CREATE TABLESPACE
  - テーブル空間を作成するためのコマンド。
  - 実行にはスーパーユーザ権限が必要。
  - OWNER句でテーブル空間を所有するロールを指定可能。

❖【実習】テーブル空間の作成

- 「/home/data」にテーブル空間「lecspace」を作成。
- テーブル空間「lecspace」の所有者としてロール「dbr」を指定。

1. /home/dataディレクトリを作成。

```
[lecture]$ su
パスワード:lecdb1912
[root]# mkdir /home/data
```

- テーブル空間用ディレクトリは、Linuxユーザ「postgres」がアクセスできる場所に作成しなければならない。

2. dataディレクトリの所有者の変更。

```
[root]# chown postgres:postgres /home/data
[root]# exit
```

- テーブル空間用ディレクトリの所有者は、Linuxユーザ「postgres」である必要がある。

3. 「psql」コマンドを使いロール「postgres」でデータベース「postgres」に接続。

```
[lecture]$ psql -U postgres
パスワード:lecdb1912
```

- テーブル空間の作成にはスーパーユーザ権限を持つロールが必要。

4. テーブル空間を作成。

```
postgres=# CREATE TABLESPACE lecspace
           OWNER dbr
           LOCATION '/home/data';
```

5. 作成したテーブル空間を確認。

```
postgres=# ¥db
```

❖ テーブル空間の削除方法

```
postgres=# DROP TABLESPACE lecspace;
```

- テーブル空間の削除はスーパーユーザ権限を持つロールか、テーブルを所有するロールのみ実行可能。
- 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;
```

- CREATE DATABASE
  - データベースを作成するためのSQLコマンド。
  - 実行にはデータベース作成権限を持つロールが必要。
  - テーブル空間を指定すると、テーブル空間ディレクトリ下にデータベースディレクトリが作成される。

❖【実習】データベースの作成

- データベース「lecdb」をテーブル空間「lecspace」内に作成。
- データベース「lecdb」の所有者としてロール「dbr」を指定。

1. 「psql」コマンドを使いロール「postgres」でデータベース「postgres」に接続。

```
[lecture]$ psql -U postgres
パスワード:lecdb1912
```

- 「CREATE DATABASE」の実行にはデータベース作成権限を持つロールが必要。

2. データベースを作成。

```
postgres=# CREATE DATABASE lecdb
           OWNER dbr
           TABLESPACE lecspace;
```

3. 作成したデータベースを確認。

```
postgres=# ¥l
```

- 「/home/data」以下も確認してみましょう。

❖ データベースの削除方法

```
postgres=# DROP DATABASE lecdb;
```

- データベースの削除はスーパーユーザ権限を持つロールか、データベースを所有するロールのみ実行可能。

6.4 テーブルの作成

❖テーブルの作成を実施

- 天体カタログ「PGC2003」を収めるテーブル「pgc2003」の作成を実施。
- テーブル「pgc2003」へのデータの登録を実施。

❖6.4節目次

6.4.1 テーブルの構造

- **RDB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、データ型と異なるデータは入力できない。
    - 制約を定義したカラムには、制約に違反したデータは入力できない。
  - 行と列をRecordやFieldと言っている人もいるが、用語の定義が曖昧なので推奨しない。

			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

テーブルを作成するということは、カラムを定義すること。

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、文字データ型、日付・時刻データ型を紹介。

❖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型、INTEGER型、BIGINT型
  - データの桁数に応じて使い分ける。
- REAL型、DOUBLE PRECISION型
  - 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型かDECIMAL型を使用する。
- NUMERIC型とDECIMAL型
  - NUMERIC型とDECIMAL型は全く等価（歴史的な理由？）。
  - 精度（全体の桁数）と小数点以下の桁数を指定でき、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型
  - 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ギガバイトまで）。
  - TEXT型は標準SQLにはないが、多くのRDBMSがサポートしている。
- CHARACTER VARYING(N)型、VARCHAR(N）型
  - 最大n文字までの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長を確認する点だけが異なる。
  - テーブル定義からデータ量を見積もることができるのが利点か。
- CHARACTER(N)型、CHAR(N)型
  - 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い。使うメリットはない。

❖ 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)] [WITH TIME ZONE]	8バイト	hh:mm:ss[+/-hh]	1マイクロ秒
INTERVAL [FIELDS] [(P)]	16バイト	時間間隔	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型、DATE型、TIME型
  - それぞれ「年月日時分秒」、「年月日」、「時分秒」のデータを格納できる。
  - データの入力形式は ‘2019-01-01 00:00:00’ 等。
- INTERVAL
  - 各種時間単位（microsecond、millisecond、second、minute、hour、day、week、month、year、decade、century、millennium）を使って時間間隔データを格納できる。
  - 例えば「現在時刻-ある時間」など。
- WITH TIME ZONEオプション
  - postgresql.confで設定された時間帯を参照し、入力値をUTCに変換して格納する。
  - 時間帯が日本の場合’2019-01-01 00:00:00’は’2018-12-31 15:00:00’と格納される。
    - あるいは ‘2019-01-01 00:00:00+9’と時間帯を指定できる。
  - 出力時は’2019-01-01 00:00:00+9’と表示される。

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/lecture/DB/catalogue/pgc2003.dat
- データベース化のため、以下の加工を行っている。
  - 赤経を時角から度角に変換。
  - 赤緯を度分秒角から度角に変換。
  - 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 (
    カラム名1    データ型     [NOT NULL],
    カラム名2    データ型     [NOT NULL],
     ⋮
    [CONSTRAINT 主キー名 PRIMARY KEY (カラム名)]
);
```

- カラム名に大文字と「+*/-.':」は使用しない
  - 大文字を入力しても小文字として正規化される。
  - 二重引用符「""」で囲めば大文字も記号も使えるが、二重引用符付きのカラム名が出来てしまう。
- 制約（CONSTRAINT）
  - 列及びテーブルに対して課すことのできる制限。
  - 制約に違反したデータを格納しようとするとエラーを返し、データの誤入力を防止できる。
- 主キー制約（PRIMARY KEY）
  - テーブル内の各行を一意に識別できるカラムに対し設定できる制約。
  - 主キー制約は1つのカラムにしか設定できない。
  - 主キー制約を設定したカラムにはインデックスと非NULL制約が自動的に作成・設定される。
- 非NULL制約（NOT NULL）
  - NULL値を取らないカラムに対し設定できる制約。

❖【実習】テーブルの作成

- 天体カタログ「PGC2003」を収めるためのテーブルをデータベース「lecdb」に作成。
- 「CREATE TABLE」文を記述したファイルを作成し、作成したファイルを読み込んでSQLを実行する。

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

```
[lecture]$ emacs /home/lecture/DB/sql/ct_pgc2003.sql
```

2. テーブル作成用のSQLの記述。

```
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. 「psql」コマンドを使い、ロール「dbr」でデータベース「lecdb」に接続。

```
[lecture]$ psql -U dbr lecdb
パスワード:lecdb1912
```

4. 作成したファイル「ct_pgc2003.sql」を実行する。

```
lecdb=> ¥i /home/lecture/DB/sql/ct_pgc2003.sql
```

5. 作成したテーブルの確認。

```
lecdb=> ¥d
lecdb=> ¥d pgc2003
```

❖ テーブルの削除方法

```
lecdb=> DROP TABLE pgc2003;
```

- テーブルの削除はスーパーユーザ権限を持つロールか、テーブルを所有するロールのみ実行可能。

6.4.5 データの登録

❖¥copy FROMコマンドの書式

```
¥copy テーブル名 FROM ファイルのパス DELIMITER ‘区切り文字’ NULL
AS ‘NULL値とする文字’
```

– ¥copy
  - ファイルからテーブルへデータをコピーするメタコマンド。
  - 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（NULL）。

❖【実習】データの登録

1. 「psql」コマンドを使い、ロール「dbr」でデータベース「lecdb」に接続。

```
[lecture]$ psql -U dbr lecdb
パスワード:lecdb1912
```

2. 「/home/lecture/DB/catalogue/pgc2003.dat」をテーブル「pgc2003」にコピー。

```
lecdb=> ¥copy pgc2003 FROM /home/lecture/DB/catalogue/pg
c2003.dat DELIMITER ‘,’ NULL AS ‘’
```

- 983261行コピーされる

3. 「VACUUM ANALYZE」を実行。

```
lecdb=> VACUUM ANALYZE pgc2003;
```

- VACUUM ANALYZE：不要領域の回収と統計情報の更新を行うコマンド。
- PostgreSQLではテーブルからレコードの削除等を行っても、VACUUMを実行するまではテーブルからデータが物理的に削除されない。VACUUM ANALYZEを実行することで不要な領域の回収と統計情報の再計算を行い、回収した領域を再利用できるようにする。
- PostgreSQL8.1以降はAUTOVACUUM機能により定期的に実行される。

4. テーブルの確認。

