

はじめに

本書は、PostgreSQL を題材に、SQL を理解し、身につける為の本となります。

SQL は ISO で標準化された、もっとも有名な言語の 1 つです。

これは PostgreSQL 以外も含めたデータベース(以降「DB」と記載)を操作する事を目的とする言語ではありますが、昨今はデータサイエンス(DS)の分野でも SQL が活用されており、snowflake 等のデータウェアハウス(DWH)でも使用できるなど活用範囲は広範囲と考えます。

その為、DB を操作する DB エンジニアやアプリケーション開発エンジニア以外の方も「エンジニアの一般教養」として学んでおいて損はない言語かと思います。

一方ではじめて IT の世界に足を踏み入れる初心者や、DB を操作しない立場で業務を行っていると独特の文法もあり中々慣れない言語で難しくも感じるかと思います。

いきなり実業務で長文の SQL と遭遇し、心が折れかけた方、苦手意識がついてしまった方もいるかもしれません。あるいは書籍や資格等で「わかった」気にはなったものの、実際に使おうとすると苦労する、という方もいるかもしれません。

本書ではそういう方々に向けてハンズオンを通して「知る」だけでなく、「身につけてもらう」事を目標としています。

是非「読んで理解」した気になるだけでなく、ハンズオンを通して SQL による DB の操作方法を身につけてもらえば、と思います。

本書により読者の皆様の知見が向上する事を願っております。

目的

本書は情報の提供のみを目的としています。

免責事項

本書の内容を用いた開発・運用は、必ずご自身の責任と判断によって行ってください。

開発・運用の結果について、著者はいかなる責任も負いません。

また本書には一部見解も記載していますが、筆者の個人的な見解となります。

なお本書の動作検証は Ubuntu20.04 LTS, PostgreSQL12 によるものです。

バージョンや使用する OS により差異が生じる可能性がありますので、その点はご了承ください。

注意点

本書は SQL の理解を目標としていますが、PostgreSQL の機能に依存するものも存在します。

また、SQL は多様なオプションや省略記法など様々な記載方法があり、全パターンを記載する事は難しい為、代表的な物に絞っている事、ご了承ください。

なお本書ではSQL文を「小文字」で記載していますが、お作法としては「大文字」で記載する事が一般的です。小文字で記載している点は本書の都合とご理解ください。

商標

本書に掲載するシステム名や製品名は関係各社の商標または登録商標です。
本書では、商標マークや著作権マーク、登録商標マーク等は省略します。

対象読者

本書は以下のような皆様を想定しています。

- DB初心者
- PostgreSQLをこれから触ってみたい人

ただし後半はDBの世界に踏み込んだ内容にも触れる為、学び直しやある程度SQLを知っているという方にも参考になればと思っております。

本書の構成

本書は大きく以下の2部構成となっています。

- 教科書編
- 問題編

教科書編ではSQLの種類ごとに構文や何ができるのかを説明します。

問題編はハンズオン形式で読者の皆様が手を動かして解く為の練習問題を記載した章となります。

ドキュメントの都合上、教科書編と問題編は1冊となる為、構文とそれを使う練習問題が離れたページに記載されますが、その点は本書をダウンロード後、名前を変える事で2冊同時に開いて進める等の工夫をお願いします。

練習問題の答えや、誤記等による正誤表は[当社ブログサイト^{*1}](#)に掲載しますので適宜ご確認ください。

なお、本書は「まずはふんわりでも理解する」事を重視している事と社内研修で使用する資料をベースに作っている都合上、フランクな記載やふわっとした内容、パワーポイントスライドのスクリーンショット画像が多くなりますが、ご了承ください。

1. <https://www.ntt-tx.co.jp/column/>

不明点や誤記等があった場合

不明点やご意見等がありましたら、以下のどちらかでご連絡ください。

メールから： yoshiki4490117@yahoo.co.jp

目次

- はじめに 1

• 第1部 教科書編	9
• 1章 データベースと PostgreSQLについて	10
• 1-1.データベースの種類とリレーションナルデータベースについて	10
• 1-2.データベースで考えるべきことと SQLについて	14
• 1-3.用語整理と PostgreSQLへの接続・操作方法について	16
• 2章 DDLについて	20
• 2-1.本章の概要	20
• 2-2.PostgreSQL の内部構成について	21
• 2-3.各種文法	23
• データベースやスキーマ等の作成(create)	
• データベースやスキーマ等の変更(alter)	
• テーブルの作成と関連するもの	
• 各種制約	
• シーケンス	
• インデックス	
• ビューとマテリアライズドビュー	
• 説明文の追加(comment)	
• データベース、スキーマ等の削除(drop)	
• 3章 DCLについて	32
• 3-1.本章の概要	32
• 3-2.DCL とロールについて	33
• 3-3.文法等について	35
• ロールの操作に関する流れとロールの作成	
• ロールへの権限付与(grant文)	
• ロールからの権限削除(revoke文)	
• 所有者(オーナー)という概念について	

• 4 章 TCL について	38
• 4-1. 本章の概要	38
• 4-2. トランザクションとその操作について	39
• トランザクションとは	
• トランザクションを操作する各種文法(begin/savepoint/commit/rollback/abort 文)	
• savepoint 関連の文法(rollback to savepoint/release savepoint 文)	
• 明示的トランザクションと暗黙的トランザクションについて(txid_current 関数)	
• 4-3. ロックについて	43
• デッドロックについて	
• テーブルロック(lock table 文)	
• 行ロック(select for update/select for share 文)	
• 4-4. トランザクション分離レベルと禁止観点について	46
• トランザクション分離レベルと禁止観点について	
• トランザクション分離レベルの変更方法と確認方法(set/show 文)	
• 5 章 DML の基礎について	48
• 5-1. 本章の概要	48
• 5-2. データの登録/変更/削除/取得について	49
• データの登録(insert 文/copy)	
• データの更新と更新箇所の確認(update 文/returning)	
• データの削除(delete/truncate 文)	
• データの取得(select 文)	
• 5-3. 操作対象データの条件指定について	54
• 条件の指定(where 句)と演算子	
• 各種関数と count,null の指定(is null)	
• 文字の部分一致検索(like, ilike/similar to/正規表現/参考:JSON データの検索)	
• 重複排除と列の別名付与(distinct/distinct on/as)	
• グルーピングとその検索について(group by/having)	
• 並び替えと取得範囲の絞り込みについて(order by/offset/limit/between)	

• 6 章 DML の応用について	67
• 6-1. 本章の概要	67
• 6-2. 結合について	68
• inner join/outer join/cross join	
• on/using/natural	
• 6-3. 問い合わせ結果の結合について	74
• union/union all/except/intersect	
• 6-4. 副問合せについて	76
• 副問合せの基本的な記法(select ~ from (select ~) / select (select ~))	
• 相関副問合せについて	
• where 句の条件で複数行指定する方法(in/some,any/exists)	
• 6-5. with 句と再帰的問い合わせについて	84
• with	
• with recursive	
• 6-6. その他	87
• SQL の内部処理方法の確認(explain/explain analyze)	
• Window 関数(over (partition by) / over (order by))	
• UPSERT(on conflict/excluded/merge)	
• DML のおわりとして	
• 7 章 より高度な仕組みの活用・操作	94
• 7-1. 本章の概要	94
• 7-2.+α その1: 高度な操作・活用	95
• 宣言的パーティショニングとテーブル空間について	
• 自ら定義する型：ドメインについて	
• プリペアドステートメントについて	
• 外部キー制約の細部アクション設定について	
• データ分析の高度化に向けて(case 文/grouping sets,rollup, cube)	
• contrib と dblink の活用、2相コミットメントについて	
• [参考] 表明について	
• 7-3.+α その2: PL/pgSQL の活用 及び 自動処理について	110
• トリガーファンクションとルールについて	
• ストアドファンクションとストアドプロシージャ	
• カーソルについて	
• 教科書編のおわりに	

• 第2部 問題編	121
• 環境構築	121
• 各種問題	125
• はじめに	125
• 1章 DDL	127
• 2章 DCL	137
• 3章 TCL	142
• 4章 DML 基礎	153
• 5章 DML 応用	171
• 6章 高度な操作	194
• その1:外部制約の詳細や条件文等	194
• その2:パーティションとテーブル空間	199
• その3:contribとdblink	204
• その4:PL/pgSQLと自動処理	209
• 7章 +α	221
• その1:PL/pgSQLや自動処理	221
• その2:更に色々試してみたい人向け	223
• 各種問題のヒントと答え	230
- 各種問題のヒント	230
- 各種問題の答え	251
• 問題編にのみ登場した文法等の補足	252
• おわりに	271

教科書編

以降では「データベースとは何か」という点から、SQLの種類ごとに構文や概念等を記載する。
教科書編の読み方は以下の通り。

教科書編の読み方

- 黒字で基本的な説明、
赤字で構文・文法を、
青枠で例文を記載します。

※ SQL文は資料上、小文字(先頭大文字)で記載されていますが、
大文字で記載する方が一般的です。

- まずは例文の内容を詳細に理解するのではなく、「何をやっているのか」「どういう時に使うのか」といったが概要を把握するのが良いと考えます。
→ 詳細は練習問題を解く時に確認ください。
- 構文・文法や例文含め、画像ファイル内に埋め込んでいる事がが多いです。
この理由は読みやすさを重視した事と、
SQLはコピペをせずにゼロから手打ちで作り上げていく事が
身につける事につながると考える為です。
ハンズオン時には教科書編を確認しつつ、ゼロから作成をしてみましょう。

1章 データベースと PostgreSQL について

本章では「データベースとは何か」という点から、データベースの種類・データベースとSQLの関係等について触れていく。

1-1. データベースの種類とリレーションナルデータベースについて

データベース(DB)は以下のように活用タイミングは様々である。

DBとは？

- DBとはシステムが使用するデータを管理する為のもの。
効率よく大量のデータを管理できる。
→特にユーザが入力したデータを覚えておく必要があるシステム等、多くのシステムで使う。
エンジニアが使用するタイミングもマチマチ。(以下、例)

【開発時】

- バグ解析(データ異常系)
- 試験データ投入
- 試験環境準備

【運用時】

- パフォーマンスチューニング
- 問い合わせの為の解析

※ システム開発だけでなく、OSSを使用する際にもDBが使用されるケースはある
例) OpenStack, Zabbix 等

ユーザが入力したデータを覚えておく必要があるものの代表例として「ユーザID」と「パスワード」があげられる。

これはアクセスしたその日・その時だけでなく、後日ユーザが再度アクセスした場合にも必要となる。

このように長期間保存が必要な情報はDBに登録する。

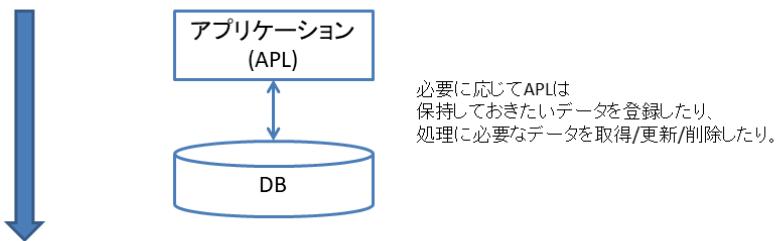
* OpenStack^{*1} は AWS のような仮想化リソース(仮想マシン(VM)等)を管理する為のオープンソース(OSS)のクラウド基盤ソフトウェア。作成した VM 等の管理に DB を使用する。

Zabbix^{*2} はオープンソースの監視ソフトウェア。例えばアプリケーションを載せているサーバ(マシン)に不具合があった場合等に通知を行う。

1. <https://www.openstack.org>
2. <https://www.zabbix.com/jp>

アプリケーションとDB

- DBはあくまでデータ管理のツール。
アプリ側との連携方法含めてシステム開発では検討する。
DBの事を知っていれば無駄な処理をアプリで実装しなくても
良くなるかもしれない。



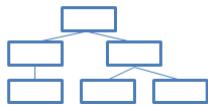
その意味でも操作(SQL)とどんなことができるのか
理解しておくことは重要。

DBにも様々な種類がある。以下に「概念」で分けた場合の種類とPostgreSQLがどれに該当するかを記載する。

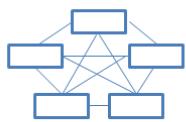
DBの種類 -概念-

- DBには以下の種類が存在

① 階層型



② ネットワーク型



③ リレーションナル型



よく使うのは③で、PostgreSQLも③。

※ 以降リレーションナル型DBを「RDB」と記載。
また、PostgreSQLを「postgres」と記載。

【ポイント】リレーションナル型ならエクセルでいいのでは？

エクセルだと沢山データを管理すると激重(例:1億とか)。
また複数人が同時に触ってデータ矛盾が発生しない、というような機能もない。

参考までにエクセルの1シートの最大行は104万8576行である。

同時編集はbox等により同時編集も行いややすくなったが、セルの同時編集には注意が必要である。

RDBではSQLを使用する。SQLというキーワードで見るとRDB以外にNoSQLやNewSQLと呼ばれるものがある。

DBの種類 –SQLとNoSQL–

- SQLデータベースとNoSQLデータベースという分け方もできる。
→どちらかが完全に上位というわけではなく、
お互いに長所が異なる為、使い分けが必要。
postgresはSQLデータベース。

【NoSQLデータベースとは】

- 以下にNoSQLデータベースの代表例を記載。

NoSQL	ossの代表例	Aws上のサービス例
KVS (Key Value Store)	Redis , memcached	ElastiCache
カラム指向DB	Cassandra, HBase	Keyspaces
ドキュメントDB	MongoDB	documentDB
時系列DB	InfluxDB	Timestream
グラフDB	Neo4j	Neptune

[参考]

最近はNoSQLと比較した言い方で、NewSQLというのもも。
AIの普及でDWH的な使い方も求められており、DBは古くからあるが、
新しい物も出続けている。

NoSQLはNot Only SQLの略であり、No SQLではない事を注意する必要がある。

NewSQLは分散SQLデータベースとも呼ばれる。

RDBとNoSQLの得意分野について

- RDBとNoSQLのトレードオフは一般的には
以下のようない形となる

観点	RDB	NoSQL	備考
処理速度	×	○	検索やデータ投入処理にかかる時間。
検索性/操作性	○	×	SQLによる複雑な処理はNoSQLはできず 独自クエリ言語を使用する。 またSQLでできた事に対する制約が付く事が 多い
一貫性	○	×	データが矛盾なく登録されること。 トランザクション(※)の有無、ともいえる。
分散、拡張性	×	○	性能向上の為のスケールアウトさせやすい。

※トランザクションについては後述。

RDB の 1 つに本書が扱う postgres が該当するが、その他にも様々な RDB が存在する。

RDBの種類

- 以下にRDB(リレーショナルデータベース)の代表的な物を記載する。

種類名	概要
Oracle	有料。性能も良いし、機能もリッチ。
MySQL	無料と有料版がある。 一般的にマルチスレッドタイプのアーキテクチャでWeb系に強いが複雑なSQLに不向き。
SQL Server	Windows製品(excel等)と相性が良い。
SQLite	無料。複雑な設定不要で、簡易な検証等向け。
PostgreSQL	無料(OSS)。複雑なSQLに対応でき、基幹系システムで利用できる。

1-2. データベースで考えるべき事とSQLについて

以下にDBを扱う際に考えないといけない事を記載する。

必要知識は開発の工程（設計や試験）や業務（開発か運用か）により異なる為、以下は一例である。

DBで考える事と本書の対応の範囲

- DBは操作だけできれば良いというわけではない
設計等も含めると、DB関連で考える事・覚える事は多い。
以下に例を記載。

段階	考える事
設計	概念設計
	論理設計
	物理設計
開発	プログラムとの組み合わせ
	試験環境準備・データ準備
運用	バックアップ/リストア
	パフォーマンスチューニング
	トラブルシューティング

※ 最近はデータ分析でもよく使われる

【ポイント】

- 設計等は別書で学習してもらうものとし、本書は操作に特化する。
操作もpostgresの設定等も関係する箇所があるが、今回は最小限とする。

RDBを学ぶ為になぜはじめにSQLを学ぶのか

- RDBの操作の第一歩としてSQLを学ぶことで、
何ができるのか、何をやっているのかが理解しやすい為。
また、一番最初に躊躇やすいのがSQLである為。
 - 自分で考えて問い合わせ文を作り、
 - うまくいかない理由や改善案を考え、
 - 正しく動いているのを見て説明の意味を理解できる！

Q それってテキスト生成AI(特にLLM)でよくない？(正しく理解する意味あるの？)

A 生成AIに頼る事で生産性を上げる事自体は良いと考える。
ただし複雑な物や、データが少ないと思われる踏み込んだ内容では時々誤りが発生する。

⇒だからAIがダメというつもりはない。ただし最後に判断するのは人間。

(責任をとるのも人間、人は正しく理解できている必要がある)

⇒嘘情報や、正しく動いているように見えるが、
他パターンだと動かないケースなどに対処することが求められる。

(設計や背景をAIが理解できていない場合等。)

※ 特に言語は正解は複数ある。

複雑な事をやろうとすればするほど柔軟性はなくなる点は人間が作る分にも同じだが、要件達成につながるのか人間が最終判断をする。

LLMは正答率100%を保証できない。
正解は人による為。

SQLの種類は以下の4種がある。

以降の章では各SQLの種類ごとに細部を紹介していく。

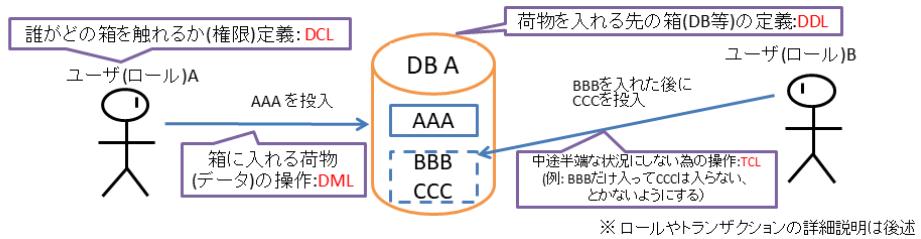
ここでは下図を確認し、まずは各SQLの種類の概要をおさえてほしい。

SQLの種類

- SQLとはDBを操作する為の言語。
SQLは大きく分けると以下のような種別となる

種類	日本語名	説明	例
DDL	データ定義言語	必要な要素を定義する為のもの	create alter drop
DCL	データ制御言語	アクセス権の付与等を行うもの	revoke grant
TCL	トランザクション制御言語	トランザクションを管理する為のもの	transaction commit
DML	データ操作言語	DB内のデータを操作する為のもの	select insert update delete

※ TCLはDCLにまとめられることもある



SQLの種類は引っ越しをイメージするとわかりやすいかもしれない。

引っ越しを行う際には、新居に運ぶものをまずは段ボール等に収納する。

この物を入れる「箱」を用意する、またどんな物を入れるのか等を定義するものがDDLである。

実際に用意した箱に物を収納するのがDMLである。

また、箱の中身を自分以外に見られたない場合には何かしらの対策を打つと考えられるが、誰がその箱を開けるのか等を定義するのがDCLである。

なお、TCLはトランザクションを制御する言語だが、トランザクションは4章で説明する為、ここでは例は省略とする。

1-3.用語整理とPostgreSQLへの接続・操作方法について

テーブル(表)関連の用語を以下に記載する。

本書でもカラムやレコードといったキーワードは使用する為、確認しておいてほしい。

表に関する用語について

- 表(テーブル)については今後、以下の用語を使用して説明する。

【personテーブル(表)】(例)

カラム(列)					
レコード(行)	部署名	メンバ	ランク	年齢	カラム名(列名)
	総務	A	7	30	フィールド
	開発	B	8	25	
	経企	C	5	41	
	開発	D	6	35	
	開発	E	7	27	

以降は PostgreSQL のセットアップや SQL 実行モード(対話モード)への接続方法等を記載する。

PostgreSQLの初期設定

- PostgreSQLをインストールするとデフォルトで設定されたDBが動く事もあるが、自分で設定したい場合は以下を実行する。
- PostgreSQLをインストールするとデフォルトで設定されたDBが動く事もあるが、自分で設定したい場合は以下を実行する。

```
### postgresユーザにスイッチ
sudo su - postgres

### DBクラスタの新規作成(初回のみ実行)
initdb -D [データ配置ディレクトリ] --no-locale --encoding=utf8

##### PostgreSQLを起動
pg_ctl -D [データ配置ディレクトリ] start
```

マシンを再起動した場合には
再実施が必要な事がある

DB クラスタは PostgreSQL を実行する為のフォルダ群(ディレクトリ)と実体ファイル群というイメージである。例えばどのように PostgreSQL を動かすのか、細かい設定ができる設定ファイルや DB そのものデータファイル等が該当する。DB クラスタの作成は 1 回作れば、以降は PostgreSQL が処理に応じて変更を加えていく為、基本は初回のみ実施と考えてよい。

DB クラスタを作成後、DB(PostgreSQL)を起動するコマンドとして pg_ctl がある。これも PostgreSQL を起動するマシンが起動している際は 1 回で良いが、マシン再起動をすると PostgreSQL も停止する為、再起動後に再実施が必要となるケースがある。

ただしマシンに PostgreSQL をマシン起動後に自動起動する設定を入れておけば上記は不要となる為、マシン設定により必要かどうかは変わる。

これらの操作は問題編の環境構築でも手順を扱うため、あわせて確認してみてほしい。

PostgreSQL起動後は psql で接続を行う。

psqlとSQLの実行について

- PostgreSQLのSQLを実行する為のモードに接続するには psqlコマンドを実行する。
psqlでは様々なオプションがあるが、以下3つは本書でもよく使うので覚えておいてほしい。
 - U: ログインするロールを指定(デフォルトpostgres)
 - d: 接続するDBを指定(デフォルトpostgres)
 - p: 接続するポート番号指定(デフォルト5432)

```
psql -U postgres -d postgres -p 5432
```

デフォルトのものを使う場合は
それぞれ指定不要。

- psqlで接続後はSQL文を実行できる。
1文の完了は「;」で判定している。
よって1行に1SQLを書かないといけないわけではなく、改行する事で複数行で記載が可能となる

```
select * from table1  
where id = 1 ;
```

;を入れなければSQL文完了とみなされない。
長文SQLはわかりやすくするために、改行して書いても良い。

psqlとあわせて覚えておきたいのが、SQL実行モードに接続後に使用できるメタコマンドについてである。
特に\lと\qはSQL実行モードを使う際に確実に使う為、覚えておいてほしい。

メタコマンドについて

- SQL標準ではなく、postgres独自の便利機能としてメタコマンド(psqlの機能)というものが存在します。
これはDBに接続後、特定キーワードを打つと様々な情報が見られるものです。
演習でも使うケースが出てきますので、よく使うものを以下に記載します。

メタコマンド	説明
\l	DB一覧取得
\dn	スキーマ一覧取得
\dt	テーブル・ビュー・シーケンス一覧と所属スキーマ表示
\dt [対象物名]	テーブル・ビュー・シーケンス・インデックスの説明
\di	インデックス一覧取得
\dg	ロール一覧取得
\x	結果表示切替
\r	入力のリフレッシュ(最初から入力となる)
\q	SQL実行モード(対話モード)から抜ける

- SQLでは最後に「;」をつけるのが基本。
ただし、メタコマンドは「;」は不要です。
※ Linux OSコマンドも同様に「;」不要です。

SQLを入力していて誤字った場合などは
\rでリフレッシュすると良い

最後にPostgreSQLのセットアップが完了・起動した状態でマシンに再接続した場合に、SQL実行モードに接続するまでの手順を記載する。

[参考] 初期セットアップ後のDBへの接続方法

- 対象マシンに接続後、以下を実施（Linux OSコマンド）。

```
### postgresユーザに切り替え  
sudo su - postgres
```

```
### DBに接続(-Uでpostgresユーザを指定。-dで対象postgres DBに入る)  
psql -U postgres -d postgres
```

- DBからの接続解除は以下。

```
¥q
```

2章 DDLについて

本章ではデータを入れる「箱」を整える DDLについて触れていく。

2-1. 本章の概要

はじめに本章で触れる概念について整理する。

この章内でどんな事について触れていくのか概要をおさえてほしい。

DDL 概要

【登場概念】

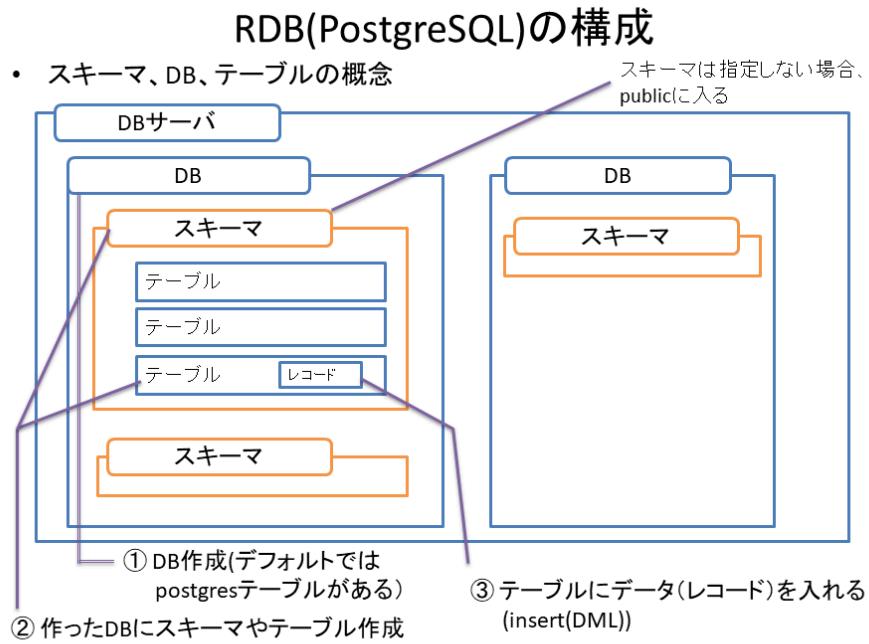
タイトル
データベース
スキーマ
テーブル
制約
インデックス
シーケンス
ビュー
マテリアルズドビュー

【実施概要】

文法
create
alter
drop
reindex
cluster
refresh materialized view
comment
if exist

2-2.PostgreSQLの内部構成について

postgresでは、DBサーバ（プロセスそのもの。postgresそのものと考えてもらって良い）があり、そのなかに「DB」「スキーマ」「テーブル」という階層構造となっている。



※ スキーマという言葉は「データの性質、形式、他データとの関連」等、データ定義の事を指す場合もある為、注意。

postgresを起動した際にDBサーバはできる。

言い換えると initdb による DB クラスタの作成と pg_ctl による DB 起動で DB サーバが起動する。

「サーバ」という名前がついているが、マシンそのものではなく、プロセスである点に注意してほしい。

よってマシンそのものに1つしかDBサーバを立ち上げられないわけではない。

initdbにて対象とするディレクトリを別に指定し、受けつけるポート番号を変更すれば1マシンで複数DBサーバを起動することも可能である。

この点は演習問題を実施する際にも扱う為、併せて確認してほしい。

また、DBについてはpostgres等、スキーマはpublicというデフォルト値があり、初期化した後には存在する。（もちろん自分で作成しても良い）

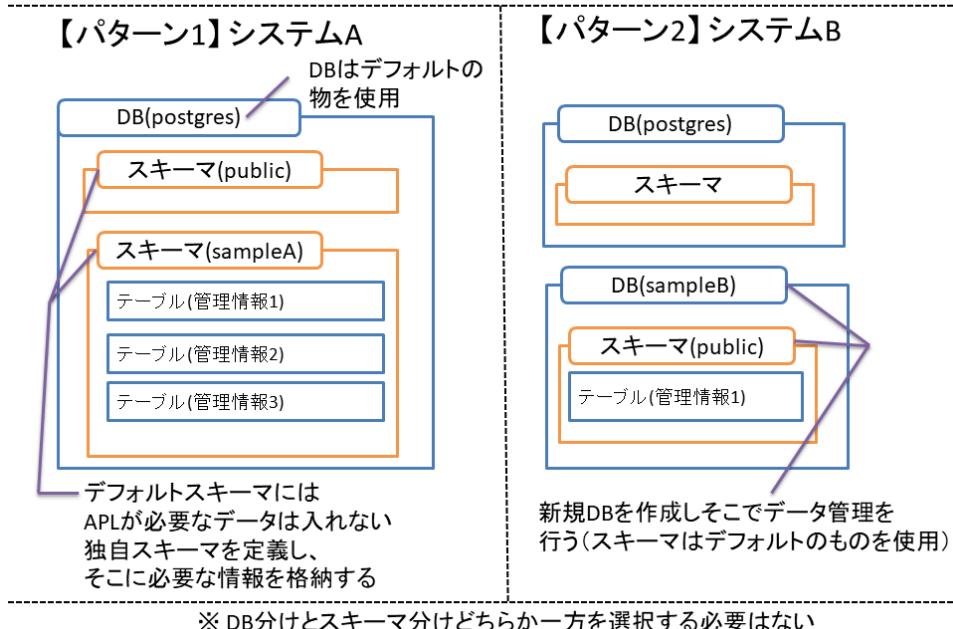
テーブルはデフォルト値もない為、使用する際に必ず作成が必要となる。

以下に上記構成を活用した具体例を記載する。

なお、どちらが良いか等はシステム要件により異なる為、必ずしもどちらが良いというものではない。

また例はわかりやすくするために2例を出しているが、サンプルAとサンプルBの両方を実施する案をとっても良い。

PostgreSQLの構成具体例



2-3.文法等について

(1).データベースやスキーマ等を作成する「create」文

各種作成(create)

■ スキーマ作成

- `create schema [スキーマ名];`

名前を付ける際には予約語は使えないで注意(例: userやcase)

■ DB作成

- `create database [DB名];`

■ table作成

- `create table [スキーマ名].[table名]([カラム名].[型], ~);`
詳細は後ページ

(2).データベースやスキーマの変更する「alter」文

各種変更(alter)

■ スキーマ関連

[名前変更]

- `alter schema [旧スキーマ名] rename to [新スキーマ名];`

■ DB変更

[名前変更]

- `alter database [DB名] rename to [新DB名];`

[DBの設定変更]

- `alter database [DB名] set [パラメータ名] to [値 or default];`

- `alter database [DB名] reset [パラメータ名];`

※ パラメータはDB設定に関するものでSET文(後述)で変えられるもの
toは = でもよい。

■ Table変更

- 後述

(3). テーブルの作成と関連するもの

テーブル作成は以下のように細かい指定ができる。

また、テーブルには制約、シーケンス、インデックスを付与できる。

テーブル作成の詳細(create table)

- `create table [スキーマ名].[table名]([カラム名][型][制約], ~)`

記載しなくてよい
(その場合はpublicに作成)

自由

表の要素のタイトルに近い

カラムを複数作る場合
に以降を指定。

名前	年齢	部署
山口	28	開発
岩村	31	運用

右の表なら、名前と年齢
と性別の3つがそれぞれ
必要になる

- 型について、以下に代表例を記載。

型	説明	備考
smallint	整数	-32768から+32767
integer (int)	整数	-2147483648から+2147483647
bigint	整数。	-9223372036854775808から9223372036854775807
numeric	小数点も含めた数値	小数点前までは131072桁、小数点以降は16383桁
varchar(n)	文字数指定の文字。	nは数値を入れる。
text	文字数制限なしの文字。	
timestamp	時刻型。	
boolean	論理型。	True or False

例)

```
create table person ( name text, age smallint, department varchar(2) );
```

(3)-a.データをカラム(列)に登録する際の条件を指定する制約

制約とは

- カラムに入れる値には様々な制限をつける事ができる

制約名	制約の記述	説明
主キー(プライマリーキー)	primary key	1つのテーブルに1つしかつけられない。 一意性かつnot null。 Indexも付与される。 複数列を1つの主キーともできる。 Pkeyとも呼ばれる。
一意性制約(ユニークキー)	unique	重複した値の登録を許容しない Indexも付与される
非NULL制約(Not Null制約)	not null	空(null)の値を許容しない
外部キー制約	references [テーブル名] [カラム名]	他のテーブルにある情報しか登録できない
検査制約	check	条件を満たした値以外登録できない

例)

```
create table person (
    id integer primary key ,
    name text unique not null,
    age smallint check 22 > age ,
    department varchar(2) references department( name ) );
```

制約の別名と変更

- 制約には名前をつけられる
これによりログがわかりやすくなる他、
自動で生成されるindexも同様の名前となる。

テーブル
情報

例)

```
create table person (
    id integer primary key ,
    name text constraint uni_null unique not null,
    mail text ,
    age smallint check 22 > age ,
    department varchar(2) references department( name ) );
```

\$d
テーブル "public.test_const"
列 | 型 | 照合順序 | Null 値を許容 | デフォルト
-----+-----+-----+-----+-----
id | integer | notnull | |
name | text | not null | |
mail | text | | |
age | smallint | | |
department | varchar(2) | | |
Constraintをつづないと以下になる
"person_name_key" UNIQUE CONSTRAINT, btree (name)

- 制約を外す場合は以下。

```
alter table [テーブル名] drop constraint [制約名];
alter table [テーブル名] alter column [カラム名] drop [制約の記述];
```

(3)-b.番号を払い出すシーケンス

シーケンスとは(create sequence)

- 自動で連番を払い出してくれるもの。
Pkey(主キー)と相性が良い。

`create sequence [シーケンス名];`

シーケンスでは
今の番号を取り出す`currval`関数、
次の番号を取り出す`nextval`関数、
値をセットする`setval`関数がある。

例)

```
create sequence test_seq;
select setval('test_seq', 1);
select currval('test_seq');
select nextval('test_seq');
```

上記関数を使い、データ登録時に自動で入れる。

`insert into person values (nextval('test_seq'), 'yamaguchi');`

`nextval`を使わず、テーブル作成時にデフォルトで指定するやり方もある
`create table person (`
 `id integer primary key default nextval(test_seq),`
 `name text);`

デフォルト設定は`insert`時に何も指定されていない時に自動で値を入れる。
Pkey以外にも、データを登録した日時や、更新日時をデフォルトで入れる使い方もある。

上記に記載の通り、テーブル作成時に列(カラム)のデフォルト値の指定ができる。(併せて覚えておこう)
指定方法は「列名型 default 値」と記載する。(シーケンスだけでなく、固定値を指定できる。)

シーケンスの設定詳細

- シーケンスは開始時点と終了時点を指定できる他、
インクリメントする数値の数なども変更できる

`create sequence [シーケンス名]`
`increment [インクリメントする数]`
`minvalue [最小値] maxvalue [最大値]`
`start [開始値]`

例)

```
create sequence test_seq increment 100
maxvalue 10000 start 100;
```

(3)-c.検索速度を向上させるインデックス

インデックスとは(create index)

【indexとは】

- ・ テーブルの特定カラムにindexを付与すると検索処理(select)が早くなる
一方で容量も食ったり、沢山つけすぎるとメンテナンスにも時間を要する
ようになるため、つければよいというわけではない。
- ・ `create index [index名] on [テーブル名] using [index種] ([カラム名])`

btree, hash, gist等、様々なものがあるが、基本はbtreeでOK
何も指定しない場合はbtreeになる
※ 青色は記載なくともOK

indexは書籍の索引のようなもの、とイメージすると良い。

例えば読者が技術書を読んでいたとして、あるキーワードについて調べたい時、索引がないと最初のページから確認していくしかないが、索引があれば時間をかけずにどの辺にあるのかがわかり、高速に目的の情報を取得できる。