```
lecdb=> SELECT * FROM pgc2003 LIMIT 10;
```

❖ データの削除方法

```
lecdb=> DELETE FROM pgc2003;
```

- データの削除はスーパーユーザ権限を持つロールか、テーブルを所有するロールのみ実行可能。

❖【コラム】「¥copy」と「COPY」

- 「¥copy」と同じ働きをする「COPY」というSQLコマンドが存在する。
- ¥copy
  - 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する。
- COPY
  - サーバ側にあるファイルをデータベースにコピーする。
  - サーバ側のファイルはLinuxユーザ「postgres」が読み込み可能でなければならない。
  - 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

❖インデックスの作成を実施

- RDBMSにおけるインデックスの役割りと仕組みを簡単に紹介。
- テーブル「pgc2003」のカラムにインデックスを設定。
- インデックスがない場合とある場合の検索速度の比較を実施。

❖6.5節目次

6.5.1 インデックスの仕組み

❖インデックス（索引）

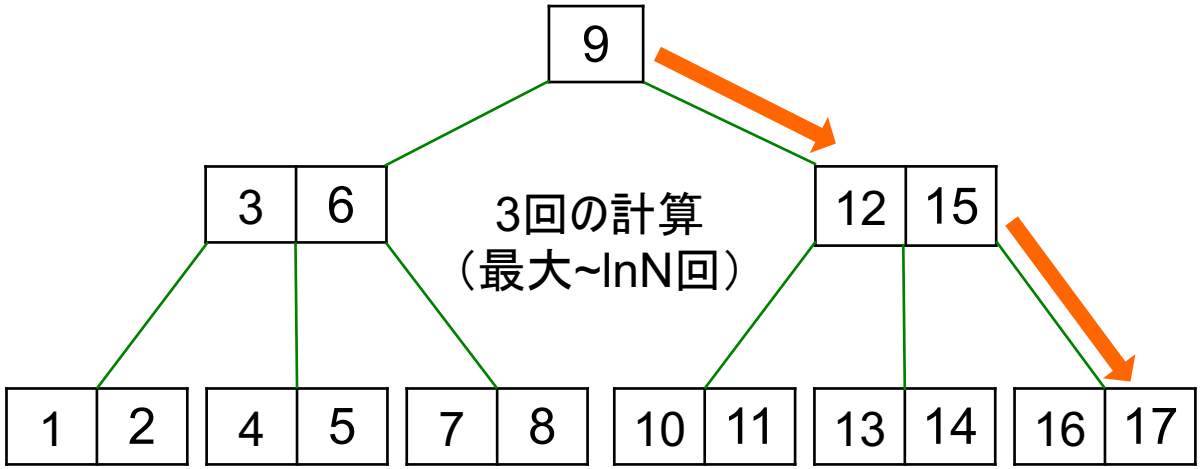
- カラムデータを予めツリー構造として展開し、目的のデータまでの検索手順を最小化するデータ構造とその仕組み。インデックスを作成することでRDBMSは本来の性能を発揮し、インデックスがない場合と比べ格段に高速の検索を行える。
- 主キーを設定したカラムには自動的にインデックスが作成されるが、その他のカラムは必要に応じて作成する必要がある。
- PostgreSQLではデフォルトでB-TREEインデックスという方式が使用される。

❖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であり、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17までのデータの中から16を検索する場合。

シーケンシャルスキャン

B-TREEインデックススキャン



- インデックススキャンが使用できない場合はシーケンシャルスキャンが行われる。

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(カラム名);
```

- CREATE INDEX
  - 指定したカラムにインデックスを設定するためのSQL文。
  - インデックス名を省略すると自動でインデックス名が割り当てられる。

❖【実習】インデックスがない場合の検索時間の測定

- テーブル「pgc2003」から赤緯±5度の領域内の天体を検索し、その検索時間を測定。

1. 「psql」コマンドを使い、ロール「dbr」でデータベース「lecdb」に接続。

```
[lecture]$ psql -U dbr lecdb
パスワード:lecdb1912
```

2. 「¥timing」コマンドを実行し、実行時間が計測されるようにする。

```
lecdb=> ¥timing
タイミングは on です。
```

3. 以下の検索を3回以上行い、最後の3回の実行時間の中間値をメモする。

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

平均時間	ms

❖【実習】インデックスの設定

– テーブル「pgc2003」のカラム「ra」と「dc」にインデックスを設定。

1. インデックスを作成。

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

2. インデックスの確認。

```
lecdb=> ¥d pgc2003
```

3. 「VACUUM ANALYZE」の実行。

```
lecdb=> VACUUM ANALYZE pgc2003;
```

- 実行しないと即座にインデックスが反映されない。

4. 以下の検索を3回以上行い、最後の3回の実行時間の中間値をメモする。

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

平均時間	ms

5. 両者の実行時間を比較する。

❖【実習】インデックスの追加作成

– テーブル「pgc2003」の残りのカラムにインデックスを設定。

```
lecdb=> CREATE INDEX ON pgc2003(hoge);
```

- P.62のPGC2003諸元を参照してインデックスを設定しましょう。
- 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カラムのみに設定するのが正しい。

演習 NVSSカタログのDB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/lecture/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。
  - 赤経を時角から度角に変換。
  - 赤緯を度分秒角から度角に変換。
  - 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）。
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	deg	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	deg	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カタログを収めるテーブル「nvss」を作成しデータを登録せよ。

1. 【ct_nvss.sql】データベース「lecdb」上にNVSSカタログを格納するためのテーブル「nvss」を作成せよ。
   - カラム「id」に主キー制約を設定する。
   - 各カラムに適切にNOT NULL制約を設定する。
2. テーブル「nvss」にNVSSカタログのデータをコピーせよ。
   - データの在り処：/home/lecture/DB/catalogue/nvss.dat
3. テーブル「nvss」の全カラムにインデックスを設定せよ。

❖【解答例】NVSSカタログを収めるテーブル「nvss」を作成しカタログを登録せよ。

1. データベース「lecdb」上にNVSSカタログを格納するためのテーブル「nvss」を作成せよ。

```
[lecture]$ emacs /home/lecture/DB/sql/ct_nvss.sql
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カタログのデータをコピーせよ。

```
lecdb=> ¥copy nvss FROM /home/lecture/DB/catalogue/nvss.
dat DELIMITER ‘,’ NULL AS ‘’
```

- 1773484行コピーされる

3. テーブル「nvss」の全カラムにインデックスを設定せよ。

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

- ¥dでインデックスが作成できたか確認する。

❖【コラム】PostgreSQLにおけるデータの操作コマンド

- PostgreSQLには「¥copy (COPY) FROM」、「INSERT」、「UPDATE」という3つのデータ操作コマンドが存在する。
- ¥copy (COPY) FROM
  - 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データベース化するのに向いている。
  - ファイルからしかコピーできない。
- INSERT
  - テーブルにデータを追加するためのコマンド。
  - 既存のテーブルのデータを追加することができる。
  - 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するためのコマンド。

7. テーブルへの問い合わせ

7.1 概要

- SELECT文を使ってPostgreSQLサーバ上のテーブルへの問い合わせを行う方法を紹介
- 7章目次
  - 7.1 概要
  - 7.2 SELECT文の基本形
  - 7.3 検索条件の指定
  - 7.4 カラムに対する演算
  - 7.5 副問い合わせ
  - 7.6 テーブルの結合
  - 7.7 遅くならないWHERE句の書き方

7.2 SELECT文の基本形

❖SELECT文

- SELECT文はテーブルから行を検索し、結果を表示するSQL。
- 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を行うことができる。

❖7.2節目次

- 7.2.1 基本形
- 7.2.2 検索結果の出力件数の指定
- 7.2.3 検索結果の並び替え
- 7.2.4 カラム・テーブルの別名の定義

7.2.1 基本形

❖SELECT文の書式

```
SELECT カラム名
FROM テーブル名;
```

- カラムの指定
  - 複数のカラムを指定する場合はカラム名とカラム名をカンマ「,」で区切る。
  - 全カラムを指定する場合はワイルドカード「*」を指定する。
- 検索結果の操作
  - 「Enter」キーで1行、「Space」キーでページ単位で送る。
  - 「q」キーで検索結果の表示を終了。

本講習会ではSQLコマンドは大文字、カラム名とテーブル名は小文字で書きます。入力時は小文字でok。

❖実行例

- テーブル「pgc2003」からカラム「id」、「ra」、「dc」を検索。

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

- テーブル「pgc2003」の全列を検索。

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

- 「¥x」コマンドで表示形式を切り替え可能。

7.2.2 検索結果の出力件数の指定

❖LIMIT句の書式

```
SELECT カラム名
FROM テーブル名
LIMIT  出力件数;
```

- LIMIT句
  - 指定された件数のみ表示する。
  - SELECT文の末尾に記述しなければならない。

❖実行例

- 検索結果の先頭20件を出力。

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

7.2.3 検索結果の並び替え

❖ORDER BY句の書式

```
SELECT カラム名
FROM テーブル名
ORDER BY カラム名 [ASC|DESC];
```

- 昇順で並び替え
  - ASC句を使うと指定したカラムの値が小さい順に検索結果を表示。
  - デフォルト値であるため「ASC」を省略した場合は昇順並び替えとなる。
- 降順で並び替え
  - DESC句を使うと指定したカラムの値が大きい順に検索結果を表示。

❖実行例

- 検索結果をIDが小さい順に並び替えて出力。

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

7.2.4 カラム・テーブルの別名の定義

❖別名の定義方法