インデックスの種類について

- ・ 参考までにindexの種類を記載する。

Indexの種類	得意分野・ユースケース
btree	Where句での一致条件や大小比較、最大最小などで使う
hash	長い文字列データに向いている。 一致条件で使用する。
GIN	配列やJSONなど、複数の要素を持っているものに向いている。 テキスト検索で「～を含む」の条件など。
Gist SP-Gist	テキスト検索、範囲データ、幾何データ(地図データ)の 重なり条件等に向いている。
BRIN	履歴データの日付や連番等に向いている。

indexの管理には以下のように reindex 文や cluster 文といった特別な文法を使用する。

Indexの管理(reindex,cluster)

【Indexの作り直し】

`reindex [index/table/database/system] [対象物]`

※ 主な使い道はindexの設定を変更した場合や
indexが壊れた場合等。書き込みロックがかかるので注意

【Indexに合わせたテーブルの並び替え(整頓)】

`cluster [テーブル名] using [index名]`

※ 検索ヒット速度が速くなったり、内部の検索処理が変化する
読み書き両方のロックがかかるので注意。
(vacuum full(後述)では並び替えは行わない為、配置法で差異あり)

※ ロックについて後述。

なお、テーブルの変更も alter 文を使用する。

以下のように様々な変更ができる。

テーブルの変更処理(alter table)

- テーブル名の変更

`alter table [旧テーブル名] rename to [新テーブル名];`

- テーブル内のカラム名の変更

`alter table [テーブル名] rename column [旧カラム名] to [新カラム名];`

- テーブル内のカラムの追加

`alter table [テーブル名] add column [カラム名] [型];`

- テーブル内のカラムの削除

`alter table [テーブル名] drop column [カラム名];`

- テーブル内のカラム定義変更

`alter table [テーブル名] alter column [カラム名] type [変更する型];`

(4). ビューとマテリアライズドビュー

ビューについて

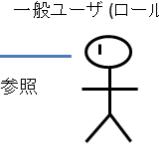
- テーブルとよく似たものに「ビュー」が存在する。
- ビューとは、実体を持たないテーブル。
ビューを確認しようとすると、事前定義していたテーブルからの参照文(select文)が実行され、テーブルがあるかのようにみせかける。
よって元テーブルの値が変わると、ビューの値も変わる。
- 上記のメリットは、目的の異なる同じようなテーブルを複数持つことがなくなる
(=容量節約・管理の簡易化)事や、テーブルから見える範囲を絞り込んで
ロール(=アカウント)によってアクセスできる範囲を絞り込む(=セキュリティ強化)
があげられる。
- 以下が文法。
`create view [ビューネ名] as select文 ;`
 上記をした後であれば、以下のようにしてデータ確認ができる
`select * from [ビューネ名];`
 ※ select文の詳細はDML基礎章、ロールはDCL章を参照。

【従業員テーブル】

id	名前	年齢	部署
3542	山口	28	開発
9999	岩村	31	運用

【一部情報_従業員ビュー】

id	部署
3542	開発
9999	運用



セキュリティの関係上、一般ユーザにはidと
部署名しか見せたくない

ビューを活用！

マテリアルズドビューについて

- マテリアルズドビュー(マテビュー)とは実体を持つビュー(=テーブル)で、
作った時のデータを持つテーブルとなる。
その時のデータとなる為、元ネタが更新されても併せて更新されることはない。
- ユースケースとしてはその時の状態をとておきたいスナップショット的な
使い方や、複雑な計算・計算式に使う値があまり変わらないものを毎回
計算するのではなく計算結果を一時保存しておく(=これにより計算を高速化
する)、といった使い方があげられる。
- 以下が文法。

`create materialized view [ビューネ名] as select文 ;`

自動更新されないのでデータが古くなる。その場合は以下を実施(作り直し)。

`refresh materialized view [ビューネ名];`

なお、マテビューを作った後は、マテビューに対してupdate文等で更新はできない。

(5).メモや補足情報を残せるcomment文

comment文を使えばDB内にメモを残しておくことが可能。

本来はテーブル名等にはわかりやすい名前を付けるべきだが、わかりづらい名前の場合等に有効である。

各種データに説明文を追加(comment)

- DBやテーブルを作成後に、そのテーブルの説明等を記載しておくことが可能。

■コメント追加

- comment on database/table/view/index/sequence [対象名] is '[コメント]' ;
comment on column [table名].[カラム名] is '[コメント]' ;

【例：テーブルにコメントした場合】

```
# \d+
          リレーション一覧
スキーマ | 名前 | 型 | 所有者 | サイズ | 説明
-----+-----+-----+-----+-----+
public | test_table | テーブル | postgres | 8192 bytes | test comment
```

■コメント削除

- comment on column [table名].[カラム名] is null ;

なお、コメントの削除は後述するdrop文ではなく、nullを指定するので注意すること。

(6).各種削除をするdrop文

各種削除(drop)

- これまで作成や変更例を記載してきたが、削除は「drop」を使用する。

■DB/schema/table/sequence/index削除

- drop database/schema/table/sequence/index/view
[対象名] ;

if exists句について

- drop文は存在しないテーブルを指定するとエラーとなる。
仮に存在しないテーブルを指定してもエラーとしたい場合、if exists文を使用する。

`drop table if exists table名 ;`

【例】

`drop table if exists nothing_table ;`

該当テーブルが存在しなくて
もエラーとならない。

3章 DCLについて

本章ではロール(アカウント)ごとに権限を付与する DCL とその関連情報について触れていく。

3-1. 本章の概要

はじめに本章で触れる概念について整理する。

DCL 概要

【登場概念】

タイトル
ロール
所有者(オーナー)

【実施概要】

大分類	概要
ロール操作	create alter drop
権限操作	grant revoke
所有者変更	alter

章のタイトルは「DCL」だが、正確には DCL に該当するのは grant, revoke 文のみである。

本構文が扱う「ロール」についても本章で説明する為、DCL 以外の文法も登場する点はご了承いただきたい。

3-2.DCLとロールについて

postgresにはロールという概念が存在する。

DCLとロールについて

- DCLはDBへのアクセス権の制御等を行う言語。
→アクセス権の制御はロールで行う。

【ロールとは】

- 一般的にいう「ユーザ(アカウント)」に近い。
- DDL演習ではpostgresユーザ(管理者ユーザ)で触っていたが、
上記は何でもできてしまう為、不特定多数の人でDBを触る場合は権限を
絞った別アカウントを作るのが好ましい



昨今、社内からのデータ漏洩も発生しており、見える情報の適切な管理は管理者の重要な役割と言える。

DBは顧客情報などを入れる場合が多く、全ての情報が、全担当から見えるのが望ましくない事がある。

その場合、ロールを活用するというのも1つの手段となる。

上記例では開発部門が使う develop ロールと、運用部門が使う operation ロールを分けたイメージである。

運用担当は顧客からの問い合わせを受けてデータ状態等を確認する事はあっても、DBの操作のような本格対処は開発部門に任せる、という運用を想定する。

その場合、運用担当は顧客の情報が入っているDBへの select (情報取得) のみで権限は良いはずである。

一方開発者は顧客環境で何か不具合が発生した場合、本格対処で最悪データを加工したり削除する必要もある為、insert/update/delete といった権限が必要である。

このように権限を分ける事で、運用担当が誤ってデータ加工をしてしまわないようにガードをかけることができる。

上記はDBを例に記載しているが、テーブルごとにも権限を付与可能である。

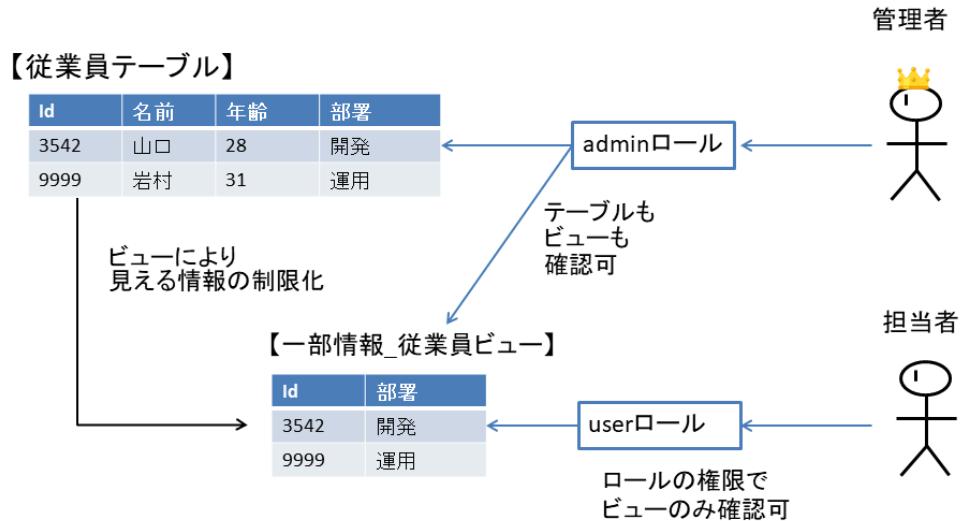
なお、上記例のようにロールは正確には「ユーザ」+「グループ」のような概念である。

その点は後述する為、まずはユーザのようなものと理解してもらえば、と思う。

またセキュリティ観点では1つのテーブルの見える範囲を絞る「ビュー」とあわせて使われることも多い。ロールはテーブルそのものへの権限を設定する点がビューとの違いとなる。

ビューとロール

- ビューとロールは組み合わせて使う事も多い。
以下に例を記載する。



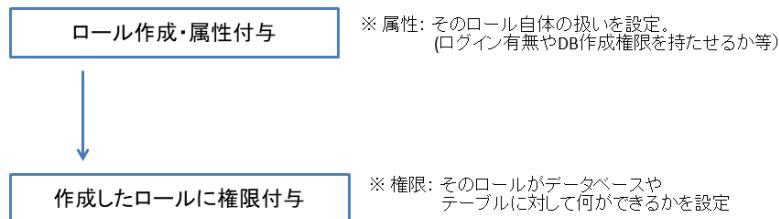
3-3.文法等について

(1).ロールの操作に関する流れとロールの作成

まずロールに関する操作の全体的な流れを記載する。

ロールを作る・設定する手順について

- 大きく以下の2段階がある



以下で、手順の1つ目のロールの作成について記載する。

ロールの操作(create/alter/drop role)

【ロールの作成】

※ 正確にはDCLではない

- `create role [ロール名] [属性];`

【属性変更】

- `alter role [ロール名] [属性]`

【属性】

属性名	記載法	説明	無効化記載法
ログイン	login	DBへの接続を可能にする	nologin
パスワード付与	password 'パスワード'	ログイン時にパスワードを要求される	password null
スーパーユーザ	superuser	ログイン以外に何でもできる	nosuperuser
DB作成	createdb	DB作成が可能になる	nocreatedb
ロール作成	createrole	ロール作成が可能になる	nocreaterole

【ロール削除】

- `drop role [ロール名];`

(2). ロールへの権限付与(grant文)

ロールへの権限付与には grant 文を使用する。grant 文では「DB やテーブル等に対するロールへの権限付与」と「ロールにロールを参加させる」という2つの事ができる。

テーブルに対する権限付与(grant)

【ロールに権限を付与することができる】

- `grant [権限] on [テーブル名] to [ロール名];`

【全ロールにまとめて権限付与】

- `grant [権限] on [テーブル名] to public;`

【全テーブルにまとめて権限付与】

- `grant [権限] on all tables in schema [スキーマ名] to [ロール名];`

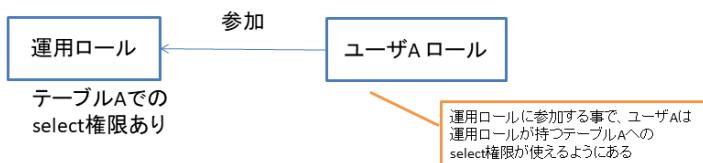
【権限】(テーブルに対するもの)

※ 他にもDBやシーケンス等につけられるものも存在する

記述法	備考
<code>select</code>	—
<code>insert</code>	—
<code>update</code>	—
<code>delete</code>	—
<code>truncate</code>	—
<code>all (privileges)</code>	Privilegesはなくても良い

ロールに(別)ロールを参加させる(grant)

- テーブル等への権限だけでなく、あるロールを別のロールに参加させることができる。
(=ロールをグループのような使い方で使える)
- 別ロールに参加させると、参加したロールの権限が継承される。



- 文法は以下。
`grant 参加するロール to 参加させるロール;`

(3).ロールからの権限削除(revoke 文)

ロールからの権限削除には revoke 文を使用する。

ロールの権限削除(revoke)

- 付与した権限を取り除くにはrevoke文を使う。

【テーブルに付与した権限解除】

revoke [権限] on [テーブル名] from [ロール名];

【ロールに参加させたロールの解除】

revoke 参加させたロール from 取り除くロール ;

(4).所有者(オーナー)という概念について**DB/スキーマ/シーケンス/テーブルの所有者について**

※ 正確にはDCLではない

- DB/スキーマ/シーケンス/テーブルを作成すると、作成したロールが「所有者」となる。
最初は所有者以外は作ったDBを操作(名前の変更等、alter)できない。
(ただしスーパーユーザはどれでも触れる)
- 所有者の変更は以下で実施。
alter database/schema/sequence/table [各種名] owner to [新たな所有者];
※ indexの所有者はtableの所有者変更で自動で切り替わる

[補足] 所有者の確認方法

- DB
- ¥I
- スキーマ/シーケンス/テーブル
- ¥d

4章 TCLについて

本章ではトランザクションと、その制御を行うTCLについて記載する。

4-1. 本章の概要

はじめに本章で触れる概念について整理する。

TCL 概要

【登場概念】

タイトル
トランザクション
ロック
デッドロック
トランザクション分離レベル

【実施概要】

大分類	概要
トランザクション操作	begin commit rollback(abort) savepoint
ロック操作	lock table select for update select for share
トランザクション分離 レベルの設定/確認	set show

本章のタイトルは「TCL」だが、一部「TCL」外のSQLが登場する為、注意してほしい。

具体的には「select for update」「select for share」はDMLに該当するSQLとなる。

また「set」「show」文も「TCL」とは異なる。

4-2. トランザクションとその操作について

まず、トランザクションについて説明する。

トランザクションとは

- トランザクションとは複数の処理を1つにまとめたもので、全処理成功か、全処理失敗かの二択のみの動きとなる。
→データの矛盾発生を抑える。
- 例えば、以下のようにAからBに入金したとする。
その場合、以下の2処理を実施するが、片方だけ成功してしまった場合、データに矛盾が発生する。
 - ① Aからは金額を引き、
 - ② Bには金額を増やす必要がある。



このように一部だけ成功が許されない処理は
1つの処理にまとめる(=トランザクション)

1 トランザクションは全て成功するか、全て失敗するかの二択である事は ACID 特性でいうところの A(Atomicity, 原子性)となる。

ACID 特性は、データ管理の信頼性を上げる為のトランザクションに求められることを示したもので、重要な概念となる為、理解しておけると良いだろう。

参考として、ACID 特性の説明を以下に記載する。

ACID の要素	説明
Atomicity (原子性)	トランザクションは全て成功するか、全て失敗するかの二択である事。
Consistency (一貫性)	データに矛盾が発生しないように整合性がとられる事。 処理するデータが特定のルールや制約に守られること。
Isolation (隔離性)	実行中のトランザクションが他のトランザクションから影響を受けない事。 後述するトランザクション分離レベルと関連。
Durability (永続性)	障害などが発生してもトランザクションにて完了させた記録が失わない事。 WAL(Write-Ahead Logging)という仕組みにより実現させる。

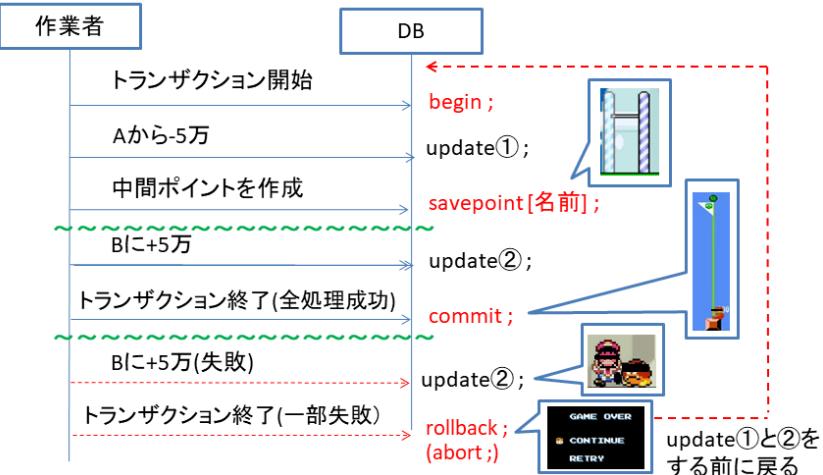
余談だが、上記は IPA の基本情報技術者試験や応用情報技術者試験でも登場するワードである。

その観点でも覚えておけると良いだろう。

トランザクションの基本的な操作は以下の通りである。

トランザクション操作(1)

- 前ページの例をベースに記載する。



操作を行う構文としては大きく「begin」「savepoint」「commit」「rollback(abort)」がポイントとなる。

「begin」は必須、「commit」「rollback(abort)」はどちらか必須で必要、「savepoint」は任意で使用する。

トランザクションの操作は2D マリオをイメージするとわかりやすい。

「トランザクションとは」に記載したように、1トランザクションの中身の処理は全て成功か、全て失敗の二択である。

よって途中まで成功していても、その後に失敗した場合は全て失敗とみなす。(= マリオで道中で自機を失うイメージである。)

上記で記載の図では緑の波線で、すべて成功したケースと、全て失敗とみなすケースをそれぞれ記載している。
(上記図は中間ポイントを作成後、update②の処理が成功するケースと失敗するケースに分岐している)

途中で処理に失敗した場合、一番最初からまた巻き戻されるとつらい場合は「中間ポイント」を作ることができる。これを作つておけば処理に失敗しても（マリオが敵にあたって自機を失っても）開始時点に戻る事なく、中間ポイントから改めて再トライが可能となる。

マリオを違う点は中間ポイントは複数作れる点となる。

全ての処理が終わった場合は明示的に commit を実行する。

これにより処理が完了し、savepoint も削除される。

よって次、同じ処理をする場合(=同じステージをする場合)、またはじめからのトライとなる。

なお、マリオの場合は時期を失った場合、自動で中間ポイントから始まるが、トランザクションにおいては復旧する中間ポイントを指定する必要がある。「rollback(abort)」を実行すると、そのトランザクションはまたはじめとなる。

これはマリオでいうところの全機失った場合のゲームオーバーに近い。(ゲームオーバーとなった場合は、中間ポイントをとっても、次トライする場合にはステージの最初からとなる)

前述の中間ポイントへの復旧方法は以下のように「rollback to savepoint」を実行する。

また、明示的にトランザクション内でsavepointを消す場合は「release savepoint」を実行する。

トランザクション操作(2)

- 途中で処理失敗した際にsavepointに戻りたい場合は
以下を実施
 rollback to savepoint [名前];
※ rollbackだけだとトランザクションが終了し、
savepointは使用できなくなるので注意。
- コミットやロールバックをするとsavepointは自動で消えるが、
明示的に消す場合は以下。
 release savepoint [名前];

補足となるが、これまで記載してきた「begin」から始まり「commit」あるいは「rollback(abort)」で終わるトランザクションは「明示的トランザクション」と呼ばれる。

明示的にトランザクションをかけない場合も、実際にはトランザクションが動いている点は理解しておけると良い。

明示的トランザクションと暗黙的トランザクション

- begin; 等は明示的なトランザクションであり、通常のinsertやupdateも暗黙的にトランザクションをかけている。
- トランザクションは txid_current() 関数でIDを確認できる
(トランザクションの度にトランザクションIDを払い出している)

【例】`select txid_current();`

- 本番環境等でinsertやupdate文を打つ(失敗できない)という時はbeginをかけてから実施すると良い。(やらかしていたら戻せる)
ただし、更新処理をしてからはロックがかかるのでなるべく早めに開放することが求められる。

なお、上記にも記載の通り、明示的トランザクションは本番環境での更新作業等の巻き戻しにも活用できる。

事前にバックアップをとっておき、それを使って巻き戻す(リストア)案もあるが、時間がかかる可能性がある。

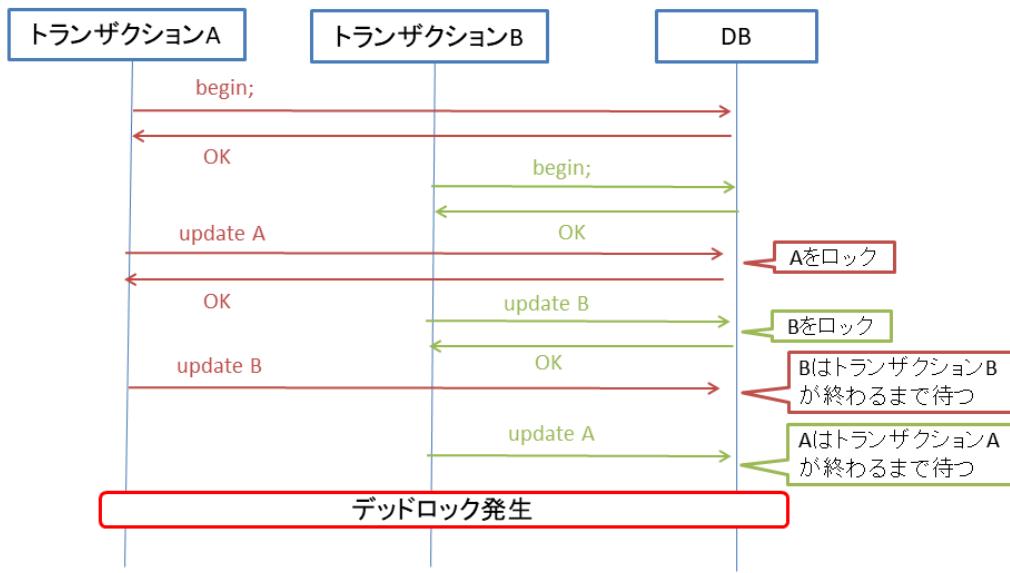
本番環境で何か手を入れる際にはバックアップをとっておくことは重要だが、ユーザのアクセスが少なかつたり直ぐに終わる更新処理であれば、即座に変更処理を実行前に戻せる明示的トランザクションは有効である。

4-3.ロックについて

トランザクションは作り方によっては「デッドロック」を発生させるので注意が必要である。

デッドロック

- ロックした箇所に後発で処理をしようとすると、前のトランザクション（ロックを取得しているトランザクション）が完了するまで待たされる
→ 2つのトランザクションがお互いにロックしている箇所を触ってしまい、お互いに待ちとなってしまうケースがある（デッドロック）



デッドロックを発生させないようにするには、各トランザクションの処理の流れを意識する（正しく設計する）ことが重要である。

上記と合わせて、明示的にロックをかける方法についても説明する。

明示的にロックをかける方法には大きく以下の2種がある。

- ① テーブルに対してロックをかえる(lock table文)
- ② 行に対してロックをかける(select for update, select for share文)

明示的ロックについて①(lock table)

トランザクション中は更新したレコード(行)についてロックがかかる
ロックがかかると該当テーブルに対してupdate等の処理ができなくなる。

以下のようにすることで、トランザクション中に明示的にロックすることも可能。
ロックはトランザクションを終了すると解除される。

この仕組みを活用することで、例えばトランザクション内で複数処理・複数テーブルを操作する場合に1つ目の処理が終わった後に2つ目でデッドロックが発生してトランザクション失敗する、というようなことを防止できたり、メンテナンス等で処理予定のテーブルに対し、予定外の処理が入り、待たされる時間の発生等を減らすことができる。

■ テーブルロックの場合(access exclusiveモード)

lock table [テーブル名] in lockmode [ロックモード]

→
なくても良い。指定しない場合は最も強いロック(access exclusive)となる
(該当テーブルにinsert/update/deleteに加え、selectもできなくなる)

ロックレベルの詳細はここでは触れないが、気になる方は以下を参照すると良い。

<https://www.postgresql.jp/document/14/html/explicit-locking.html>¹

1. <https://www.postgresql.jp/document/14/html/explicit-locking.html>

明示的ロックについて②(select for update/share)

明示的ロックではテーブルごとだけでなく、テーブルの行(レコード)ごとにロックも可能。

■ 行ロックの場合 (row shareモード)

[排他ロック]

`select ~ for update;`

他トランザクションからselectは可能だがinsert/update/delete等は不可。

排他ロックがかかった行に後から、select for update/select for share不可。

※ この辺は正確にはロックモードにより異なる(共有ロックも同様)

例) `select * from 部署 where id = 3 for update; # idが3のレコードのみロック`

[共有ロック]

`select ~ for share;`

他トランザクションからselectは可能だがinsert/update/delete等は不可。

共有ロックがかかった行に後からselect for updateは不可、select for shareは可。

ただし他トランザクションと共有ロックが被った箇所はinsert/update/delete等不可できない

【排他・共有ロックの関係性について】

		先発	
		排他	共有
後発	排他	×	×
	共有	×	○

排他ロック・共有ロックを既にロックがかかった行に後からかけられるかどうかは、【排他・共有ロックの関係性について】表をみるとわかりやすいのではないかと思う。

あるトランザクションが先に、特定行にロックをかけた場合 (=先発) でも共有ロックなら、別のトランザクションが後から同じ行に共有ロック (=後発) をかけられる。

それ以外の組み合わせは後発は同じ行にロックはかけられない。

4-4. トランザクション分離レベルと禁止観点について

トランザクション分離レベルはACID特性のI(隔離性)に関連する概念である。

トランザクション分類レベルは一言でいうと、トランザクションの動作モードと思ってもらってよい。どのモードで動かすかにより、非推奨の動作（各トランザクション分離レベルで禁止される観点）をどこまで防げるかが変わる。

まずはトランザクション分離レベルの名前（4つのモード）と、各禁止観点について説明する。

トランザクション分離レベルと観点

- 各トランザクション分離レベルとは規制の厳しさに応じて4種ある
 - リードアンコミットド(read uncommitted)
 - リードコミットド(read committed)
 - リピータブルリード(repeatable read)
 - シリアルライザブル(serializable)
- 各種レベルで禁止される4つの観点は以下。

名前	説明
ダーティリード	他トランザクションから、 コミットしていない 変更情報が見えてしまう
反復不能読み取り	同時にトランザクションが動いている時、片側が コミットした結果が、もう一方の未コミットトランザクションから見えてしまう (更新 が見えてしまう)
ファントムリード	同時にトランザクションが動いている時、片側が コミットした結果が、もう一方の未コミットトランザクションから見えてしまう (追加/削除 が見えてしまう)
直列異常	複数のトランザクションを正常にコミットした結果が、1つずつ順番に実施した場合の結果と異なる

[例] 直列異常の場合

数値
10
20

- トランザクションAで、sum(数値)(=30)をinsert
- トランザクションBで、sum(数値)(=30)をinsert
- トランザクションAをコミット
- トランザクションBをコミット
→この時、トランザクションBの値は1で追加した30の分を見れていない
トランザクションAが完了した後にBを実施した場合と結果が異なる

各分離レベルと、禁止される観点をまとめたものが以下である。

トランザクション分離レベルと禁止観点の関係

- トランザクションと禁止観点の関連は以下のようになる。

	リードアンコミットド (read uncommitted)	リードコミットド (read committed)	リピータブルリード (repeatable read)	シリアルizable (serializable)
ダーティリード	×	○	○	○
反復不能読み取り	×	×	○	○
ファンタムリード	×	×	△	○
直列異常	×	×	×	○

○: 発生しない(安全)

△: SQL標準的に発生は許容されるが、postgresでは発生しない。

×: 発生する可能性あり

※ postgresではread uncommittedも指定できるが、動作はread committedとなる

[補足] トランザクション分離レベルはserializableを選んでおけば安泰?
一般的に厳しい分離レベルにすればするほど処理は遅くなっていく為、APLの動作と相談。

分離レベルの変更にはset文を使用し、確認はshow文を使用する。

トランザクション分離レベルの変更方法(set・show)

- トランザクション分離レベルの設定方法は以下。

① beginする際に指定。

`begin transaction isolation level [分離レベル];`

※ [分離レベル]は、分離レベル名の英語表記を記載。

なお、この指定だとトランザクション完了後に元に戻る

② beginのみ指定。

`set transaction isolation level [分離レベル];`

※ トランザクション完了後に元に戻る

- 分離レベルの確認

`show transaction isolation level;`

[補足] 永続的にトランザクション分離レベルを変えるには？

postgresql.confの設定を変更する。あるいはalter databaseを使用する。

SET文・SHOW文は一時的に設定を変更・設定確認するコマンドで、

トランザクション分離レベル以外にも変更できるものがある。

5章 DMLの基礎について

本章ではデータを操作するDMLの基礎について記載する。

5-1. 本章の概要

はじめに本章で触れる概念について整理する。

DML 概要

【DML 基礎】

大分類	概要	詳細
データ投入	insert	insert insert~select
	copy	
データ更新	update	
	returning	
データ削除	delete	
	truncate	
データ取得	select	
条件式等	where	and/or/not
	演算子や関数、キャスト変換、null取得について	算術関数 集約関数 is null/ is not null 文字列関数 日付/時刻関数 データ型書式設定関数 その他関数
	文字検索	like, ilike similar to 正規表現
	distinct	
	group by	asも含む
	having	
	order by	
	offset/limit	
	between	

【DML 応用】

大分類	概要	詳細
条件式等	結合(join)	inner join outer join cross join
	集合演算	union/ union all except intersect
	副問合せ	select ~ from (select) 相間副問合せ exists in some/any
	再帰処理	with with recursive
	SQLの動作解析	explain/ explain analyze
	window関数	over (partition by ~) over (order by ~)
	UPSERT	on conflict (excluded) merge

DMLは種類も多い為、基礎編と応用編にわけている。

5章の基礎編では赤枠で囲った部分を対象とする。

5-2.データの登録/変更/削除/取得について

(1).データの登録(insert文/copy)

データ登録をするには以下2種の構文がある

- ① insertを使う
- ② copyを使用する

データの挿入1(insert)

- データの投入にはinsert文を使用する

`insert into [テーブル名]([カラム名1], ~) values (値1, ~);`

複数カラム入れる場合は
続ける。

カラム名と値は同じ並びにする
省略も可。その場合はテーブルの左の列から
入れる形となり、足りない箇所はnullとなる

- selectの結果を挿入することも可能。

`insert into [テーブル名] select ~`

例) 単一レコードを入れる場合

`insert into person(age,name) values (23, '木村');`

例) 複数レコードを入れる場合

`insert into person(age,name) values (24, '石井'), (25, '鈴木');`

例) selectの結果を入れる場合 # 登録済のデータを再度投入する
`insert into person select * from person;`

データの挿入2(copy)

- データの投入にはcopyコマンドも使用できる

copyコマンドはcsvファイル等のファイルに書かれたデータをテーブルに入れる際に使う。

`copy [table名] from [ファイル名] delimiter [区切り文字] [ファイルの型]`

- 同様にcopyコマンドで今テーブルに入っているデータを抜き出す事も可能

`copy [table名] to [ファイル名] delimiter [区切り文字] [ファイルの型]`



- 上記を使って一定数入った情報をDBから抜き出し、そのcsvをもとに
excel等で加工してデータを増やして挿入したりするとデータを増やしやすい

例) csvで出力

`copy person to '/var/tmp/output.csv' delimiter ',' csv;`

例) csvで挿入

`copy person from '/var/tmp/input.csv' delimiter ',' csv;`

(2).データの更新と更新箇所の確認(update文/returning句)

データの更新には update 文を使用する

データの更新(update)

- 条件に一致するレコードの更新を行う

update [テーブル名] set [カラム名1] = [値1], ~ where 条件式
 update [テーブル名] set ([カラム名1], ~ = [値1], ~) where 条件式

複数のカラムを更新する場合のみ指定 なくてもよい
句については
後述

※ 記載はどちらでもOK

条件式を指定しない場合は該当のカラムのデータがすべて
更新される

Id	name	age	
1	山口	28	Where句で条件指定することで一部のみ変更する where区を入れないと、ageのレコードがすべて 書き換わる
2	鈴木	29	
3	佐藤	35	

なお、update 時には通常は「UPDATE 1」等の更新数しか表示されない。

正しい場所が更新されたか確認する際には returning 句をつけておくと、更新した行(レコード)が表示されるため、便利である。

更新箇所のチェック(returning)

- 更新箇所のチェックには更新箇所をselectする方法もあるが、returning句を使用すると便利。

update [テーブル名] set [カラム] = [値] where [条件]
 returning [取得レコード]

【例】

```
update person set name = '上原' where id = 1 returning *;
```

Id	name	age
1	山口	28
	上原	
2	鈴木	29
3	佐藤	35

【結果】		
id	name	age
1	上原	28

※ returning句を使わないと“UPDATE 1”
とかしかでない

※ returning句はinsertやdeleteでも使用できる

(3).データの削除(delete/truncate 文)

データ削除をするには以下2種の構文がある

- ① delete 文
- ② truncate 文

データの削除(delete・truncate)

- 条件に一致するレコードを削除する
何も指定しない場合はテーブルの全てのレコードが削除される

`delete from [テーブル名] where 条件式;`

Id	name	age
1	山口	28
2	鈴木	29
3	佐藤	35

条件を指定することで一部のレコードを削除。
指定しないとテーブル内の全レコード削除。



- テーブル内の全レコードを削除する場合はdeleteではなく、truncateも使用できる。(こちらの方が早い)

`truncate [テーブル名];`

参考までにDDLで実施したdropとの違いを以下に記載する。

[参考] dropとdelete/truncate の違い

- dropは表ごと削除。
delete・truncateは表は残し、レコード(データ)のみ削除。

【dropの場合】

Id	name	age
1	山口	28
2	鈴木	29
3	佐藤	35



なし

※ 新たにデータ入れるなら
create tableで表の作り直しから。

【delete・truncateの場合】(全削除)

Id	name	age
1	山口	28
2	鈴木	29
3	佐藤	35



Id	name	age
1	山口	28
2	鈴木	29
3	佐藤	35

※ 新たにデータ入れるなら
insertすればすぐ入れられる
(createは不要)

`delete` と `truncate` の違いは速度の違い以外に、削除した際に物理容量が開放される(データの物理ファイルが小さくなる)かどうかという点もある。

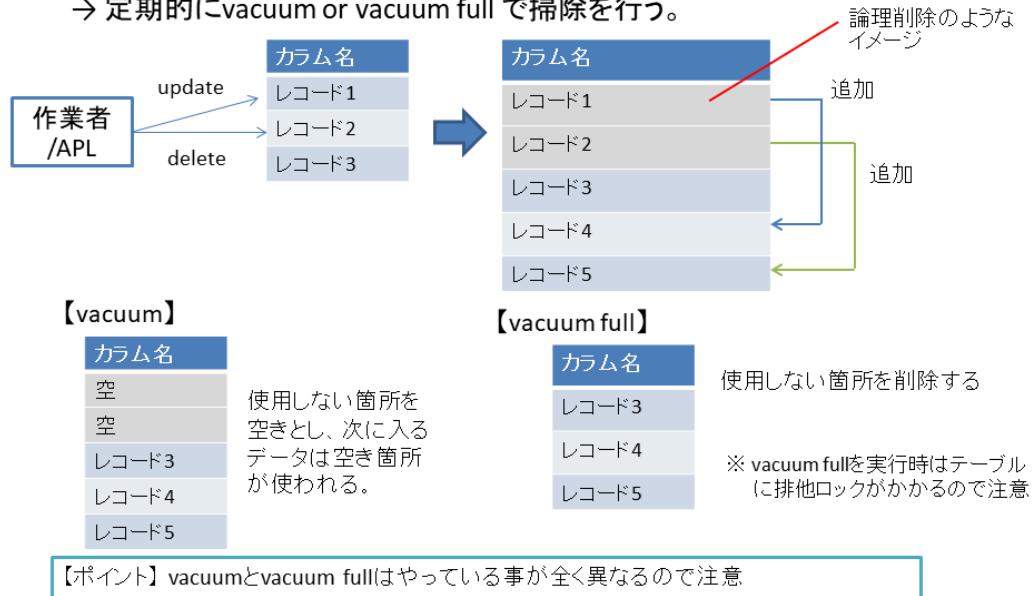
`postgres` は追記型アーキテクチャと呼ばれる動きをしており、`delete` 文では削除した箇所を「削除した」とマークして非表示にする (=本当の意味で削除していない) 動作をする為、`delete` でレコードを削除しても容量が空かない。