```
SELECT カラム名 AS 別名
FROM テーブル名 AS 別名;
```

- 別名について
  - カラムに対して定義した別名は、SELECT文の処理の関係上WHERE句では参照できない。ただし、ORDER BY句とGROUP BY句では参照できる。

❖実行例

- テーブル「pgc2003」とカラム「id」、「ra」、「dc」に別名を定義した場合。

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

- 「テーブル名.カラム名」はカラムの厳密な指定の仕方。
- テーブルに別名を定義した場合は、カラム名を厳密に指定しなければならない場合が多い。

演習 SELECT文の基本形

❖【演習】以下の問い合わせを実施せよ。

1. 【s72ex1.sql】テーブル「pgc2003」のカラム「id」、「ra」、「dc」を、カラム「id」が小さい順に20件表示せよ。
   - ディレクトリ「/home/lecture/DB/sql/」以下にファイル「S72EX1.sql」を作成し、コマンド「¥i」で読み込みましょう。
2. 【s72ex2.sql】テーブル「pgc2003」のカラム「id」、「ra」、「dc」にそれぞれ「pgc」、「sekkei」、「sekii」という別名を与え、カラム「sekii」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. 【s72ex1.sql】テーブル「pgc2003」のカラム「id」、「ra」、「dc」を、カラム「id」が小さい順に20件表示せよ。

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

2. 【s72ex2.sql】テーブル「pgc2003」のカラム「id」、「ra」、「dc」にそれぞれ「pgc」、「sekkei」、「sekii」という別名を与え、カラム「sekii」が大きい順に20件表示せよ。

```
SELECT id AS pgc,
       ra AS sekkei,
       dc AS sekii
FROM pgc2003
ORDER BY sekii DESC
LIMIT 20;
```

7.3 検索条件の指定

❖ WHERE句

- SELECT文中でWHERE句を使って条件式を設定することで、条件式を満たすレコードのみを返すことができる。
- WHERE句内で使われる基本的な演算子と句を紹介。

```
SELECT カラム名
FROM テーブル名
WHERE 条件式;
```

❖ 7.3節目次

7.3.1 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

– IDがPGC0077777の天体を検索。

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

7.3.2 IS演算子

❖IS演算子の書式

```
SELECT カラム名
FROM テーブル名
WHERE カラム名 IS [NOT] NULL;
```

- IS [NOT] NULL
  - NULL値を含む（含まない）レコードを検索する際に利用する。
  - 通常の比較演算子（「=」や「!=」）でNULL値の比較演算を行うと真や偽ではなく「NULL」を返すため、NULL値を検索することができない。

❖実行例

- カラム「mtype」がNULLであるレコードを、比較演算子「=」を使って検索。

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

- カラム「mtype」がNULLであるレコードを、IS演算子を使って検索。

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

7.3.3 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

– 赤緯が-88度以下か+88度以上かつ「mtype」の値がNULLではない天体を検索。

```
lecdb=> SELECT id,ra,dc,mtype
        FROM pgc2003
        WHERE (dc<=-88 OR dc>=88) AND
              (mtype IS NOT NULL)
        ORDER BY dc;
```

7.3.4 BETWEEN句

❖BETWEEN句の書式

```
SELECT カラム名
FROM テーブル名
WHERE カラム名 [NOT] BETWEEN a AND b;
```

- BETWEEN句
  - カラムの値がa以上からb以下の間にあるレコードを返す。

❖実行例

- 赤緯が-0.1度以上から+0.1度以下までの天体を検索。

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

7.3.5 パターンマッチング

❖LIKE式とSIMILAR TO式の書式

```
SELECT カラム名
FROM テーブル名
WHERE カラム名 [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。	×	○

❖実行例

– 「NGC5194」を検索。

```
lecdb=> SELECT id,ra,dc,anames
        FROM pgc2003
        WHERE anames LIKE ‘%NGC5194%’;
```

演習 検索条件の指定

❖【演習】以下の問い合わせを実施せよ。

1. 【s73ex1.sql】テーブル「pgc2003」から赤経が359度から1度かつ赤緯が-1度から+1度の範囲にある天体を検索し、カラム「id」、「ra」、「dc」を赤緯が小さい順に表示せよ。
2. 【s73ex2.sql】テーブル「pgc2003」から「logd25」が3以上である天体を検索し、カラム「id」、「ra」、「dc」、「logd25」、「anames」を「logd25」が大きい順に表示せよ。

❖【解答例】以下の問い合わせを実施せよ。

1. 【s73ex1.sql】テーブル「pgc2003」から赤経が359度から1度かつ赤緯が-1度から+1度の範囲にある天体を検索し、カラム「id」、「ra」、「dc」を赤緯が小さい順に表示せよ。

```
SELECT id,ra,dc
FROM pgc2003
WHERE (ra>=359 OR ra<=1) AND (dc BETWEEN -1 AND 1)
ORDER BY dc;
```

2. 【s73ex2.sql】テーブル「pgc2003」から「logd25」が3以上である天体を検索し、カラム「id」、「ra」、「dc」、「logd25」、「anames」を「logd25」が大きい順に表示せよ。

```
SELECT id,ra,dc,logd25,anames
FROM pgc2003
WHERE (logd25>=3)
ORDER BY logd25 DESC;
```

7.4 カラムに対する演算

❖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文中でカラム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

- 7.4.1 算術演算子
- 7.4.2 文字列演算子
- 7.4.3 型変換関数
- 7.4.4 算術関数
- 7.4.5 集約・統計関数

7.4.1 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

– 整数と整数の割り算。

```
lecdb=> SELECT 10/3;
```

– 整数と小数の割り算。

```
lecdb=> SELECT 10/3.0;
```

- 異なるデータ型同士の演算時は暗黙的な型変換が行われる。

– カラム「logd25」のlogをはずして単位「arcmin」で表示。

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

- 演算結果にカラム名を付与しないと、カラム名が「?column?」となってしまう。

7.4.2 文字列演算子

❖文字列演算子一覧

演算子	説明	例	結果
||	文字列の結合	hoge || hoge	hogehoge

❖実行例

– カラム「id」に文字列「hoge」を結合。

```
lecdb=> SELECT 'hoge' || id AS hogehoge
        FROM pgc2003
        LIMIT 20;
```

7.4.3 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。	1.2 :: INTEGER	1

- CAST関数
  - 明示的にあるデータのデータ型を他のデータ型に変換することができる。

❖実行例

- 整数とdouble precision型に型変換した整数の割り算。

```
lecdb=> SELECT 10/CAST(3 AS DOUBLE PRECISION);
```

- カラム「polflux」が999.9である天体を検索。

```
lecdb=> SELECT id,ra,dc,polflux
        FROM nvss
        WHERE polflux=CAST(999.9 AS REAL);
```

- WHERE polflux=999.9も試してみましょう。
- SELECT pg_typeof(999.9);を確認してみましょう。

7.4.4 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	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。

```
lecdb=> SELECT id,
               TRUNC(CAST(ra AS NUMERIC),0) AS nra,
               TRUNC(CAST(dc AS NUMERIC),0) AS ndc
        FROM pgc2003
        LIMIT 20;
```

- TRUNC(V,S)関数のVの入力データ型はNUMERIC型である必要がある。カラム「ra」と「dc」のデータ型はREAL型であるため、CAST関数でNUMERICに変更する必要がある。

– 赤経(度)と赤緯(度)を赤経赤緯(時,分,秒,度,分,秒)に変換。

```
lecdb=> SELECT id,
               TRUNC(CAST((ra/15.0) AS NUMERIC),0) AS hh,
               TRUNC(MOD(CAST((ra/15.0) AS NUMERIC),1)*60,0) AS mm,
               MOD(MOD(CAST((ra/15.0) AS NUMERIC),1)*60,1)*60 AS ss,
               TRUNC(CAST(dc AS NUMERIC),0) AS dd,
               ABS(TRUNC(MOD(CAST(dc AS NUMERIC),1)*60,0)) AS mm,
               ABS(MOD(MOD(CAST(dc AS NUMERIC),1)*60,1)*60) AS ss
        FROM pgc2003
        WHERE id='PGC0077777';

    id      | hh | mm |        ss         | dd  | mm |      ss
------------+----+----+-------------------+-----+----+------------
PGC0077777 |  7 | 37 | 4.99991999998800 | -11 | 50 | 54.999600
```

- 赤経(時):赤経(度)を15度で割って整数部を取り出す。
- 赤経(分):赤経(度)を15度で割った値の小数部を60倍し整数部を取り出す。
- 赤経(秒):赤経(度)を15度で割った値の小数部を60倍した値の小数部を60倍する。

7.4.5 集約・統計関数

❖統計関数一覧

関数	引数型	戻り値型	例
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。

- 集約・統計関数
  - 複数の入力値から1つの結果を返す関数。
  - 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。

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

- 「logd25」の最大値を検索。

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

- 「logd25」の最大値がどのIDなのか知りたいとき（間違い例）。

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

❖GROUP BY句とHAVING句の書式

```
SELECT カラム名 FROM テーブル名
GROUP BY カラム名
HAVING 条件式;
```

– GROUP BY句とHAVING句
  - GROUP BY句：同じ値を持つ複数のレコードをグループ化する。
  - HAVING句：グループ化した結果の中から条件式を満たすレコードを返す。

❖実行例

– 「mtype」毎にグループ化して[「SB%」タイプの]天体数を表示。

```
lecdb=> SELECT mtype,count(*)
        FROM pgc2003
        GROUP BY mtype
        [HAVING mtype LIKE 'SB%'];
```

– 赤経10度ごとにグループ化して、「mtype」が明らかになっている天体の天体数を表示。

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

演習 カラムに対する演算

❖【演習】以下の問い合わせを実行せよ。

1. 【s74ex1.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. 【s74ex2.sql】テーブル「nvss」から「flux」が1000mJy以上の電波源を検索し、それぞれの偏波率を表示せよ。偏波率は小数第一位で四捨五入すること。
   - 偏波率：fpol=(polflux/flux)*100
   - 四捨五入：ROUND(V,S)

```
SELECT id,ra,dc,flux,polflux,OOO
FROM nvss
WHERE (flux>=1000) AND (polflux IS NOT NULL)
ORDER BY fpol DESC
LIMIT 20;
```