「論理削除」をしているようなイメージをしてもらえるとわかりやすいのではないか、と思う。

以下に参考として追記型アーキテクチャの説明と、`delete` で削除した場合に該当領域を再利用可能とする `vacuum` について記載する。

[参考] 追記型アーキテクチャと`vacuum`, `vacuum full`とは

- `vacuum`, `vacuum full` は DB 内の掃除をしてくれるもの。
- `postgres` は **追記型アーキテクチャ** であり、ほっておくとサイズが肥大化していく。
→ 定期的に `vacuum` or `vacuum full` で掃除を行う。



[参考] vacuumの文法について

- 以下のように実行する。

[vacuum]
· `vacuum [テーブル名];` # テーブル名を省略すると全テーブル対象。

[vacuum full]
· `vacuum full [テーブル名];` # `vacuum full`の実行。テーブル名省略時は同じ。

[備考]

通常vacuumはautovacuumという仕組みで定期実行する。よって「特定の時間帯に実施したい」等のごだわりがなければ、明示的に実行する機会は少ないかもしれません。

なお、vacuum文は標準SQLにはない。

vacuumもSQL対話モードで実行可能であるが、SQL標準にはない文法であり、DBを操作するというよりはpostgresを操作する際に覚えておきたいものとなる。

なおpostgresにはautovacuumと呼ばれる自動vacuum機能もある為、定期的な実行は本機能を使う事が基本となる。

また、vacuumは他にも様々なオプションがある事も留意しておいてほしい。(上記はシンプルな例で記載している。)

(4).データの取得(select文)

データの取得にはselect文を使用する。

データの取得(select)

- データの取得にはselect文を使用する

`select [カラム名1],~ from [テーブル名] where [条件式];`

特定カラムの情報をとる場合。
全カラムの情報をとる場合は '*' を指定。

条件式を付けない場合、
テーブルの全データを取得する

【person表】

Id	name	age
1	山口	28
2	鈴木	29
3	佐藤	35

【全データ取得】
`select * from person;`

【name列だけ取得】
`select name from person;`

【山口だけ取得】
`select name from person where id = 1;`

5-3.操作対象データの条件指定について

(1).条件の指定(where句)と演算子

条件の指定には where 句を使用する。

条件の指定と演算子(where)

- update,delete,selectでは条件式を指定できた。
条件を指定する場合はwhereを指定する。

where [カラム名] [演算子] [条件]

例) select * from person where id = 1;

※ 条件式は[and][or]を使い、複数条件指定が可能
[not]を使うと否定になる

【演算子】

演算子	説明
<	小なり
>	大なり
<=	以下
>=	以上
=	等しい
<> または !=	等しくない

演算子の種類と計算の優先度は以下の通り。

その他演算子

【算術演算子】

演算子	説明	使い方例	結果
+	和	1+3	4
-	差	3-1	2
*	積	3*2	6
/	商	5/2	2.5
%	剰余	5%2	1
^	累乗	2^3	8
/	平方根	/ 2	1.41321...
!	階乗	5!	120
@	絶対値	@ -5	5

【優先度】

演算子
累乗(^)
乗算(*), 除算(/), 剰余(%)
加算(+), 減算(-)
その他(@,!等)

↑ 高
↓ 低

※ 乗算より加算を優先したい等、上記優先度より先に優先したい場合は(カッコ)を使う

【日付/時刻演算子】

演算子	説明	使い方	結果
+	和	date '2021-10-16' + 7	2012-10-23
-	差	timestamp '2012-10-16 12:00' – interval '23 hour'	2012-10-15 13:00:00
*	積	interval '90 minutes' * 365	547:30:00
/	商	interval '60 minutes' / '25'	00:02:24

(2).各種関数と count,null の指定(is null)

postgres には様々な関数が用意されている。

関数とは、値を渡す事で特定の処理を行う便利な機能の事である。

わかりやすい例としては、特定の列の合計値(sum)や平均値(avg)、最大値(max)を出す等のものがある。

関数には数値型を扱う「算術関数」や文字列を扱う「文字列関数」、日付型を扱う「日付/時刻関数」の他、「集約関数」「データ型書式設定関数」等がある。

算術関数とキャスト変換について

- 数値型のデータに対して特定の演算を実施する関数(一部抜粋)。

関数名	説明	実行例	実行結果
dev(x,y)	x/yの整数商	dev(9, 4)	2
mod(x,y)	x/yの剰余	mod(9, 4)	1
floor(x)	Xより大きくなき最大の整数	floor(2.5)	2
ceil(x)	Xより小さくなき最小の整数	ceil(2.5)	3
round(x)	Xを整数値に四捨五入	round(2.5)	3
trunc(x)	Xの小数点を切り捨て	trunc(2.5)	2
trunc(x,y)	Xの小数点y桁以降を切り捨て	trunc(0.123, 2)	0.12
random()	0.0 <= x < 1.0 の範囲でランダムな値書き出し	random()	0.4601028
abs(x)	Xの絶対値	abs(-2.5)	2.5
sqrt(x)	Xの平方根	sqrt(2)	1.41321...
log(x)	Xの常用対数	log(1000)	3
pi()	円周率	pi()	3.145926...

【例】ランダムな整数の取得
select (random() * 100) ::int ;

random関数は小数の数値型。
→ 欲しいのは整数であり、小数点以下はいらない。
→ 整数型に型を変換する(キャスト変換)

書式は以下。
(値)::[変換したい型]

上記では random 関数が小数の数値を返すが整数が欲しい為、×100 をしている。

しかし元が小数(例 : 0.1111...)の為、×100 をしても「11.11...」となり、引き続き小数も含んだ値となる。

欲しい値が整数の場合、「round 関数」「trunc」関数等を使う手もあるが、ここではキャスト変換の説明をする為、::int としている。

キャスト変換とは元の型から、異なる型へ変換する事である。(今回の場合は小数型から整数型への変換。他の型もできる) キャスト変換をする際には一部情報が捨てられる事がある(今回は小数の数値が捨てられている)為、注意する事。

集約関数

- 複数の値を使用し、単一の結果を取得するもの。
- group by(後述)とあわせて使われることが多い。

関数名	説明
sum(カラム名)	指定された列の合計値を取得
avg(カラム名)	指定された列の平均値を取得
max(カラム名)	指定された列の最大値を取得
min(カラム名)	指定された列の最小値を取得。
count(カラム名)	指定された列の数を取得。 詳細は後述。

【personテーブル】

部署名	メンバ	年齢
総務	A	30
開発	B	25
経企	C	41
開発	D	35

【例】全社員の平均年齢を取得
select avg(年齢) from person ;

集約関数の中でも count は(カッコ)内に*(アスタリスク)を入れるか、列名を入れるかで動作が変化する。

ヒット数の取得(count)

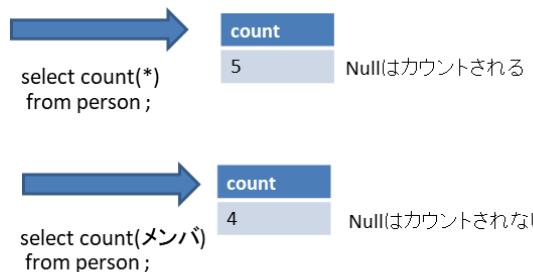
- 条件にヒットするレコードがどの程度あるか確認するには「count」を使う。

select count([カラム名]) from [テーブル名] where [条件式] ;

※ カラム名の部分は「*」とするとnullもカウント。
カラム名を指定するとnullはカウントしない

【personテーブル】

部署名	メンバ	年齢
総務	A	30
開発	B	25
経企	C	41
開発	D	35
営業	(null)	(null)



nullは扱いがやや特殊であり、nullを探したい場合は「is null」構文を使用する。

Nullレコード取得について(is null)

- Nullとは、データが入っていない状況の事を言う
(空白スペースとは異なる)
→ nullを取得するには is null を使うと良い

where [カラム名] is null ;

is not null でnullじゃないデータをとることも可能

【例】

select * from person where 年齢 is null ;

部署名	メンバ	年齢
総務	A	30
開発	B	25
経企	C	41
開発	D	35
営業	(null)	(null)

文字列関数

- 数値だけでなく文字列に対しても演算する関数がある。

関数名	説明	実行例	実行結果
a b	文字列aとbを連結する。 片方が文字データ型以外でも指定可。	'post' 10.0	post10.0
lower(a)	文字列aを小文字に変換。	lower('aBc')	abc
upper(a)	文字列aを大文字に変換。	upper('aBc')	ABC
substring(a from x for y)	文字列aのx文字目からy文字取り出す。	substring('ABCDE' from 3 for 3)	CDE
replace(a,x,y)	文字列a中のxをyに変換。	replace('abc','b','X')	aXc
octet_length(a)	文字列aのバイト数を取得	octet_length('あああ')	9 ※ UTF8の場合
length(a)	文字列aの文字数を取得	length('あああ')	3
trim(b from a)	文字列aから文字列bを除外	trim('ab' from 'abcab')	c
lpad(a,x,y)	文字列aをx桁になるまでyを追加。 (先頭に文字追加) パディング。	lpad('10',5,'0')	00010

日付/時刻関数

- 日付/時刻関数には、現在の時刻取得から、日付・時刻データに対し特定の演算を行うものや、特定の部分だけを抽出する関数等がある
※ 時刻データは文字列ではなく、時刻型で持つておきたい。
でないと下記のような関数は使えない

関数名	説明	時刻取得タイミング
current_date	現在の日時	トランザクション開始時点
current_timestamp	現在の日時と時刻	トランザクション開始時点
current_time	現在の時刻	トランザクション開始時点
now()	日時と時刻	トランザクション開始時点
transaction_timestamp()	日時と時刻	トランザクション開始時点
statement_timestamp()	日時と時刻	SQL文の開始時点
clock_timestamp()	日時と時刻	関数実行時点
timeofday()	日時と時刻	関数実行時点
age(a, b)	aとbの時刻・日付差分を取得	-
age(a)	現在の日時(午前0時)とaの差分取得	-
date_part('field', a)	aから部分フィールドの時刻データ取得	-
date_trunc('field', a)	aから部分フィールドに指定した精度で時刻データを取得	-
extract(field from a)	aから部分フィールドの時刻データ取得	-

Fieldはsecond(秒), minute(分), hour(時), day(日), month(月), year(年)等

データ型書式設定関数

- 多様なデータ型を整形された文字列に変換したり、特定のデータ型に変換する事ができる
- キャスト変換と異なり、書式を規定することができる

関数名	説明
to_char(a , format)	様々なデータ型であるaを、書式に合わせて文字列型に変換する
to_date(a , format)	文字列aを書式に合わせて日付型に変換
to_number(a , format)	文字列aを書式に合わせて数値型に変換
to_timestamp(a, format)	文字列aを書式にあわせてタイムスタンプ型に変換

【日付/時刻データ用の書式】

書式	説明
YYYY	年(4桁以上)
MM	月番号(01-12)
DD	月の日付(01-31)
HH24	時(00-23)
MI	分(00-59)
SS	秒(00-59)
MS	ミリ秒(000-999)
TZ	大文字による時間帯名
AM,PM	ピリオドなしの午前・午後の指定
Mon	短縮形の月名
Dy	短縮形の曜日名

【数値データ用の書式】

書式	説明
9	指定された桁数での値
0	前にゼロが付いた値
.	小数点
,	桁区切り
PR	負の値の角括弧表示

【例】

```
select to_char(current_timestamp, 'YYYYMMDD');
結果: 20221002
select to_char(10,'00999.99');
結果: 00010.00
select to_char(2.5, '9.9PR');
結果: 2.5
select to_char(-2.5, '9.9PR');
結果: <-2.5>
```

その他、便利な関数として generate_series 関数がある。

この関数自体は下記の通り、連続したデータを取得するだけだが、上手く使うと大量のダミーデータ（試験データ）を生成できる。

その他関数(generate_series)

- 指定した値で連続したデータを取得できる。

【例】1～5の数値を取得

```
select generate_series(1, 5);
```

- 3つ目の数値を指定することでインクリメントする値も指定できる。

【例】0～10で偶数の値を取得

```
select generate_series(0, 10, 2);
```

- 演算子やクロスジョイン(後述)と insert into [テーブル名] select 文と組み合わせる事で試験データの作成に非常に役立つ。

【例】本日から1日ずつのデータを5日分取得

```
select (current_date + generate_series(0, 4));
```

(3). 文字の部分一致検索(like, ilike/similar to/正規表現)

文字列検索は演算子(= 等) でも検索できる。

ただしこれは「完全一致」検索となる。

部分一致検索には以下に紹介する構文を使用する。

文字検索(like, ilike)

- likeを用いる事で特定パターンの文字列を取得できる
ilikeでは大文字・小文字を区別せずに検索を行う
`where [カラム名] like/ilike [条件]`

【条件】

- 使う文字は以下。
 - _ : 任意の1文字に該当
 - %: 任意の文字(複数文字)に該当
- 上記の置き場にあわせて、前方一致、中間一致、後方一致がそれぞれ行うことができる

【例】

- ① `select * from test where name like 'AB%';`
- ② `select * from test where name like '%BC%';`
- ③ `select * from test where name like '%EF';`
- ④ `select * from test where name ilike 'AB%';`

【testテーブル】

name
Abc
bCD
CDEF
ABCD
BCDE

④はこれらがヒット

文字検索(similar to)

- similar toはlikeで使えた「_」「%」に加え、以下メタ文字も使用し、複雑なパターンマッチングが実行できる

`where [カラム名] similar to [条件];`

メタ文字	説明
	二者択一
*	直前の項目の0回以上の繰り返し
+	直前の項目の1回以上の繰り返し
?	直前の項目の0回もしくは1回の繰り返し
{m}	直前の項目のm回の繰り返し
{m,n}	直前の項目のm回以上かつn回以下の繰り返し
()	()内の1つの論理項目とみなす
[]	[]内のいずれか1文字に一致するパターン

【testテーブル】

name
Abc
bCD
CDEF
ABCD
BCDE

【例】特定のカラムからヒットする情報取得

`select * from test where name similar to '%(EF|GZ)'`

【例】簡易な試験

`select 'ABEF' similar to '%(EF|GZ)' ;`

※ like、正規表現(後述)でも上記簡易確認は可。ヒットすればtrue、しなければfalseが返ってくる

文字検索(正規表現)

- likeやsimilar toより強力なパターンマッチングが可能。
- likeでの「%」は「*」と表記し、前方/後方一致には「^」「\$」を用いる等、記法は異なる。
- 正規表現でのマッチングは以下演算子を用いる。

演算子	ヒット条件	真を返す例
~	正規表現に一致 大文字・小文字は区別する	Select 'abcd' ~ '^a.*';
~*	正規表現に一致 大文字・小文字は区別しない	Select 'abcd' ~* '.*D\$';
!~	正規表現に不一致 大文字・小文字は区別する	Select 'abcd' !~ '.*c\$';
!~*	正規表現に不一致 大文字・小文字は区別しない	Select 'abcd' !~* '^B.*';

その他、参考としてJSONデータの検索手法を以下に記載する。

JSONは1つのJSON単位で例に記載しているように様々な情報を入れられる。(attributeのように入れ子構造も可能である)

JSONデータをどのようにDBに登録するかはアプリケーションの動作にもよるが、そのまま入れた場合は下記例に記載している情報が1フィールドに入るイメージとなる。

この場合、1フィールドのなかに複数の要素が入るが、その要素で検索したいというケースがあり得る。(例えば全レコードからJSON内のidが1000のデータのみ取得したい、といったケース)このようなJSONならではのケースにも対応できるように以下のような文法が用意されている。

[参考] その他、文字検索

- ここではJSONデータの取得について記載する。
- JSONとはシステム間でのやり取りに必要なデータを電文内に入れて使用する為のもの。書式は以下のようなイメージ。

【JSON例】

```
{ "id" : "3542",
  "name" : "yamaguchi"
  "attribute": {
    "dept" : "FN1BU",
    "rank" : "7" }
```

- postgresにはjsonb型というJSONを入れておくデータ型もある。
左記の単位で1レコードとした際に、様々なjsonデータから特定の情報を見つけるには「->」や「->」「@>」等特殊な記法が必要となる。

【例】「name: yamaguchi」を探す場合

```
select * from sample_json where (json->>'name') = 'yamaguchi' ;
```

【例】「dept: FN1BU」を探す場合

```
select * from sample_json where (json->'attribute'->'dept') = '"FN1BU"'
select * from sample_json where json->'attribute'@>'{"dept":"FN1BU"}' ;
```

※「->」は1段目にあるものを取得、「->」は2段以降にネストされている物を取得する際に使用。
「@>」はkeyとvalueの組み合わせが一致しているカラムをとる時に使うもの。