❖【解答例】以下の問い合わせを実施せよ。

1. 【s74ex1.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。

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

2. 【s74ex2.sql】テーブル「nvss」から「flux」が1000mJy以上の電波源を検索し、それぞれの偏波率を表示せよ。偏波率は小数第一位で四捨五入すること。

```
SELECT id,ra,dc,flux,polflux,
       ROUND(
         CAST((polflux/flux)*100 AS NUMERIC),1
       ) AS fpol
FROM nvss
WHERE (flux>=1000) AND (polflux IS NOT NULL)
ORDER BY fpol DESC
LIMIT 20;
```

❖【コラム】SELECT文の評価順序

– SELECT文は以下の順序で処理が行われる。

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

```
SELECT TRUNC(CAST(ra AS NUMERIC),-1) AS sekkei,COUNT(*)
FROM pgc2003
WHERE mtype IS NOT NULL
GROUP BY sekkei
ORDER BY sekkei;
```

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

- PostgreSQLではGROUP BY句でSELECT句の別名を参照できる。一方WHERE句とHAVING句では別名を参照できない。

7.5 副問い合わせ

❖副問い合わせ

- SELECT句での副問い合わせ
  - 問い合わせの結果に副問い合わせの結果を結合することができる。
- FROM句で副問い合わせ
  - 副問い合わせの結果に対して問い合わせを行うことができる。
- WHERE句で副問い合わせ
  - 副問い合わせの結果を基に評価を行うことができる。

❖7.5節目次

7.5.1 SELECT句での副問い合わせ

❖SELECT句での副問い合わせ書式

```
SELECT (
  SELECT カラム名
  FROM テーブル名
  WHERE 条件式
) AS 別名
FROM テーブル名;
```

– SELECT句での副問い合わせについて
  • SELECT句中での副問い合わせの結果は、一行一列の値でなければならない。

❖実行例

– 電波源「133108+303032」のFluxが全電波源の平均Fluxよりも大きいか検査。

```
lecdb=> SELECT id,ra,dc,flux,flux > (
          SELECT AVG(flux)
          FROM nvss
        ) AS hantei
        FROM nvss
        WHERE id='133108+303032';
```

7.5.2 FROM句での副問い合わせ

❖FROM句での副問い合わせ書式

```
SELECT 別名.カラム名
FROM (
   SELECT カラム名
   FROM テーブル名
   WHERE 条件式
) AS 別名;
```

– FROM句での副問い合わせについて
  - 副問い合わせの検索結果には別名を指定しなければならない。
  - 主SELECT文ではカラム名を「別名.カラム名」で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行われない。

❖実行例

– 赤緯が±0.1度内の天体を検索し、その中から「mtype」が「SB%」の天体を検索。

```
lecdb=> SELECT new.id,new.mtype
        FROM (
          SELECT *
          FROM pgc2003
          WHERE dc BETWEEN -0.1 and 0.1
        ) AS new
        WHERE new.mtype LIKE 'SB%' LIMIT 20;
```

7.5.3 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 カラム名
FROM テーブル名
WHERE カラム名 比較演算子 (
   SELECT カラム名
   FROM テーブル名
   WHERE 条件式
);
```

- 一行一列の値を返す副問い合わせについて
  - 副問い合わせが単一の値を主SELECT文に返す。
  - WHERE句は論理・比較演算子と副問い合わせで構成される。

❖実行例

– 「logd25」の最大値がどのIDなのか知りたいとき。

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

– 電波源「133108+303032」よりもFluxが大きい天体を検索。

```
lecdb=> SELECT id,ra,dc,flux
        FROM nvss
        WHERE flux >= (
          SELECT flux
          FROM nvss
          WHERE id='133108+303032'
        )
        ORDER BY flux
        LIMIT 20;
```

❖複数行複数列の値を返す副問い合わせ書式

```
SELECT カラム名
FROM テーブル名
WHERE カラム名 [NOT] IN (
   SELECT カラム名
   FROM テーブル名
   WHERE 条件式
);
```

- 複数行複数列の値を返す副問い合わせについて
  - 副問い合わせが複数の値を主SELECT文に返す。
  - IN:副問い合わせが返したそれぞれの値が、指定したカラムに存在するか評価。
  - WHERE句内で(カラム1,カラム2)とすることでカラムを複数指定できる。

❖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。

```
lecdb=> 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 カラム名
FROM テーブル名
WHERE [NOT] EXISTS (
  SELECT カラム名
  FROM テーブル名
  WHERE 条件式
);
```

- 相関副問い合わせについて
  - 副問い合わせの結果が真である場合のみ主SELECT文の結果を返す問い合わせ。
  - 主問い合わせのテーブルを副問い合わせで参照するため相関問い合わせと呼ばれる。

❖実行例

- テーブル「pgc2003」の赤経赤緯の値がテーブル「nvss」中に存在する場合のみ結果を返す。

```
lecdb=> 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中で評価し、真の場合結果を返している。

7.5.4 共通テーブル式

❖WITH句の書式

```
WITH 別名1 AS (
   SELECT カラム名
   FROM テーブル名
   WHERE 条件式
) [,別名2 AS ・・・];
```

– 共通テーブル式
  • WITH句に記述した問い合わせの検索結果を一時テーブルとして保持できる。

❖実行例

– 赤緯が±1度内の天体を検索し、その中から「mtype」が「SB%」の天体を検索。

```
lecdb=> WITH new AS (
           SELECT *
           FROM pgc2003
           WHERE dc BETWEEN -1 and 1
         )
         SELECT new.id,new.mtype
         FROM new
         WHERE mtype LIKE 'SB%' LIMIT 20;
```

演習 副問い合わせ

❖【演習】以下の問い合わせを実行せよ。

1. 【s75ex1.sql】FROM句での副問い合わせを使ってテーブル「nvss」から赤経359度から1度の領域にある電波源を検索し、その中から「flux」が1000mJy以上の電波源を検索せよ。

```
SELECT new.id,new.flux FROM ...;
```

2. 【s75ex2.sql】上記の問い合わせををWITH句を使った形式に変形して実行せよ。

```
WITH new AS (...) SELECT ...;
```

❖【解答例】以下の問い合わせを実施せよ。

1. 【s75ex1.sql】FROM句での副問い合わせを使ってテーブル「nvss」から赤経359度から1度の領域にある電波源を検索し、その中から「flux」が1000mJy以上の電波源を検索せよ。

```
SELECT new.id,new.flux
FROM (
  SELECT *
  FROM nvss
  WHERE (ra>=359 OR ra<=1)
) AS new
WHERE new.flux>=1000;
```

2. 【s75ex2.sql】WITH句を使ってEx7-5-1を実行せよ。

```
WITH new AS (
  SELECT *
  FROM nvss
  WHERE (ra>=359 OR ra<=1)
)
SELECT new.id,new.flux
FROM new
WHERE new.flux>=1000;
```

7.6 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を行うことができる。
- 「交差結合」、「内部結合」、「外部結合」という結合方法が存在する。

❖7.6節目次

7.6.1 交差結合

❖交差結合の書式

```
SELECT カラム
FROM テーブル1 CROSS JOIN テーブル2;
```

- 交差結合（CROSS JOIN）
  - テーブル1の1レコード毎にテーブル2の全レコードを結合する。即ち全ての組み合わせを求める。
  - クロス結合で得られるテーブルの行数は、テーブル1とテーブル2のレコードの積。

❖図解

- 主食テーブルとおかずテーブルを交差結合した場合

主食テーブル

ID	主食	種別
1	ごはん	和
2	パン	洋
3	炒飯	中

X

おかずテーブル

ID	おかず	種別
1	納豆	和
2	ハム	洋

=

交差結合結果

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」の赤経赤緯の値が完全に一致するレコードを交差結合を使って検索。

```
lecdb=> 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;
```

7.6.2 内部結合

❖内部結合の書式

```
SELECT カラム
FROM テーブル1 INNER JOIN テーブル2 ON 結合条件;
```

- 内部結合（INNER JOIN）
  - テーブル1の各レコードに対して、結合条件を満たすテーブル2の各行を結合する。
  - 結合条件の例としては「テーブル1.id = テーブル2.id」などが挙げられる。

❖図解

- 主食テーブルとおかずテーブルを「主食テーブル.種別=おかずテーブル.種別」という条件で内部結合した場合

主食テーブル

ID	主食	種別
1	ごはん	和
2	パン	洋
3	炒飯	中

X

おかずテーブル

ID	おかず	種別
1	納豆	和
2	ハム	洋

=

内部結合結果

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

– テーブル「pgc2003」と「nvss」の赤経赤緯の値が完全に一致するレコードを内部結合を使って検索。

```
lecdb=> 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;
```

– 3個以上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;
```

❖内部結合の書式(簡易系)

```
SELECT カラム
FROM テーブル1,テーブル2
WHERE 結合条件;
```

- 内部結合(簡易系)
  - 簡易系では3つ以上のテーブルを結合することができない。

❖実行例

- テーブル「pgc2003」と「nvss」の赤経赤緯の値が完全に一致するレコードを内部結合を使って検索。

```
lecdb=> 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;
```

7.6.3 外部結合

❖外部結合の書式