上記は「sample_json」テーブルで、jsonカラムにjsonデータが入っていた場合を想定。

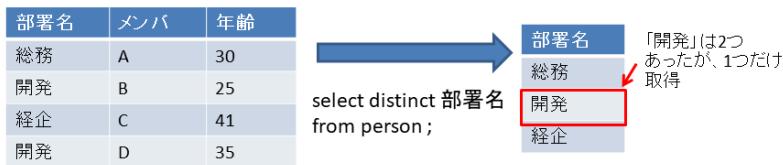
- 他にも配列や列挙型で独自の取得法(記法)がある

(4).重複排除と列の別名付与(distinct/distinct on/as)

重複排除には distinct を使うが、distinct と distinct on で動作が異なる。

重複排除(distinct)

- 重複したデータを取得したくない時は「distinct」を使用する
`select distinct [カラム名] from [テーブル名];`



- distinct on (カラム名)とした場合、重複排除の上、ソートして出力
`select distinct on ([重複排除して並び替えるカラム名])[表示するカラム]
from [テーブル名];`



また、distinct は他の集約関数などと一緒に使う事もできる。

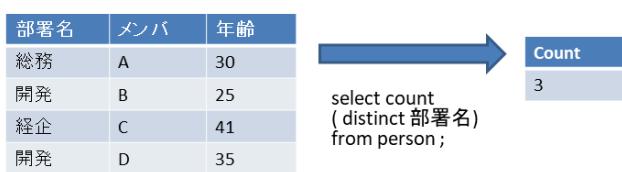
例として count と一緒に使った場合を以下に記載する。

なお as を使う事で、結果出力時の列名を任意の列名とすることができる。

distinctとcountの組み合わせと結果出力(as)について

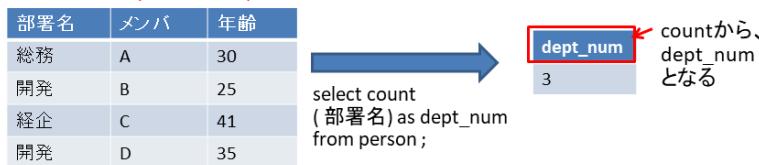
- countとの組み合わせも可能。

`select count(distinct [カラム名]) from [テーブル名];`



- countや集約関数は取得する際に「count」や集約関数名がカラム名となる。
asを使い、わかりやすい名前に変える事が可能。

`select count([カラム名]) as [別名] from [テーブル名];`



(5). グルーピングとその検索について(group by/having)

group by は指定したカラムの情報毎にグループを作り、集約関数を使って集計する。

例えば以下の例のように「部署名」列でグルーピングした場合、部署名列にある値（総務、開発、経企）ごとにグループを作り、グループごと(それぞれの部署ごと)にヒット数集計 (count)や平均値取得(avg)等の集約関数の処理を実施できる。

グルーピング(group by)

- ・ 指定したカラムの情報毎にグルーピングしてまとめる
- ・ 集約関数(前述)と組み合わせて使う事が多い
 where [条件式] group by [指定カラム] ;
 ※ whereはなくても可

【personテーブル】

部署名	メンバ	年齢
総務	A	30
開発	B	25
経企	C	41
開発	D	35

- ① select 部署名 from person group by 部署名 ;
 ② select 部署名, count(メンバ), avg(年齢)
 from person group by 部署名 ;

【①の結果】

部署名
総務
開発
経企

部署ごとに出力
なので重複なく
部署が表示される

【②の結果】

部署名	count	avg
総務	1	30
開発	2	30
経企	1	41

集約関数を使う事で
グループごとの結果
がわかる

group by とあわせてよく使うものに「having」がある。

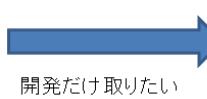
グルーピングした後の条件指定(having)

- group byでグループ化した結果に対して条件指定を行う場合は「having」を使用する。
指定可能なカラムはgroup byで指定した列化、集約関数を用いた列となる
～ group by [カラム名] having [条件式]；

【group by の結果】

部署名	count	avg
総務	1	30
開発	2	30
経企	1	41

```
select 部署名, count(メンバ), avg(年齢)
from person group by 部署名;
```



部署名	count	avg
開発	2	30

開発だけ取りたい
select 部署名, count(メンバ),
avg(年齢) from person group by
部署名 having 部署名 = '開発'；

whereとhavingの違い

- whereもhavingも条件を指定する構文。
使い分けは以下
 - where : group by でまとめるより前の条件としたい場合
 - having : group by でまとめた後の条件としたい場合

部署名	メンバ	ランク	年齢
総務	A	7	30
開発	B	8	25
経企	C	5	41
開発	D	6	35
開発	E	7	27

【例】
部署ごとに平均年齢を出したい。
ただし、ランク7～9だけで見たい。
→そのまま部署でグルーピングしてしまうと
条件に合わない人も入ってしまう。
グルーピングの前に7～9のランクを条件にする(where)

その上で今回は開発だけ見たい。
→グルーピングした上で、必要な部署だけ見る。
そのままだと他部署も出てしまう。
グルーピング後の結果に対して条件指定。(having)



部署名	count	avg
開発	2	26

```
select 部署名, count(メンバ), avg(年齢) from person
where ランク between 7 and 9 group by 部署名
having 部署名 = '開発'；
```

(6).並び替えと取得範囲の絞り込みについて(order by/offset/limit/between)

並び替えには order by を使用する。

上記とあわせて情報取得範囲の絞り込みで limit や offset が使われることがある。

並び替えと取得数について(order by と offset/limit)

- 指定したカラムでソート(並び替え)をして結果を出力する
`where [条件式] order by [指定カラム] [asc/desc];`
※ ascは昇順(デフォルト:指定なし可)。descは降順。whereはなくても良い

id	部署
1	開発
2	運用
3	総務

select * from 部署
order by id desc ;

id	部署
3	総務
2	運用
1	開発

- 取得数は「1つだけ取得する」とか「結果の3つ目から取得する」といった指定も可能。(offsetとlimit。片方だけ指定もできる)
 - offsetが「何件目から取得する」という指定(開始は0行目判定なので注意)
 - limitが「何件取得する」という指定

`where [条件式] order by [指定カラム] offset [件数] limit [件数];`

offset番号	id	部署
0	1	開発
1	2	運用
2	3	総務

select * from 部署
offset 1 limit 1 ;

id	部署
2	運用

offset 1なので2つ目から取得。
limit 1なので1つだけ結果を取得。

order by と distinct onとの違いは、重複排除した上で並び替えをするかどうか。

検索結果をいくつとるか、という範囲指定ではなく、検索条件の範囲指定には between を使用する。

範囲の取得(between)

- X～Y の値のレコードを取得する、といった範囲を指定したデータ取得が可能
`where [カラム名] between [範囲X] and [範囲Y];`

id	部署
1	開発
2	運用
3	総務
4	経企
5	営業

select * from 部署
where id between 2 and 4 ;

id	部署
2	運用
3	総務
4	経企

6章 DMLの応用について

本章ではデータを操作するDMLの応答について記載する。

6-1. 本章の概要

まず、本章で触れる概念について整理する。

本章は5章の続き（応用）となる為、SQLに慣れていない人はまずは5章から確認する事を推奨する。

DML概要

【DML基礎】

大分類	概要	詳細
データ投入	insert	Insert Insert~ select
	copy	
データ更新	update	
	returning	
データ削除	delete	
	truncate	
データ取得	select	
条件式等	where	And/or/not
	演算子や関数、キャスト変換、null取得について	算術関数 集約関数 is null/ is not null 文字列関数 日付/時刻関数 データ型書式設定関数 その他関数
	文字検索	like to similar to 正規表現
	distinct	
	group by	asも含む
	having	
	order by	
	offset/limit	
	between	

【DML応用】

大分類	概要	詳細
条件式等	結合(join)	inner join outer join cross join
	集合演算	union/ union all except intersect
	副問い合わせ	select ~ from (select) 相間関連問い合わせ exists in some/any
	再帰処理	with with recursive
	SQLの動作解析	explain/ explain analyze
	window関数	over (partition by ~) over (order by ~)
	UPSERT	on conflict (excluded) merge

6章の応用編では赤枠で囲った部分を対象とする。

本章で扱う構文は複雑になりがちで、初見だと中々理解しきるのは難しいかもしれないが、現場でもよく使われる物もある為、シンプルな例から要点をおさえながら、問題とあわせて確認をして理解を深めてほしい。

6-2.結合について

本節では結合について扱う。

結合と言にいっても、下記のように結合には複数の種類がある。

結合とは

- 2つ以上の表を見て、条件に合うデータを取得する場合には、表をつなげて(合体させて)データをとる
→この合体処理を「**結合(join)**」と呼ぶ。
- 結合タイプには大きく3種がある
 - inner join
 - outer join
 - cross join
- また、結合条件にも3種ある
 - on
 - using
 - natural

まずは結合タイプの1つ、inner joinについて説明する。

内部結合(inner join)とは

- 結合条件が一致するカラムのみ取得する結合
- ```
select [カラム名] from [テーブル名1] inner join[テーブル名2]
on [テーブル1].[カラム名]=[テーブル2].[カラム名];
※書式は「on」の場合で記載。Usingとnaturalは省略
```

- select \* from users inner join dept on users.部署id = dept.部署id ;
- select \* from users inner join dept using (部署id) ;
- select \* from users natural inner join dept ;
- select \* from users, dept where users.部署id = dept.部署id ;

※ 上記は全て同じ結果となる

【usersテーブル】

【deptテーブル】

| 【usersテーブル】 |    |      | 【deptテーブル】 |     |
|-------------|----|------|------------|-----|
|             | 名前 | 部署id | 部署id       | 部署名 |
| 1           | 鈴木 | 2    | 2          | 総務  |
| 2           | 佐藤 | 3    | 3          | 運用  |
| 3           | 田中 | 4    | 4          | 営業  |
| 4           | 渡辺 | 5    |            | 開発  |

赤枠の箇所のみ取得

次に結合条件について説明する。

## 結合条件とは

- それぞれ以下のような使い方となる

| 結合条件    | 説明                                             |
|---------|------------------------------------------------|
| on      | 結合する条件となるカラム名をそれぞれ指定                           |
| using   | 結合する条件となるカラム名が同じ場合に使用。<br>指定した列のみをイコール条件として処理。 |
| natural | 結合する条件となるカラム名が同じ場合に使用。<br>全ての列をイコール条件として処理。    |

### 【onの例】

- select \* from users inner join dept on **users.部署id = dept.部署id**;

2つのテーブルのそれぞれどのカラムが同じ意味となるのか(イコール条件となるのか)指定している。

### 【usingの例】

- select \* from users inner join dept using (**部署id**);

2つのテーブルに同じ「部署id」が存在する場合に、それがイコール条件になるという記載。

### 【naturalの例】

- select \* from users natural inner join dept;

2つのテーブルに同じカラム名が存在する場合に、それがイコール条件になるという記載。

(同名カラムは全てイコール条件として判定する為、ターゲットとなるカラム名を指定する箇所がない)

on は結びつける 2 つのテーブルの列をそれぞれ指定しているが、using は 1 つしか指定していない。

これは言い換えれば 2 つのテーブルで同じ意味を持つ列が、同じ列名である時にのみ使用できるものといえる。

natural に関しては列名を指定していない。これは 2 つのテーブルで同じ名前の列が全て同じ意味である時に使用できる。

今回の例では「person テーブル」の部署 id と、「dept」テーブルの部署 id が同じ名前の為、これを条件に自動で結び付けている。

dept テーブルの部署 id が、id だった場合、person テーブルは人の id であるのに対し、dept は部署の id という意味合いとなるが、natural を使うと自動で結び付ける為、異なる意味合いでの結合がされてしまう。

なお、以降の説明は on のみとする。

次に outer join について記載する。

これは inner join + イコール条件に一致しない一部の情報も取得する、という動きのものとなる。

outer join には「left」「right」「full」の3種類があるが、これによりイコール条件に一致しないどの情報を取得するか指定できる。

## 外部結合(outer join)とは

- 結合条件に一致したレコードだけでなく、一致しないレコードも取得できる
- [left][right][full]の3種がある

```
select [カラム名] from [テーブル名1]{left|right|full} outer join [テーブル名2]
on [テーブル1].[カラム名]=[テーブル2].[カラム名];
```

- select \* from users left outer join dept on users.部署id = dept.部署id;

※ 上記はleftを指定しているが、right、fullを指定した場合も同じ

【usersテーブル】

| ID | 名前 | 部署ID |
|----|----|------|
| 1  | 鈴木 | 2    |
| 2  | 佐藤 | 3    |
| 3  | 田中 | 4    |
| 4  | 渡辺 | 5    |

【deptテーブル】

| 部署ID | 部署名 |
|------|-----|
| 1    | 総務  |
| 2    | 運用  |
| 3    | 営業  |
| 4    | 開発  |

leftかfullを指定した場合、取得

rightかfullを指定した場合、取得

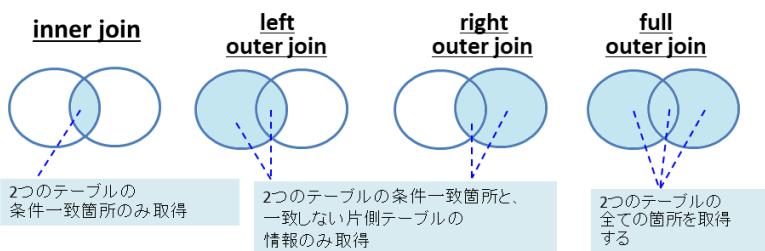
2つのテーブル関係をベン図で示した時に、inner join と outer join のそれぞれの取得範囲を以下に記載する。

上段はこれまでの復習だが、下段のようなケースもあり得る。

これらはどのように取得するか、次ページに答えを記載するので少し考えてみよう。

### 内部結合と外部結合の関係性について①

- 内部結合・外部結合のできる事をベン図で表すと  
以下のような形となる。  
(水色箇所が取得箇所)



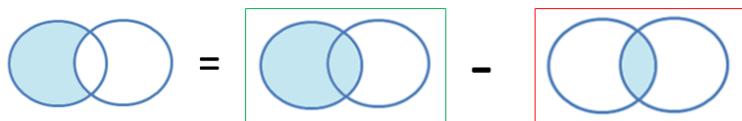
- では以下の部分の取得はどのように取得するか。



それぞれの取得方法は以下となる。

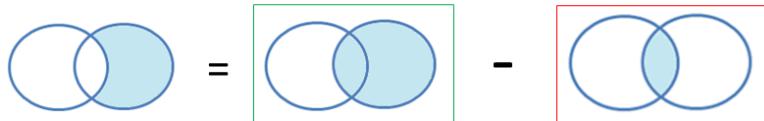
### 内部結合と外部結合の関係性について②

- 以下はouter joinに条件を加える事で実現できる。



```
select * from users left outer join dept on users.部署id = dept.部署id
where dept.部署id is null;
```

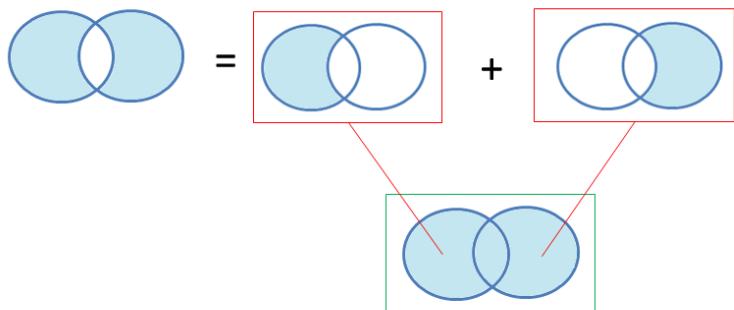
緑の結果から、イコール条件に合致しない  
(外れている部分はnull)箇所を指定  
= イコール条件を取得対象外とする



```
select * from users right outer join dept on users.部署id = dept.部署id
where users.部署id is null;
```

### 内部結合と外部結合の関係性について③

- 以下はouter joinに条件を加える事で実現できる。



```
select * from users full outer join dept on users.部署id = dept.部署id
where dept.部署id is null or users.部署id is null;
```

すべて取得した状態から、  
それぞれのテーブルの欠けた部分を  
足し算(or条件指定)する。

最後に紹介する結合タイプがcross joinである。

cross joinは記載方法が2種ある。

## 交差結合(cross join)とは

- 複数表の直積(全ての組み合わせ)を取得する
  - 例えば10行同士の表をcross joinすると、 $10 \times 10$ で100行取得される
  - whereで条件を指定することでinner joinと同じ結果を得ることができる

- select \* from users cross join dept ;
- select \* from users, dept ;

※ 上記は全て同じ結果となる

【usersとdeptのcross join結果イメージ】

| Id | 名前 | 部署id | 部署id | 部署名 |
|----|----|------|------|-----|
| 1  | 鈴木 | 2    | 1    | 総務  |
| 1  | 鈴木 | 2    | 2    | 運用  |
| 1  | 鈴木 | 2    | 3    | 営業  |
| 1  | 鈴木 | 2    | 4    | 開発  |
| 2  | 佐藤 | 3    | 1    | 総務  |
| 2  | 佐藤 | 3    | 2    | 運用  |

… (以下省略)

試験用の環境を整える為、ダミーデータを多く用意する際にはcross joinとgenerate\_series関数を使用すると便利である。

ここまで3種の結合タイプについて触れたが、初見では中々理解しづらいのではないか、と思う。

以下に参考として、3種の結合タイプの違いを記載(外部結合はfull指定動作で記載)するので、参考として欲しい。

## [参考] 3種の結合タイプ比較

| 結合テーブル1                                                                                                                                  |         | 結合テーブル2  |         | 結果  |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
|------------------------------------------------------------------------------------------------------------------------------------------|---------|----------|---------|-----|---|-----|---|-------------------------------------------------------------------------------------------------------------------------------------------|----------|---------|---|-----|---|-----|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|---------|----------|---------|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|
| <table border="1"> <tr> <th>ColumnA</th><th>ColumnB</th></tr> <tr> <td>1</td><td>AAA</td></tr> <tr> <td>2</td><td>BBB</td></tr> </table> | ColumnA | ColumnB  | 1       | AAA | 2 | BBB | × | <table border="1"> <tr> <th>ColumnA'</th><th>ColumnC</th></tr> <tr> <td>2</td><td>bbb</td></tr> <tr> <td>3</td><td>ccc</td></tr> </table> | ColumnA' | ColumnC | 2 | bbb | 3 | ccc | = | <table border="1"> <tr> <th>ColumnA</th><th>ColumnB</th><th>ColumnA'</th><th>ColumnC</th></tr> <tr> <td>2</td><td>BBB</td><td>2</td><td>bbb</td></tr> </table>                                                                                                                                                                      | ColumnA | ColumnB | ColumnA' | ColumnC | 2 | BBB | 2 | bbb |   |     |   |     |   |     |   |     |   |     |   |     |
| ColumnA                                                                                                                                  | ColumnB |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 1                                                                                                                                        | AAA     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 2                                                                                                                                        | BBB     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| ColumnA'                                                                                                                                 | ColumnC |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 2                                                                                                                                        | bbb     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 3                                                                                                                                        | ccc     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| ColumnA                                                                                                                                  | ColumnB | ColumnA' | ColumnC |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 2                                                                                                                                        | BBB     | 2        | bbb     |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| ※ ColumnAとColumnA'を結合条件とする                                                                                                               |         |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| <table border="1"> <tr> <th>ColumnA</th><th>ColumnB</th></tr> <tr> <td>1</td><td>AAA</td></tr> <tr> <td>2</td><td>BBB</td></tr> </table> | ColumnA | ColumnB  | 1       | AAA | 2 | BBB | × | <table border="1"> <tr> <th>ColumnA'</th><th>ColumnC</th></tr> <tr> <td>2</td><td>bbb</td></tr> <tr> <td>3</td><td>ccc</td></tr> </table> | ColumnA' | ColumnC | 2 | bbb | 3 | ccc | = | <table border="1"> <tr> <th>ColumnA</th><th>ColumnB</th><th>ColumnA'</th><th>ColumnC</th></tr> <tr> <td>1</td><td>AAA</td><td>-</td><td>-</td></tr> <tr> <td>2</td><td>BBB</td><td>2</td><td>bbb</td></tr> <tr> <td>-</td><td>-</td><td>3</td><td>ccc</td></tr> </table>                                                            | ColumnA | ColumnB | ColumnA' | ColumnC | 1 | AAA | - | -   | 2 | BBB | 2 | bbb | - | -   | 3 | ccc |   |     |   |     |
| ColumnA                                                                                                                                  | ColumnB |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 1                                                                                                                                        | AAA     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 2                                                                                                                                        | BBB     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| ColumnA'                                                                                                                                 | ColumnC |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 2                                                                                                                                        | bbb     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 3                                                                                                                                        | ccc     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| ColumnA                                                                                                                                  | ColumnB | ColumnA' | ColumnC |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 1                                                                                                                                        | AAA     | -        | -       |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 2                                                                                                                                        | BBB     | 2        | bbb     |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| -                                                                                                                                        | -       | 3        | ccc     |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| ※ ColumnAとColumnA'を結合条件とする<br>また完全外部結合を例とする                                                                                              |         |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| <table border="1"> <tr> <th>ColumnA</th><th>ColumnB</th></tr> <tr> <td>1</td><td>AAA</td></tr> <tr> <td>2</td><td>BBB</td></tr> </table> | ColumnA | ColumnB  | 1       | AAA | 2 | BBB | × | <table border="1"> <tr> <th>ColumnA'</th><th>ColumnC</th></tr> <tr> <td>2</td><td>bbb</td></tr> <tr> <td>3</td><td>ccc</td></tr> </table> | ColumnA' | ColumnC | 2 | bbb | 3 | ccc | = | <table border="1"> <tr> <th>ColumnA</th><th>ColumnB</th><th>ColumnA'</th><th>ColumnC</th></tr> <tr> <td>1</td><td>AAA</td><td>2</td><td>bbb</td></tr> <tr> <td>1</td><td>AAA</td><td>3</td><td>ccc</td></tr> <tr> <td>2</td><td>BBB</td><td>2</td><td>bbb</td></tr> <tr> <td>2</td><td>BBB</td><td>3</td><td>ccc</td></tr> </table> | ColumnA | ColumnB | ColumnA' | ColumnC | 1 | AAA | 2 | bbb | 1 | AAA | 3 | ccc | 2 | BBB | 2 | bbb | 2 | BBB | 3 | ccc |
| ColumnA                                                                                                                                  | ColumnB |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 1                                                                                                                                        | AAA     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 2                                                                                                                                        | BBB     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| ColumnA'                                                                                                                                 | ColumnC |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 2                                                                                                                                        | bbb     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 3                                                                                                                                        | ccc     |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| ColumnA                                                                                                                                  | ColumnB | ColumnA' | ColumnC |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 1                                                                                                                                        | AAA     | 2        | bbb     |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 1                                                                                                                                        | AAA     | 3        | ccc     |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 2                                                                                                                                        | BBB     | 2        | bbb     |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 2                                                                                                                                        | BBB     | 3        | ccc     |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |
| 行ごとに全組み合わせを表示                                                                                                                            |         |          |         |     |   |     |   |                                                                                                                                           |          |         |   |     |   |     |   |                                                                                                                                                                                                                                                                                                                                     |         |         |          |         |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |

参考として、join をしないといけないテーブル設計をする理由について記載する。

### [参考] joinが必要となるテーブル構成について

- なぜjoinしないといけないようなテーブル構成なのか。  
Joinしなくても良いように1つのテーブルに様々なデータを入れてしまえば良いのではないか。  
→あまりに色々なデータを1つのテーブルに入れると1つ変えただけだとデータ矛盾が発生する。  
なるべく1つだけ変えれば良いように作りたい。  
→「正規化」という考え方。

| ID | 名前 | 部署ID | 部署名 |
|----|----|------|-----|
| 1  | 鈴木 | 2    | 総務  |
| 2  | 佐藤 | 3    | 運用  |
| 3  | 田中 | 4    | 営業  |
| 4  | 渡辺 | 5    | 開発  |
| 5  | 山口 | 5    | 開発  |
| 6  | 伊藤 | 2    | 総務  |

運用部署に変更になったので更新する

| ID | 名前 | 部署ID | 部署名 |
|----|----|------|-----|
| 1  | 鈴木 | 2    | 総務  |
| 2  | 佐藤 | 3    | 運用  |
| 3  | 田中 | 4    | 営業  |
| 4  | 渡辺 | 5    | 開発  |
| 5  | 山口 | 5    | 運用  |
| 6  | 伊藤 | 2    | 総務  |

- 一方で正規化しすぎると  
AP観点では毎回joinしないといけず  
使いづらい。  
→joinは1つのテーブルから情報をとるより重めの処理になりがち。  
→正規化すれば良い、という問題ではない。(設計観点の話)

部署名だけ変更すると、部署IDが矛盾する  
(本来3に変えないといけない)  
→情報を沢山入れれば入れるだけ、こういった矛盾が発生しやすくなる

上記例はid 5を部署名を運用とした事で、運用の部署IDが3と5で2つになってしまった例である。

部署IDもあわせて直せば良いのでは?と思うかもしれないが、1つのテーブルに情報を入れると複数の直すべき点が発生する傾向が高く、データ矛盾につながる可能性が高い。

その為、正規化を行い、矛盾が発生しにくいテーブル構成とする必要がある。(=結果、joinが必要なテーブル構成となる事)

なお、正規化には第1~第5正規化、ボイスコッド正規化がある。

あまり正規化をしすぎると使いづらい他、処理的にも重くなる可能性がある。(json処理は重めである)

設計観点の話とはなるが、「(正規化を最後まで)やれば良い」という単純な話ではなく、データをどれだけ入れるかや機能の性能要件なども考慮して、どこまで正規化するのか考えながらテーブル設計を行う必要がある。

## 6-3.問い合わせ結果の結合について

表からの取得結果2つを以下機能を使って演算(集合演算)する事ができる。

- union(和集合:表からの取得結果2つの足し算)
- except(差集合:表からの取得結果2つの引き算)
- intersect(積集合:表からの取得結果2つの共通箇所抜き出し)

### 2つのselectの結果をまとめる(union)

- 2つのselectの結果をまとめると以下がある
  - union : 重複排除
  - union all : 重複も含める

`select * from [テーブル] union (all) select * from [テーブル];`  
※ 抽出列数とデータの型は合わせる事

【運用部テーブル】

| id | 名前 |
|----|----|
| 1  | 鈴木 |
| 2  | 佐藤 |

【開発部テーブル】

| id | 名前 |
|----|----|
| 1  | 井上 |
| 2  | 佐藤 |

select \* from 運用部 union all select \* from 開発部;

unionの場合は赤枠は表示されない(重複排除)

### 2つのselectの差を取得(except)

- 2つのselectの結果の差を取得するものにexceptがある  
oracleだとminusと呼ばれる
- `select * from [テーブル名] except select * from [テーブル名];`  
※ 1つ目の結果から、2つ目の結果を除く

【例】  
開発部でランク7～9をとりたい。  
が、30歳以上は除きたい。  
(① - ②)

①

| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 総務  | A   | 7   | 30 |
| 開発  | B   | 8   | 25 |
| 経企  | C   | 5   | 41 |

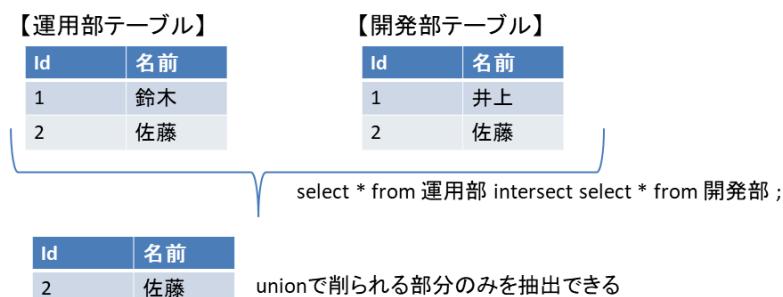
②

| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 開発  | D   | 6   | 35 |
| 開発  | E   | 7   | 27 |

select \* from person where 7 <= ランク >= 9  
expect select \* from person where 年齢 < 30;

## 2つのselectの共通部分を取得 (intersect)

- 2つのselectの結果で重複箇所のみ取得するものにintersectがある  
`select * from [テーブル] intersect select * from [テーブル];`  
※ 抽出列とデータの型は合わせる事



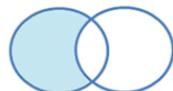
これまで触ってきた union, except, intersect も join と同様に、取得範囲をベン図で表す事ができるが、下記に記載の通り、観点がやや異なる為、注意してほしい。

## 各集合関数をベン図で表すと？

- 集合関数もjoinと同様にベン図で表す事ができる。  
※ joinは2つの情報をイコール条件で結び付けていたが、集合関数は  
イコール条件で結び付けずに演算する為、観点は変わる点は  
留意してほしい。



union: 2つの取得結果を合わせて表示する



except: 最初に取得したものから、  
後から取得したものを持った結果を  
表示する



intersect: 2つの取得結果の共通点を表示する

## 6-4.副問合せについて

ここからは副問合せについて説明する。

副問合せ（サブクエリ）とは、SQL文のなかにSQLを入れ子で記載する記法を指す。

### (1).副問合せの基本的な記法

副問合せの最も一般的な書き方は select ~ from ( select ~ ) 文である。

文法からもわかるように、SQL（最初のselect文）のなかに、SQL(2回目のselect文)が入っている。

## 副問合せ

- selectの中にselectを入れる（入れ子にする）事が可能（=副問合せ）

`select * from (select ~);`

【personテーブル】

| 部署名   | メンバ | ランク | 年齢 |
|-------|-----|-----|----|
| 総務    | A   | 7   | 30 |
| ①② 開発 | B   | 8   | 25 |
| 経企    | C   | 5   | 41 |
| ① 開発  | D   | 6   | 35 |
| ② 開発  | E   | 7   | 27 |

【例】

- ① 開発部を全員とてから  
② 20代のみ取得したい



| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 開発  | B   | 8   | 25 |
| 開発  | E   | 7   | 27 |

【例】

`select * from ( select * from person where 部署名 = '開発' ) a where a.年齢 <= 29 ;`

赤字で取得した1つ目の結果の表に別名(a)をつけている  
結果表に名前がないと2回目のselectで条件指定できない。

副問合せは where 句で指定する条件にも指定が可能である。

ただしこの場合は 1 列、かつ 1 行の情報のみ取得する、という制約がある。

このような副問合せは「スカラ・サブクエリ」と呼んだりもする。

その為、複数取得するとエラーとなる。

## 副問合せの別の書き方

副問合せの中の select による返却が 1 列、1 行であれば以下のように、 where 句内の条件にも記載もできる。

```
select * from person where 年齢 <= 45 and
部署名 = (select 部署名 from person where 部署名 = '経理');
```

【person テーブル】

| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 総務  | A   | 7   | 30 |
| 開発  | B   | 8   | 25 |
| 経企  | C   | 5   | 41 |
| 開発  | D   | 6   | 35 |
| 開発  | E   | 7   | 27 |

副問合せで取得されるのは  
1列、かつ1行の情報。  
この場合、「経理」だけ取得される。

これまで from より後に副問合せを入れる事を説明してきたが、 from より前に入れる事もできる。

その場合、 select ( select ~ ) from table 名 となる。

この副問合せも、 where 句の条件に指定する場合と同様、 1 列・1 行である事が前提となる。

## from より前に記載する副問合せ①

これまでの副問合せはいずれも from より後に記載してきた。

- ① select \* from ( select \* from person where 部署名 = '開発' ) a where a.年齢 <= 29 ;
- ② select \* from person where 年齢 <= 45 and  
部署名 = ( select 部署名 from person where 部署名 = '経理' );

from より前にも副問合せを入れる事はできる。

以下のような形である。

```
select (select 列名 from tbl名) from tbl名 ;
```

記載なくても良い

ここは 1 列、かつ 1 行しか取得されない事が前提となる。

### 【例】

```
select (select 名前 from users where id = 1),(select 部署名 from dept where id = 1);
```

( select ~ ) の副問合せは複数指定可

### 【結果】

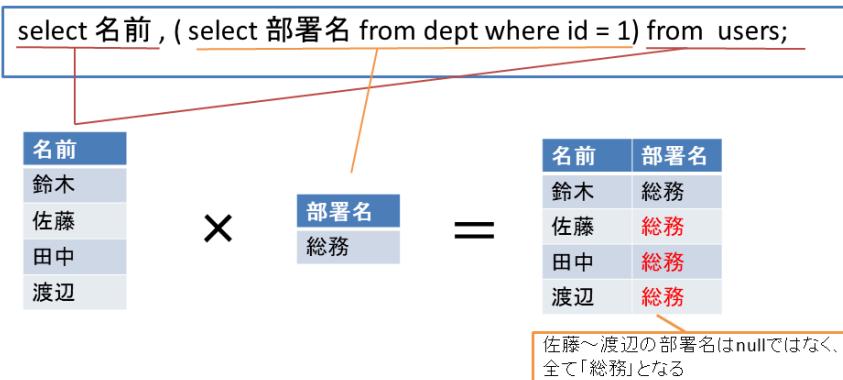
| メンバ | 部署名 |
|-----|-----|
| 鈴木  | 総務  |

person テーブルでは 鈴木さんの dept\_id(は 2(運営) だった。  
個別に取得する事で テーブル 設計に依存しない柔軟な取得が可能となる。

内側（副問合せ）は1行の情報を取得するが、外側のselect文で複数行取得する場合、どのような動作になるかを下記に記載する。

## fromより前に記載する副問合せ②

- fromより前に副問合せを入れた場合、  
fromをつける事で指定したテーブルの情報と合わせて  
確認をすることができる。  
以下に動作イメージを記載する。



## (2).相関副問合せについて

ここからは「相関副問合せ」について記載する。

相関副問合せならではの動きをする為、注意する事。

### 相関副問合せについて①

- 副問合せの基本的な考え方は(カッコ内)の内側問い合わせの後に、外側問い合わせを実施する。  
この流れの為、外側の問い合わせは内側問い合わせに影響しない。

```
select * from person where 年齢 <= 45 and 部署名 = (select 部署名 from person where 部署名 = '経理');
※ 緑字が外側、赤字が内側(前ページと同じSQL文)
```

- ただし内部側で結合を行う場合、外側で定義したテーブルを参照できる。

【外と内、同じテーブルを指定した場合】

```
select * from person as p1 where age = (select max(age) from person as p2 where p1.部署名 = p2.部署名);
```

- 上記文では以下2つの特徴的な点がある。

① 内側問い合わせのみで実行するとエラーとなる。  
= 外側問い合わせで定義したd1を、内側で使用している為。(=内側が外側を参照している)

② ①の記載から、内側が動いた場合を想定する。  
その場合、d1とd2も同じテーブルの為、結合しても元々のテーブルと変わらない。  
この状態でmax(age)をとると1つのレコード(全体で最も高い年齢)が返り、  
続いて外側の処理がされるため、最終的な結果も1つとなると考えられる。  
が、そうならない。結果は部署名ごとにmax(age)が取得される  
⇒ 結合条件(今回の場合は、部署名ごとに内部処理(max(age))が実施される。

① 外側で定義したd1を  
内側で定義なしに使っている

② 結合条件の列ごとに  
処理(max)を実施  
= 部署名「開発」の  
レコードは  
年齢最大の  
Dのみ取得

| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 総務  | A   | 7   | 30 |
| 開発  | B   | 8   | 25 |
| 経企  | C   | 5   | 41 |
| 開発  | D   | 6   | 35 |
| 開発  | E   | 7   | 27 |