```
SELECT カラム
FROM テーブル1 LEFT|RIGHT|FULL OUTER JOIN テーブル2
ON 結合条件;
```

- 外部結合
  - 結合対象のテーブルに結合条件を満たすレコードが存在しない場合も結合を行う。
- 左外部結合（LEFT OUTER JOIN）
  - LEFT OUTER JOIN句の左側のテーブルのレコードは全て表示し、右側のテーブルのレコードは結合条件を満たしたもののみを表示する。それ以外はNULL値を返す。
- 右外部結合（RIGHT OUTER JOIN）
  - LEFT INNER JOINと逆の処理を行う。
- 完全外部結合（FULL OUTER JOIN）
  - 左右のテーブルを全て表示する。
  - 結合条件を満たすレコードは結合を行い、満たさないレコードはNULL値を返す。

❖LEFT OUTER JOIN 図解

– 主食テーブルとおかずテーブルを「主食テーブル.種別=おかずテーブル.種別」という条件で左外部結合した場合

主食テーブル

ID	主食	種別
1	ごはん	和
2	パン	洋
3	焼飯	中

X

おかずテーブル

ID	おかず	種別
1	納豆	和
2	ハム	洋

=

左外部結合結果

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			

❖RIGHT OUTER JOIN 図解

– 主食テーブルとおかずテーブルを「主食テーブル.種別=おかずテーブル.種別」という条件で右外部結合した場合

主食テーブル

ID	主食	種別
1	ごはん	和
2	パン	洋
3	焼飯	中

X

おかずテーブル

ID	おかず	種別
1	納豆	和
2	ハム	洋

=

左外部結合結果

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

❖FULL OUTER JOIN 図解

– 主食テーブルとおかずテーブルを「主食テーブル.種別=おかずテーブル.種別」という条件で完全外部結合した場合

主食テーブル

ID	主食	種別
1	ごはん	和
2	焼飯	中

X

おかずテーブル

ID	おかず	種別
1	納豆	和
2	ハム	洋

=

完全外部結合結果

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

❖実行例

– テーブル「pgc2003」と「nvss」の赤経赤緯の値が完全に一致するレコードを左外部結合を使って検索。

```
lecdb=> SELECT a.id,
               a.ra,
               a.dc,
               b.id,
               b.ra,
               b.dc
        FROM pgc2003 AS a LEFT OUTER JOIN nvss AS b
             ON (a.ra=b.ra AND
                 a.dc=b.dc
             );
```

演習 テーブルの結合

❖【演習】以下の問い合わせを実行せよ。

1. 【s76ex1.sql】テーブル「pgc2003」と「nvss」を「赤経赤緯の第三位までが完全に一致する」という条件で内部結合せよ。

```
SELECT a.id,
       a.ra,
       a.dc,
       b.id,
       b.ra,
       b.dc
FROM ...
     ON (
       TRUNC(CAST(a.ra AS NUMERIC),3)=
       TRUNC(CAST(b.ra AS NUMERIC),3) AND
       TRUNC(CAST(a.dc AS NUMERIC),3)=
       TRUNC(CAST(b.dc AS NUMERIC),3)
     )
WHERE a.dc BETWEEN -1 AND 1;
```

❖【解答例】以下の問い合わせを実施せよ。

1. 【s76ex1.sql】テーブル「pgc2003」と「nvss」を「赤経赤緯の第三位までが完全に一致する」という条件で内部結合せよ。

```
SELECT a.id,
       a.ra,
       a.dc,
       b.id,
       b.ra,
       b.dc
FROM pgc2003 AS a INNER JOIN nvss AS b
     ON (TRUNC(CAST(a.ra AS numeric),3)=
         TRUNC(CAST(b.ra AS numeric),3) AND
         TRUNC(CAST(a.dc AS numeric),3)=
         TRUNC(CAST(b.dc AS numeric),3)
     )
WHERE a.dc BETWEEN -1 AND 1;
```

- なぜ195件の検索に~4.5秒も時間がかかるのか？

7.7 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。
    1. WHERE句の書き方が悪い。
    2. インデックスを張っていないカラムに対して検索している。
    3. 検索結果が膨大で、ディスクI/Oがボトルネックとなっている。
  - 質の悪いSQL文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。

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

1. 以下のSQL文は「logd25」が1 arcmin 以上の天体の数を計測するものである。SQL文を3回以上実行し、最後の3回の中間値をメモせよ。

– A

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

平均時間	ms

– B

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

平均時間	ms

2. 「SELECT」の直前に「EXPLAIN ANALYZE」をつけて前述のSQL文を実行せよ。

– A

```
lecdb=> EXPLAIN ANALYZE
        SELECT COUNT(*)
        FROM pgc2003
        WHERE (10^logd25)*0.1 >= 1;
```

– B

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

– EXPLAIN
  - 問い合わせの実行計画を表示するSQLコマンド。
  - 「ANALYZE」は問い合わせを実行し、実際の実行時間を表示するオプション。

– 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。

– カラムに対する演算
  - WHERE句内でカラムに対する演算を行うと、PostgreSQLはカラムの演算結果に対して評価を行う。即ちインデックス検索が行われない。
  - WHERE句内でのカラムに対する演算は厳禁。

❖s76ex1の高速化

– s76ex1は演算結果を条件として結合を行っているため遅い。

```
SELECT a.id,
       a.ra,
       a.dc,
       b.id,
       b.ra,
       b.dc
FROM pgc2003 AS a INNER JOIN nvss AS b
     ON (TRUNC(CAST(a.ra AS numeric),3)=
         TRUNC(CAST(b.ra AS numeric),3) AND
         TRUNC(CAST(a.dc AS numeric),3)=
         TRUNC(CAST(b.dc AS numeric),3)
     )
WHERE a.dc BETWEEN -1 and 1;
```

– 解決方法

1. 絞り込み検索
   - 利用者はSELECT文を工夫するしか術がない。
2. 式インデックス
   - あるテーブルのある列に対する計算結果に対してインデックスを設定する方法。
   - テーブルが更新されるたびに式が実行されインデックスが更新される。
3. あらかじめ計算結果を表に格納しインデックスを設定
   - （例えば銀経銀緯等の）頻繁に使われる値は計算しておいた方が効率的。

– 【s76ex1a.sql】絞り込み検索の実施

```
SELECT a.id,
       a.ra,
       a.dc,
       b.id,
       b.ra,
       b.dc
FROM ( SELECT *
       FROM pgc2003
       WHERE dc BETWEEN -1 and 1
     ) AS a INNER JOIN (
       SELECT *
       FROM nvss
       WHERE dc BETWEEN -1 and 1
     ) AS 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)
     )
;
```

- 副問い合わせを使って、結合を行う前に母数を減らしている。
- 実行時間：~0.27秒（~16倍高速化）！

❖【コラム】「*」の多様に注意

- 以下のSQLはs76ex1のFROM句を副問い合わせの形式に変形したものである。
- 新たに定義されたテーブル「t」には同名の列が含まれるため、FROM句内での検索時に新たな名前を定義しないとSQLが動作しない。

```
SELECT a.id,
       a.ra,
       a.dc,
       b.id,
       b.ra,
       b.dc
FROM (SELECT *
      FROM pgc2003 AS a INNER JOIN nvss AS b
           ON (TRUNC(CAST(a.ra AS numeric),3)=
               TRUNC(CAST(b.ra AS numeric),3) AND
               TRUNC(CAST(a.dc AS numeric),3)=
               TRUNC(CAST(b.dc AS numeric),3)
           )
     ) AS t
WHERE a.dc BETWEEN -1 and 1;
```

「t.hoge」にしないと検索できない。

「a.id AS aid, b.id AS bid」等を指定する必要あり。

8. psqlの使用方法

8.1 概要

❖psqlの使い方を紹介

- psqlはPostgreSQLの標準ターミナルであり、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する。
- 「psql」コマンドの基本的な使い方やSQLコマンドの実行方法、psql自身が実行するメタコマンドを紹介。

❖8章目次

- 8.1 概要
- 8.2 「psql」コマンドの基本的な使い方
- 8.3 SQLコマンド
- 8.4 メタコマンド
- 8.5 ログインパスワードの入力省略設定

8.2「psql」コマンドの基本的な使い方

❖「psql」コマンドの書式

```
$ psql [オプション] データベース名
```

- 接続関連オプション
  - -h ホスト名：接続先PostgreSQLサーバのホスト名かIPアドレスを指定。
  - -p ポート番号：接続先PostgreSQLサーバのポート番号を指定。デフォルトは「5432」。
  - -U データベースロール名：「データベースロール名」でデータベースに接続。
- 問い合わせ結果の出力形式変更用オプション
  - -A：位置揃えなしの出力モードに切り替える。
  - -F 区切り文字：フィールドの区切り文字を指定する。「-A -F ,」でCSV形式で出力。
  - -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”：SQLを実行しpsqlを終了する。シェルスクリプト等で有用。
  - -f “バッチファイル”：バッチファイルに記述されたSQLを実行しpsqlを終了する。
  - -o “ファイル名”：SQLの実行結果をファイルに出力する。標準出力には結果を返さない。
  - -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する。

❖実行例

– 問い合わせ結果の出力形式の変更

```
[lecture]$ psql -A -F , -U dbr lecdb
lecdb=> SELECT *
        FROM
        pgc2003
        LIMIT 10;
lecdb=> ¥q
```

– 【s82eg1.sh】シェルスクリプトからpsqlを実行しファイルに出力

```
psql -A -F , -U dbr lecdb¥
     -o ~/DB/results/psql_sample.txt¥
     -c “SELECT *
         FROM pgc2003
         LIMIT 10;”
cat ~/DB/results/psql_sample.txt

[lecture]$ sh ~/DB/sql/s82eg1.sh
ユーザdbrのパスワード: lecdb1912
```

- パスワードの省略方法については後述。

8.3 SQLコマンド

❖ psqlにおけるSQLコマンドの書式

- SQLはひと続きのコマンドからなり、セミコロン「;」がコマンドの終端となる。
- SQLでは文字列を除き大文字と小文字は区別されない。
  - テーブル名やカラム名を二重引用符「”」で囲めばそれぞれを大文字まじりで作成及び指定できるが推奨しない。
- 文字列は単一引用符「’」で囲んで入力する。
- 単一行のコメントアウトは二重ダッシュ記号「--」で行える。
- 複数行のコメントアウトは「/* コメント */」で行える。

```
lecdb=> SELECT ‘Hello world!’; --文字列入力の例
   ?column?
--------------
 Hello world!
```

8.4 メタコマンド

❖メタコマンド

- psql自身が実行するコマンドのこと。
- メタコマンドはバックスラッシュから始まり、改行がコマンドの終端となる（SQLコマンドとは異なることに注意）。
- 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”	バッチファイルに記述されたSQLを実行。
¥g “ファイル名”	「SQL文 ¥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できる。

❖表示変更用メタコマンド

メタコマンド	説明
¥x	拡張形式モードの切り替える。
¥t	結果のみ表示と完全表示の切り替える。
¥a	位置揃えなしの出力モードに切り替える。
¥f ‘区切り文字’	カラムの区切り文字を設定する。¥aを設定しないと無効。
¥H	HTML形式の出力モードに切り替える。
¥pset format ‘latex’	LaTeX形式の出力モードに切り替える。

❖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 コマンド名	SQLコマンドの説明を表示。
¥s	入力履歴を表示する。
¥timing	SQLの実行時間計測の有無を切り替える。
¥cd ‘ディレクトリ名’	現在の作業ディレクトリを変更する。
¥! UNIXコマンド	UNIXコマンドを実行する。例:¥! ls

8.5 ログインパスワードの入力省略設定

- 「psql」コマンドでPostgreSQLサーバに接続した際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「lecdb」への接続情報を記述したファイル「.pgpass」をホームディレクトリに作成。
- 「.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」に設定する。
    - ファイルの所有者(=自分)のみファイルを読み書き可能。
    - 「600」に設定しないと「.pgpass」ファイルが無視される。
  - “#”でコメントアウト可能。

❖【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。

```
$ cd
$ emacs .pgpass
localhost:5432:lecdb:dbr:lecdb1912
```

2. アクセス権の変更。

```
$ chmod 600 .pgpass
```

3. 接続の確認。

```
$ psql -U dbr lecdb
psql (10.5)
“help”でヘルプを表示します。

lecdb=>
```

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。
  - PostgreSQLではユーザ定義関数の作成にSQL言語、C言語、手続き型言語を使用できる。
  - PostgreSQLで標準で使用できる手続き型言語（Procedural Language）は以下の通り。
    - PL/pgSQL, PL/Tcl, PL/Perl, PL/Python ※PL/pgSQL以外は要インストール作業。
- 9章目次
  - 9.1 概要
  - 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、最後の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1のデータ型,引数2のデータ型,・・・)
RETURNS 戻り値のデータ型 AS $$
  SQL文;
$$ IMMUTABLE|STABLE|VOLATILE LANGUAGE SQL;
```

- 二重ドル引用符
  - CREATE FUNCTIONにとって関数の内容は文字列なので、単一引用符で囲む必要がある。
  - 二重ドル引用符は単一引用符と同じ働きをするが、単一引用符等の記号も文字として扱える。
- 関数の変動性分類（IMMUTABLE, STABLE, VOLATILE）
  - 関数がどのような振る舞いをするのか、オプティマイザに教えるためのオプション。
  - 分類の指定を省略した場合は「VOLATILE」になる。
  - IMMUTABLE：同一引数に対する呼び出しに対し、常に同一の結果を返す関数。
  - 「IMMUTABLE」でない関数をWHERE句内で使うとインデックス検索が行われない。
- 引数
  - カラム名や条件式の値としてN番目の引数の値を「$N」として使用できる。
  - テーブル名には引数を使用できない。
  - 「関数名（仮引数名 データ型）」のように仮引数名を指定することもできるが、SELECT文を使用するSQL関数では引数とカラム名を混同してしまう危険性があるため非推奨。

❖実行例

– 【s921eg1.sql】「Hello, World!」を表示する関数

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

lecdb=> ¥i ~/DB/sql/s921eg1.sql
lecdb=> ¥df
lecdb=> SELECT s921eg1();
```

– 【s921eg2.sql】引数をとるSQL関数

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

lecdb=> SELECT s921eg2(2000,19) AS seireki;
```

❖SQL関数の削除方法

```
lecdb=> DROP FUNCTION 関数名;
```

- 同じ名前の関数が複数存在する場合は「関数名(データ型)」で指定する。

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1のデータ型,引数2のデータ型,・・・)
RETURNS 戻り値のデータ型(複合型) AS $$
  SQL文;
$$ IMMUTABLE|STABLE|VOLATILE LANGUAGE SQL;
```

- 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。
  - テーブルのレコードの構造を表すために存在。テーブルを作成すると作成したテーブルと同じデータ型構造を持つ、テーブルと同名の複合型が作成される。
  - SQL関数の戻り値データ型に複合型を指定することで、複数列の結果を返すことが可能。
  - 複合型は任意に定義できる。

❖複合型の定義方法

```
CREATE TYPE データ型名 AS (
  変数1  データ型1,
  変数2  データ型2,
  ...
);
```

❖実行例

– 【s922eg1.sql】2つの整数型要素を持つ複合型

```
CREATE TYPE s922eg1 AS (
   a   INTEGER,
   b   INTEGER
);

lecdb=> ¥dT
lecdb=> ¥d s922eg1
```

– 【s922eg2.sql】複数列の結果を返すSQL関数

```
CREATE FUNCTION s922eg2(INTEGER, INTEGER)
RETURNS s922eg1 AS $$
   SELECT $1+$2,
          $1-$2;
$$ IMMUTABLE LANGUAGE SQL;

lecdb=> SELECT s922eg2(2000,19);
lecdb=> SELECT a,b FROM S922EG2(2000,19);
```

- 関数をFROM句に指定すれば、返り値を要素毎に抽出できる。

– 【s922eg3.sql】テーブルの行を返すSQL関数

```
CREATE FUNCTION s922eg3() RETURNS pgc2003 AS $$
  SELECT * FROM pgc2003 LIMIT 20;
$$ IMMUTABLE LANGUAGE SQL;

lecdb=> SELECT * FROM s922eg3();
```

- この方法では1行しか返ってこない。

❖作成したデータ型の削除方法

```
lecdb=> 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1のデータ型,引数2のデータ型,・・・)
RETURNS SETOF 戻り値のデータ型(複合型) AS $$
  SQL文;
$$ IMMUTABLE|STABLE|VOLATILE LANGUAGE SQL;
```

- SETOF句
  - SETOFを指定することで関数内の最後に記述されたSQL文が最後まで実行される。

❖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;

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

演習 SQL関数

❖【演習】赤経(度)と赤緯(度)を赤経赤緯(時,分,秒,度,分,秒)に変換するSQL関数を作成せよ。

1. 【s92ex1.sql】NUMERIC型の要素「a,b,c,d,e,f」から構成される複合型「s92ex1」を作成せよ。
2. 【s92ex2.sql】赤経(度)と赤緯(度)を赤経赤緯(時,分,秒,度,分,秒)に変換するSQL関数「s92ex2」を作成せよ。
   - 関数の本文部分は以下の通り。

```
SELECT  TRUNC(CAST(($1/15.0) AS NUMERIC),0),
        TRUNC(MOD(CAST(($1/15.0) AS NUMERIC),1)*60,0),
        MOD(MOD(CAST(($1/15.0) AS NUMERIC),1)*60,1)*60,
        TRUNC(CAST($2 AS NUMERIC),0),
        ABS(TRUNC(MOD(CAST($2 AS NUMERIC),1)*60,0)),
        ABS(MOD(MOD(CAST($2 AS NUMERIC),1)*60,1)*60);
```

3. 作成した関数「s92ex2」を使って、テーブル「PGC2003」の「id='PGC0077777'」の赤経赤緯を変換して表示せよ。

– SELECT句で関数を使用する場合。

```
lecdb=> SELECT s92ex2(ooo,ooo)
        FROM (
          ...
          ...
          ...
        ) AS t;
```

– FROM句で関数を使用する場合。

```
lecdb=> SELECT f.a,
               f.b,
               f.c,
               f.d,
               f.e,
               f.f
        FROM (
          ...
          ...
          ...
        ) AS t CROSS JOIN s92ex2(ooo,ooo) AS f;
```

❖【解答例】赤経(度)と赤緯(度)を赤経赤緯(時,分,秒,度,分,秒)に変換するSQL関数を作成せよ。

1. 【s92ex1.sql】NUMERIC型の要素「a,b,c,d,e,f」から構成される複合型「s92ex1」を作成せよ。

```
CREATE TYPE s92ex1 AS (
  a  NUMERIC, b  NUMERIC, c  NUMERIC,
  d  NUMERIC, e  NUMERIC, f  NUMERIC
);
```

2. 【s92ex2.sql】赤経(度)と赤緯(度)を赤経赤緯(時,分,秒,度,分,秒)に変換するSQL関数「s92ex2」を作成せよ。

```
CREATE FUNCTION s92ex2 (DOUBLE PRECISION,DOUBLE PRECISION)
RETURNS s92ex1 AS $$
  SELECT  TRUNC(CAST(($1/15.0) AS NUMERIC),0),
          TRUNC(MOD(CAST(($1/15.0) AS NUMERIC),1)*60,0),
          MOD(MOD(CAST(($1/15.0) AS NUMERIC),1)*60,1)*60,
          TRUNC(CAST($2 AS NUMERIC),0),
          ABS(TRUNC(MOD(CAST($2 AS NUMERIC),1)*60,0)),
          ABS(MOD(MOD(CAST($2 AS NUMERIC),1)*60,1)*60);
$$ IMMUTABLE LANGUAGE SQL;
```

3. 作成した関数「s92ex2」を使って、テーブル「PGC2003」の「id='PGC0077777'」の赤経赤緯を変換して表示せよ。

– SELECT句で関数を使用する場合。

```
lecdb=> SELECT s92ex2(t.ra, t.dc)
        FROM (
          SELECT *
          FROM pgc2003
          WHERE id='PGC0077777'
        ) AS t;
```

– FROM句で関数を使用する場合。

```
lecdb=> SELECT f.a,
               f.b,
               f.c,
               f.d,
               f.e,
               f.f
        FROM (
          SELECT *
          FROM pgc2003
          WHERE id='PGC0077777'
        ) AS t CROSS JOIN s92ex2(t.ra, t.dc) AS f;
```

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以降にはデフォルトでインストールされている。

❖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(引数1 データ型,引数2 データ型,・・・)
RETURNS 戻り値のデータ型 AS $$
  DECLARE
    変数名 データ型;
  BEGIN
    変数名 := 値;
    RETURN 式;
  END
$$ IMMUTABLE|STABLE|VOLATILE LANGUAGE PLPGSQL;
```

- 変数の宣言
  - 変数を使用する場合はDECLARE文内で宣言する。
- RETURNコマンド
  - 本文内に「RETURN 式」を記述することで、式の内容を結果として返すことができる。

❖実行例

– 【s931eg1.sql】Hello, World!

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

lecdb=> SELECT s931eg1();
```

– 【s931eg2.sql】引数をとる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;

lecdb=> SELECT s931eg2(2000,19) AS seireki;
```

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1 データ型,引数2 データ型,･･･)
RETURNS 戻り値のデータ型(複合型) AS $$
  DECLARE
    変数名 複合型;
  BEGIN
    変数名 := (値1,値2);
    SELECT カラム名 INTO 変数名 FROM テーブル名 WHERE 条件;
    RETURN 式;
  END
$$ IMMUTABLE|STABLE|VOLATILE LANGUAGE PLPGSQL;
```

- 複合型
  - 変数のデータ型として複合型を指定することができる(行変数)。
  - 複合型変数への代入方法は、直接代入とSELECT文を使った方法がある。

– 【s932eg1.sql】複数列を返すPL/pgSQL関数

```
CREATE FUNCTION s932eg1(x INTEGER, y INTEGER)
RETURNS s922eg1 AS $$
  DECLARE
    z s922eg1;
  BEGIN
    z := (x+y,x-y);
    RETURN z;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;

lecdb=> SELECT a,b FROM s932eg1(2000,19);
```

- s922eg1：9.2.2節で作成した複合型（INTEGER,INTEGER）

– 【s932eg2.sql】テーブルの行を返すPL/pgSQL関数

```
CREATE FUNCTION s932eg2() RETURNS pgc2003 AS $$
  DECLARE
    row pgc2003;
  BEGIN
    SELECT * INTO row FROM pgc2003 LIMIT 20;
    RETURN row;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;

lecdb=> SELECT * FROM s932eg2();
```

- 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,引数2 データ型,･･･)
RETURNS SETOF 戻り値のデータ型(複合型) AS $$
  DECLARE
    変数名 複合型;
  BEGIN
    RETURN QUERY SQL文;
    RETURN NEXT 変数名;
    RETURN;
  END
$$ IMMUTABLE|STABLE|VOLATILE LANGUAGE PLPGSQL;
```

- RETURN QUERY SQL文
  - 問い合わせの実行結果を関数の返り値テーブルに追加する。
- RETURN NEXT 変数名
  - 「変数名」に代入されているレコードを関数の返り値テーブルに追加する。
- RETURN
  - 関数が終了したことを表す。

❖実行例

– 【s933eg1.sql】RETURN QUERYの実行例

```
CREATE FUNCTION s933eg1() RETURNS SETOF pgc2003 AS $$
  BEGIN
    RETURN QUERY SELECT * FROM pgc2003 LIMIT 20;
    RETURN;
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;

lecdb=> SELECT * FROM s933eg1();
```

– 【s933eg2.sql】RETURN NEXTの実行例

```
CREATE FUNCTION s933eg2() 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;

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

- OFFSET：指定した行数を飛ばして表示

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;
```

❖単純CASE文

```
CASE 式
  WHEN 定数1 then 実行文;
  WHEN 定数2 then 実行文;
  ELSE 実行文;
END CASE;
```

❖検索CASE文

```
CASE
  WHEN 式 = 定数1 then 実行文;
  WHEN 式 = 定数2 then 実行文;
  ELSE 実行文;
END CASE;
```

❖整数FORループ文

```
FOR name IN 整数値..整数値 LOOP
  実行文;
END LOOP;
```

- FOR文のname変数はINTEGER型として自動的に定義される。

❖問い合わせFORループ文

```
FOR 変数名 IN SELECT文 LOOP
  実行文;
END LOOP;
```

❖WHILE文

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

演習 PL/pgSQL関数

❖【演習】

1. 【s93ex1.sql】強度「flux」と偏波強度「polflux」から偏波率を求める関数を作成せよ。
   - 引数と返り値の型：DOUBLE PRECISION
   - 関数の内容は以下の通り

```
RETURN ROUND(CAST((polflux/flux)*100 AS NUMERIC),1);
```

2. 【s93ex2.sql】作成した関数「s93ex1」を使って、テーブル「nvss」の「id='232541+593113'」の偏波率を求めよ。

❖【解答例】

1. 【s93ex1.sql】強度「flux」と偏波強度「polflux」から偏波率を求める関数を作成せよ。

```
CREATE FUNCTION s93ex1 (flux DOUBLE PRECISION,
                        polflux DOUBLE PRECISION)
RETURNS DOUBLE PRECISION AS $$
  BEGIN
    RETURN ROUND(
              CAST((polflux/flux)*100 AS NUMERIC),1
            );
  END
$$ IMMUTABLE LANGUAGE PLPGSQL;
```

2. 【s93ex2.sql】作成した関数「s93ex1」を使って、テーブル「nvss」の「id='232541+593113'」の偏波率を求めよ。

```
SELECT t.id,s93ex1(t.flux,t.polflux)
FROM (SELECT *
      FROM nvss
      WHERE id='232541+593113') AS t;
```

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

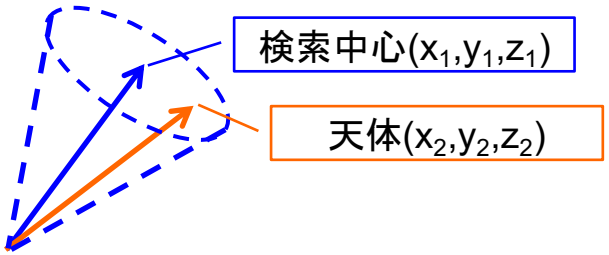
❖ コーンサーチ

- 天球面上のある座標を中心とした任意角度範囲内の天体を検索する方法は「Cone Search」や「Radial Search」と呼ばれる。
- 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する。

❖ 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すれば良い。

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



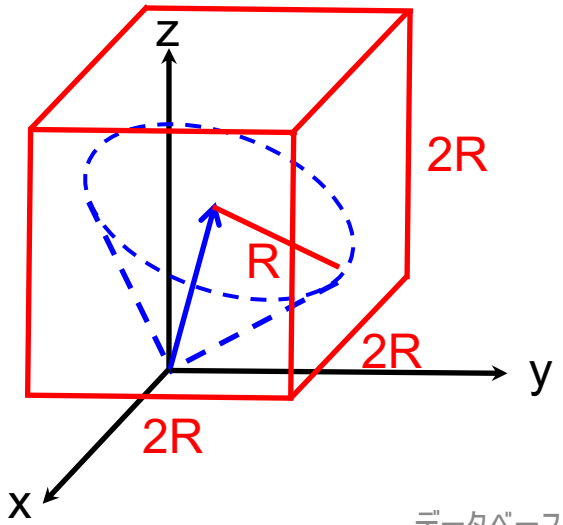
- このSQL文はWHERE句内でカラムに対する演算を行っているため検索に時間がかかる。
- このような検索をRDBMSに実装する定石として以下のような検索手法が存在する。
  1. インデックスを使ってざっくりと検索し、検索母数を減らす。
  2. 1の結果に対して厳密な検索を行う。

❖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したテーブル（天体番号、x、y、z）を用意する。
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取り出す。
3. 2で取り出した天体に対して検索中心からの角度を検索し、条件を満たす天体のみを取り出す。

– この実装手法は極付近の特別扱いが不要であるため単純に実装でき、検索半径が小さい時に高速に検索できる。

❖10章目次



10.2 直交座標系用テーブルの作成

- コーンサーチに使用する、天体の赤道座標を直交座標に変換したテーブルを作成
- 10.2節目次
  - 10.2.1 テーブルの作成
  - 10.2.2 座標変換用関数の作成
  - 10.2.3 テーブルへのデータの登録
  - 10.2.4 複合インデックスの作成

10.2.1 テーブルの作成

❖【演習】テーブルの作成

1. 【 CT_(pgc2003|nvss)_xyz.sql 】PGC2003カタログの天体の直行座標を収めるためのテーブル「pgc2003_xyz」と、NVSSカタログの天体の直行座標を収めるためのテーブル「nvss_xyz」を作成せよ。
   - カラム
     - id TEXT
     - x DOUBLE PRECISION
     - y DOUBLE PRECISION
     - z DOUBLE PRECISION
   - 全てのカラムに「NOT NULL」制約を設定する。
   - カラム「ID」に主キー「(pgc2003|nvss)_xyz_pkey」を設定する。
   - テーブルの作成方法はP62参照。

❖【解答例】テーブルの作成

1. 【ct_(pgc2003|nvss)_xyz.sql】PGC2003カタログの天体の直行座標を収めるためのテーブル「pgc2003_xyz」と、NVSSカタログの天体の直行座標を収めるためのテーブル「nvss_xyz」を作成せよ。

```
CREATE TABLE pgc2003_xyz (
  id          TEXT                NOT NULL,
  x           DOUBLE PRECISION    NOT NULL,
  y           DOUBLE PRECISION    NOT NULL,
  z           DOUBLE PRECISION    NOT NULL,
  CONSTRAINT pgc2003_xyz_pkey PRIMARY KEY (id)
);
```

```
CREATE TABLE nvss_xyz (
  id          TEXT                NOT NULL,
  x           DOUBLE PRECISION    NOT NULL,
  y           DOUBLE PRECISION    NOT NULL,
  z           DOUBLE PRECISION    NOT NULL,
  CONSTRAINT nvss_xyz_pkey PRIMARY KEY (id)
);
```

10.2.2 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】赤経赤緯を赤道座標系(ra,dc)から直交座標系(x,y,z)に変換するPL/pgSQL関数「fEq2X」、「fEq2Y」、「fEq2Z」を作成せよ。
   - 引数と返り値の型:DOUBLE PRECISION
   - それぞれの関数の本文は以下の通り。
     - fEq2X

```
RETURN COS(RADIANS(ra))*COS(RADIANS(dc));
```

     - fEq2Y

```
RETURN SIN(RADIANS(ra))*COS(RADIANS(dc));
```

     - fEq2Z

```
RETURN SIN(RADIANS(dc));
```

2. 次の計算を行い結果が1になることを確認せよ。raとdcは任意の値とする。

```
lecdb=> SELECT SQRT(
                  fEq2X(ra,dc)^2 +
                  fEq2Y(ra,dc)^2 +
                  fEq2Z(dc)^2
                );
```

❖【解答例】座標変換用関数の作成

1. 【fEq2(X|Y|Z).sql】赤経赤緯を赤道座標系(ra,dc)から直交座標系(x,y,z)に変換するPL/pgSQL関数「fEq2X」、「fEq2Y」、「fEq2Z」を作成せよ。

– 【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;
```

10.2.3 テーブルへのデータの登録

❖INSERTコマンド書式

```
INSERT INTO テーブル名 (カラム1,カラム2,...)
VALUES(値1,値2)|SELECT文 ;
```

- INSERTコマンド
  - テーブルへレコードを追加するためのコマンド。
  - VALUESで指定した値を、SELECT文で別テーブルの値を追加することができる。

❖【実習】データの登録

1. テーブル「pgc2003_xyz」と「nvxx_xyz」にデータを登録。

```
lecdb=> INSERT INTO pgc2003_xyz (id,x,y,z)
        SELECT id,fEq2X(ra,dc),fEq2Y(ra,dc),fEq2Z(dc)
        FROM pgc2003;

lecdb=> INSERT INTO nvss_xyz (id,x,y,z)
        SELECT id,fEq2X(ra,dc),fEq2Y(ra,dc),fEq2Z(dc)
        FROM nvss;
```

- 成功すると「INSERT 0 追加された行数」が表示される。

10.2.4 複合インデックスの作成

❖複合インデックスの作成方法

```
CREATE INDEX インデックス名 ON テーブル名(カラム1,カラム2,・・・)
```

- 複合インデックス
  - テーブルの複数のカラムに対して1つのインデックスを定義する方法。
  - あるレコードを検索する時に単一カラムに対する検索だけでは十分な絞り込みを行うことができず、複数列に対する検索をしなければならない場合に使用する(例:名字と名前、赤経と赤緯など)。
- 注意点
  - 複合インデックスのカラムの定義順序は検索性能に影響する。例えば「カラム1」、「カラム2」、「カラム3」に対して「カラム2」=>「カラム1」=>「カラム3」という順番で絞り込むのが最適と考えられる場合、この順番で複合インデックスのカラムを定義すべき。
  - 複合インデックスを設定したカラムに対して単一カラムに対する検索を行った場合、最初に定義したカラム(上記の例だとカラム2)以外はインデックス検索を行えない。

❖【実習】複合インデックスの作成

1. テーブル「pgc2003_xyz」と「nvss_xyz」のカラム「x」、「y」、「z」に複合インデックスを設定。

```
lecdb=> CREATE INDEX ON pgc2003_xyz(x,y,z);
lecdb=> CREATE INDEX ON nvss_xyz(x,y,z);
lecdb=> VACUUM ANALYZE pgc2003_xyz;
lecdb=> VACUUM ANALYZE nvss_xyz;
```

- x、y、zの検索優先度は同一であるため、カラムの定義順序は何でも良い。

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したテーブル（天体番号、x、y、z）を用意する。
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、厳密に条件を満たす天体のみを取り出す。

– 必要なもの

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

❖10.3節目次

10.3.1 コーンサーチに必要な関数の作成

❖【演習】コーンサーチに必要な関数の作成

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)を求める関数「fDistanceArcminXyz(x1,y1,z1,x2,y2,z2)」をPL/pgSQLで作成せよ。
   - 引数と返り値の型:DOUBLE PRECISION
   - 関数の内容は以下の通り。
     - 丸め誤差により「a」の値がACOSの定義域外になることがあるためIF文で対処。

```
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
```

2. 次の計算を行い結果が「5400」になることを確認せよ。

```
lecdb=> 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「fArcmin2Rad」を作成せよ。
   - 引数と返り値の型：DOUBLE PRECISION
   - 度をラジアンに変換する組み込み関数：RADIANS(V)

4. 次の計算を行い結果が「3.14...」になることを確認せよ。

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

❖【解答例】コーンサーチに必要な関数の作成

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)を求める関数「fDistanceArcminXyz(x1,y1,z1,x2,y2,z2)」をPL/pgSQLで作成せよ。

```
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 コーンサーチの練習

❖【実習】赤経(200[deg])、赤緯(5[deg])を中心とした半径(60[arcmin])内にある天体をテーブル「pgc2003」から検索。

1. 【s1032eg1.sql】検索中心を中心とした1辺2Rの立方体内にある天体の検索。

```
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))
;
```

2. 【s1032eg2.sql】1で取り出した天体から、赤経(200[deg])、赤緯(5[deg])を中心とした半径(60[arcmin])内にある天体を検索。

```
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;
```

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」用のコーンサーチ用SQL関数「fConesearchPgc2003(ra,dc,r)」を作成せよ。

```
CREATE FUNCTION fConesearchPgc2003 (
                  DOUBLE PRECISION,
                  DOUBLE PRECISION,
                  DOUBLE PRECISION
                ) RETURNS SETOF TypeCone AS $$

...

$$ IMMUTABLE LANGUAGE SQL;
```

3. 以下のSQLの検索件数が「s1032eg2」と同じか確認せよ。

```
lecdb => 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「fConesearchNvss(ra,dc,r)」を作成せよ。

5. 以下のSQLを実行し結果を確認せよ。

```
lecdb => SELECT * FROM fConesearchNvss(200,5,60);
```

❖【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」用のコーンサーチ用SQL関数「fConesearchPgc2003(ra,dc,r)」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;
```

4. 【fConesearchNvss.sql】テーブル「nvss」用のコーンサーチ用SQL関数「fConesearchNvss(ra,dc,r)」を作成せよ。

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

変数は文字列演算子「||」で
文字列と結合している。

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することができる。
- 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行い、検索半径内で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】ある赤経赤緯の天体をテーブル「nvss」から探すために必要なSQL関数の作成

1. 【fGetNearestNvssObjID.sql】テーブル「nvss」にコーンサーチを行い、検索中心に最も近い天体の「id」、「ra」、「dc」、「distance」を返す関数「fGetNearestNvssObjID(ra,dc,r)」を作成。￥

```
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;
```

fConesearchNvssとの相違点

10.4.2 クロスマッチの実行

❖【実習】クロスマッチの実行

1. 【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する。

```
SELECT *
FROM (SELECT id,ra,dc
      FROM pgc2003
      WHERE anames LIKE '%NGC5194%'
     ) AS 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。
  - 式インデックスの作成方法
  - 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

パスワード表

- **new-rXX**
  - アカウント：ホワイトボード参照
  - パスワード：ホワイトボード参照
- **ゲストOS 一般ユーザ**
  - アカウント：lecture
  - パスワード：lecdb1912
- **ゲストOS PostgreSQL管理ユーザ**
  - アカウント：postgres
  - パスワード：lecdb1912
- **ゲストOS システム管理ユーザ**
  - アカウント：root
  - パスワード：lecdb1912
- **PostgreSQL 一般ロール**
  - ロール：dbr
  - パスワード：lecdb1912
- **PostgreSQL 管理ロール**
  - ロール：postgres
  - パスワード：lecdb1912