【personテーブル】

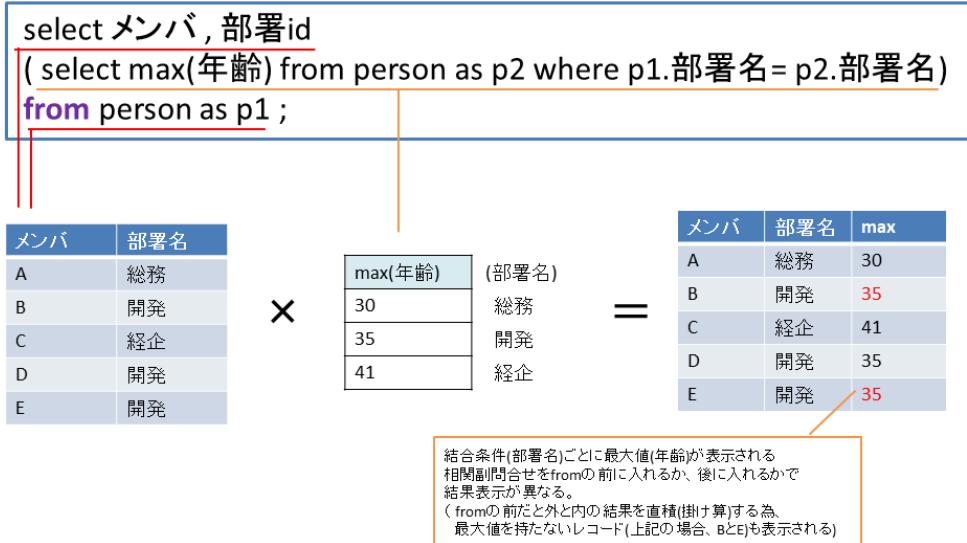
上記相関副問合せ  
SQLの実施結果



| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 総務  | A   | 7   | 30 |
| 開発  | D   | 6   | 35 |
| 経企  | C   | 5   | 41 |

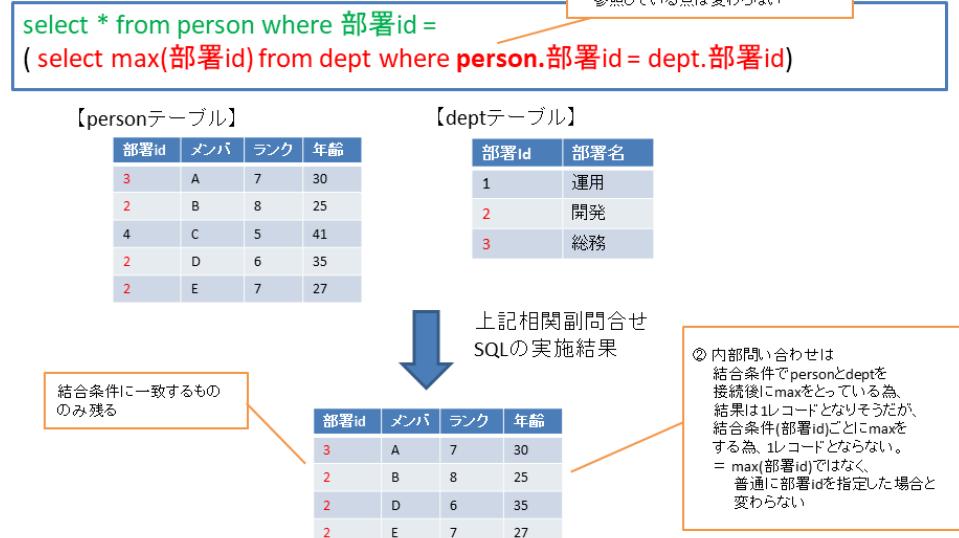
## 相関副問い合わせについて②

- fromの前に相関副問い合わせを入れる事も可能。



## 相関副問い合わせについて③

- 前述では内側・外側ともに同じテーブルだったが、別テーブルでも実施可能。  
相関副問い合わせの特徴は前と変わらない。  
以下に例を記載する。



### (3).where 句の条件で複数行指定する方法

本節の(1)にて、where 句の条件で副問合せを指定する場合、1 行である必要がある事を説明してきた。

これを複数行とし、複数条件としたい場合に in, some/any, exists 句を使用する。

なお、in と exists は副問合せ以外にも使用できる為、そちらから説明する。

まずは in 句について記載する。

in 句は or 条件を複数設定したい場合に使用する。

## inのシンプルな使い方

- in句は特定の列に、特定の情報(完全一致・複数指定可)が入っていないか確認する為に使用する。  
(or条件を指定しやすい)

`select * from [テーブル名] where [列名] in ([値1],~);`

複数指定しない場合でも()は必要  
or条件で沢山指定したい場合に便利  
ただし、nullは指定はできるが、ヒットしない。

【例】personテーブルから「開発」と「総務」の人を抜き出したい  
`select * from person where 部署名 in ('開発', '総務');`

| 【personテーブル】  |                   |     |    | 【結果】 |     |     |    |
|---------------|-------------------|-----|----|------|-----|-----|----|
| 部署名           | メンバ               | ランク | 年齢 | 部署名  | メンバ | ランク | 年齢 |
| 総務            | A                 | 7   | 30 | 総務   | A   | 7   | 30 |
| 開発            | B                 | 8   | 25 | 開発   | B   | 8   | 25 |
| 経企            | C                 | 5   | 41 | 開発   | E   | 7   | 27 |
| 部分一致はヒットとならない | 開発・運用<br>(devops) | D   | 6  | 35   |     |     |    |
|               | 開発                | E   | 7  | 27   |     |     |    |

副問合せで使用する場合も考え方は同じで、副問合せ内で取得した値を or 条件で検索する。

## inを使った副問合せ

- 前ページのように in 句は複数の値が入っているか確認するもの。  
これを応用し、in 句の指定内を select とすることで、  
in 句内で取得した結果(列情報)を検索条件とする副問合せができる

```
select * from [テーブル名] where ([列名1],~) in (select [列名2],~
from [テーブル名]);
```

In の前に not を入れる事で  
否定形にできる

複数指定する場合に記載。  
複数指定しない場合、前半の () は不要

【例】2テーブルを使用し、情報を取得する場合(売上が200万以上の部署の20代メンバを取得したい)

```
select * from person where 年齢 <= 29 and 部署名 in (select 部署名 from 売上 where 売上 >= 200);
```

()の外と中で独立している為、  
どちらの表のカラムか指定不要。  
(部署,年齢というような記載不要)

指定する列名は例では同じ「部署名」としているが、異なる列名でもOK。

【personテーブル】

| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 総務  | A   | 7   | 30 |
| 開発  | B   | 8   | 25 |
| 経企  | C   | 5   | 41 |
| 開発  | D   | 6   | 35 |
| 開発  | E   | 7   | 27 |

【売上テーブル】

| 部署名 | 売上(万) |
|-----|-------|
| 総務  | 100   |
| 開発  | 200   |
| 経企  | 300   |

【結果】

| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 開発  | B   | 8   | 25 |
| 開発  | E   | 7   | 27 |
| 経企  |     |     |    |

in 句を使用する場合、イコール条件(等価条件)で or 検索がされる。

some/any 句は比較演算子が使用できるため、in 句より更に柔軟な問い合わせができる。

## some/anyを使った副問合せ

- in 同様、some/any を使用し、副問合せを記載できる。  
上記との違いは「比較演算子」を使用する点にある。  
なお、some・any は名前は異なるが動きは同じ。

```
select * from [テーブル名] where [列名1] 比較演算子 [any/some] (select [列名2]
from [テーブル名2]);
```

=, <, >, <=, >=, != 等

【例】2テーブルを使用し、情報を取得する場合(売上が200万以上の部署の20代メンバを取得したい)

```
select * from person where 年齢 <= 29 and 部署名 = some (select 部署名 from 売上 where 売上 >= 200);
```

- (カッコ)内の副問合せで、売上テーブルから「開発」「経理」を取得
- 上記と同じ部署名のレコードを person テーブルから取得し、かつ29歳未満の人を取り出す  
(①でヒットしたものは全て or 条件で ②のテーブルから取得する。)  
→ 副問合せで受け取った値から更に比較演算子で情報を取得する際に便利。

【personテーブル】

| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 総務  | A   | 7   | 30 |
| 開発  | B   | 8   | 25 |
| 経企  | C   | 5   | 41 |
| 開発  | D   | 6   | 35 |
| 開発  | E   | 7   | 27 |

【売上テーブル】

| 部署名 | 売上(万) |
|-----|-------|
| 総務  | 100   |
| 開発  | 200   |
| 経企  | 300   |

【結果】

| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 開発  | B   | 8   | 25 |
| 開発  | E   | 7   | 27 |
| 経企  |     |     |    |

`exists`は指定した条件が存在するかどうかを確認する構文。

基本的には副問合せを使用しても、上記は変わらない。

## existsについて

- `exists`句は条件で指定した情報があるかどうかを確認するもの。
- 特定のレコードがあるかどうかを確認するには以下。

```
select exists(select * from table名 where 条件);
※ 存在する場合 t(True), 存在しなければ f(False)が返却される
```

- `where`句に`exists`を指定する事で「条件に合致したレコードが存在する」ことを検索条件とすることができます。

```
select * from table名 where exist(select * from table名 where 条件);
※ (カッコ内)で検索かけた際に該当レコードがある事が条件。
= 該当レコードがなければwhere句の条件に合致しない為、ヒット0となる。
```

- `not`を使う事で否定形にもできる。

```
select * from table名 where not exist (select * from table名 where 条件);
```

```
usersテーブルにid1のレコードがあるか確認
select exists(select * from users where id = 1);

usersテーブルにid10がある事が条件で users一覧を検索。(あればusersの一覧を取得、なければヒット0)
select * from users where exists (select * from users where id = 10);

deptテーブルにid10がある事が条件で、users一覧を検索。(あればusersの一覧を取得、なければヒット0)
select * from users where exists (select * from dept where id = 10);

deptテーブルにid10がない事が条件で、users一覧検索。(なければusers一覧取得、あればヒット0)
select * from users where not exists (select * from dept where id = 10);
```

`exists`は相関副問合せを実施すると動きが変わる。

## existsと相関副問合せを使った動作イメージ

【personテーブル】

| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 総務  | A   | 7   | 30 |
| 開発  | B   | 8   | 25 |
| 運用  | C   | 5   | 41 |
| 開発  | D   | 6   | 35 |
| 開発  | E   | 7   | 27 |

【売上テーブル】

| 部署名 | 売上(万) |
|-----|-------|
| 総務  | 100   |
| 運用  | 200   |
| 開発  | 300   |

【例】2テーブルを使用し、情報を取得する場合  
 (売上が200万以上の部署かつ、30歳以上のメンバを取得したい)  
`select * from person where 年齢 >= 30 and exists  
 (select * from 売上 where person.部署名 = 売上.部署名 and 売上.売上 >= 200);`

【結果】

| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 運用  | C   | 5   | 41 |
| 開発  | D   | 6   | 35 |

existsの()の中と外のfromでそれぞれ表を指定しているが、()内で紐づけ(使用)できる  
 exists句内の副問合せはあくまで  
 「その条件のものが存在しているかどうか」のみ確認だった。  
 しかし内部で相関副問合せを使った場合、  
 内部問い合わせで条件に一致したものが  
 外側問い合わせのand条件となる。  
 (上記例では内部問い合わせで残るのは「開発」と「運用」、  
 よって外側では「年齢が30以上」かつ「開発」か「運用」の人が  
 取得される)

## 6-5.with句と再帰的問い合わせについて

ここからは with 句と with recursive 句について説明する。  
 これらは CTE(共通テーブル式)と呼ばれることもある。  
 名前や文法は似ていても、実施できる事は異なるので、注意してほしい。

まずは with 句について記載する。  
 with 句を使用すると、複雑な処理も直感的に記載が出来る事が多い。

以下の例は ex1 という一時テーブルを作って、その結果を取得している、というイメージである。  
 これで副問合せも対応できる。  
 これまでの副問合せを比較して、やや直感的、と感じる人も多いのではないだろうか。

### withを使った問い合わせについて

- with句を使用すると副問合せでやりたい事ができる場合がある  
`with [クエリ名] as ( select ~ ) select * from [クエリ名] ~`  
 間に [クエリ名] as ( select ~ ) と入れれば複数定義可  
 [クエリ名] は、一時テーブルのようなイメージ。

【例】前ページと同じものを取得  
`with ex1 as ( select * from person where 部署名 = '開発' )  
 select * from ex1 where 年齢 >= 29 ;`

| 部署名 | メンバ | ランク | 年齢 |
|-----|-----|-----|----|
| 総務  | A   | 7   | 30 |
| 開発  | B   | 8   | 25 |
| 経企  | C   | 5   | 41 |
| 開発  | D   | 6   | 35 |
| 開発  | E   | 7   | 27 |

① 開発部を全員とってから  
 ② 20代のみ取得したい

以降で再帰的問い合わせができる with recursive 句について記載する。

再帰処理とは、簡単にいえば繰り返し処理のようなイメージである。

つまり、with recursive 句を活用することで、繰り返し処理ができるようになる。

## 再帰的問い合わせ1(with recursiveとは)

- 簡単に言うとSQLでループ(for文やwhile文)みたいなことができる。
- SQLではループ処理はせず、1回の処理でまとめて結果を出すのが基本。  
例えば、シーケンスの値を次回分3回まとめて見てみたい、とする。  
だが、以下だと1回分しか払い出されない。

```
select nextval('シーケンス名');
```

上記をやりたい場合はunionを使う方法はある。

```
select nextval('シーケンス名') union select nextval('シーケンス名')
union select nextval('シーケンス名')
```

再実行する数だけunionをつける。

ここで問題になるのが100回分ループしたい場合は  
それだけ多量に記載しないといけなくなる点。



with recursiveの出番

## 再帰的問い合わせ2(with recursiveの書き方)

- with recursiveの書式は以下。

```
with recursive [クエリ名](変数) as
(select ~ union(all) select ~) select * from [クエリ名];
```

1回目(初回)の処理

2回目以降(ループ)の処理(ループ完了の条件もここに書く)

※ 動きはwithと基本同じ。最初に()で指定したクエリを実行し、その後に  
その結果を最後のselectで取得する

【例】:シーケンス(test\_seq)の次の番号を100回払い出し

一時テーブルのカラム名(変数のようなイメージ)  
with recursive loop\_test(x, y) as
 (select 1, nextval('test\_seq')::int
 union
 select x + 1, nextval('test\_seq')::int from loop\_test where x < 100)
 select \* from loop\_test;

With recursiveはintをデフォルトで扱う。

シーケンスはデフォルトbigintを使う為、型が異なる。  
→キャスト変換する。

変数欄で指定した名前がカラム名となる。  
上記のSQLを見ると実際のところ、変数はxしか使っていない。  
ただし、yがnextvalで受け取る行となる為、記載は必要

1回目の処理ではselect 1を指定するが、  
これは1回目の処理では変数(x)を入れられない為。  
変数xの開始時点となる。(例えばselect 3とすると、xは3から開始される)

【結果】

| x   | y   |
|-----|-----|
| 1   | 1   |
| 2   | 2   |
| 3   | 3   |
| ... |     |
| 100 | 100 |

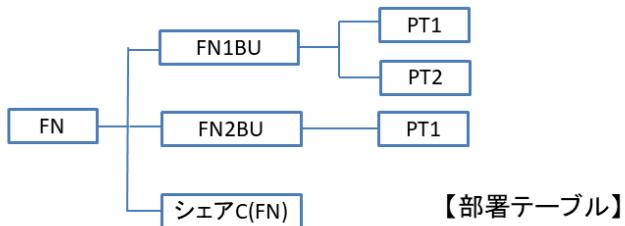
ここまでかなり極端な例であった。

with recursive は階層構造の処理等に向いている。

上記ユースケースを以降で説明する。(FN から始まる文言や PT 等は組織名と読み取ってほしい)

## with recursive のユースケース①

- with recursive は階層構造のデータ取得等に向いている。  
例えば以下のような組織図があったとする。



【部署テーブル】

| 部署id | 親部署id | 部署名      |
|------|-------|----------|
| 1    | -     | FN       |
| 2    | 1     | FN1BU    |
| 3    | 1     | FN2BU    |
| 4    | 1     | シェアC(FN) |
| 5    | -     | 総務       |
| 6    | 5     | シェアC(統括) |
| 7    | 2     | PT1      |
| 8    | 2     | PT2      |
| 9    | 3     | PT1      |

- 上記組織図を表す為に右のようなテーブルを作ったとする。  
この場合、大元(FN)に紐づく組織を取得するには親部署idが1だけでなく、2や3も対象となり、検索条件が複雑となる。  
= with recursive でまとめて取得する

## with recursive のユースケース②

- FNに関連する組織だけ取得する構文は以下となる。

```

with recursive temp(部署id , 親部署id , 部署名) as
(select 部署id , 親部署id , 部署名 from 部署 where 部署名 = 'FN'
union
select 部署.部署id , 部署.親部署id , 部署.部署名 from 部署, temp
where temp.部署id = 部署.親部署id)
select * from temp ;

```

### 【実行の流れ】

- まずは緑文を取得。tempに結果[1, null, FN]を入れる。
- 次に赤文を取得。tempの部署idには①の結果から1が入っている。  
Inner joinの処理で部署テーブルの親部署idが1であるものを取得する。  
= tempに[2, 1, FN1BU], [3, 1, FN2BU], [4, 1, シェアC(FN)]を入れる
- ②で追加された条件で赤文を再実施。  
tempの部署idには2, 3, 4が追加されている為、inner joinの条件から部署テーブルから親部署idが2, 3, 4のものを取得する  
= tempに[7, 2, PT1], [8, 2, PT2], [9, 3, PT1]が追加される。
- ③で追加された条件で赤文を再実施。  
Tempの部署idには7, 8, 9が追加されている為、inner joinの条件から部署テーブルから親部署idが7, 8, 9のものを取得する  
= tempに追加なし。(親部署idが7, 8, 9のものは存在しない)
- tempテーブルの結果を取得する。

### 【tempテーブルの状態】

| ① | 部署id | 親部署id | 部署名      |
|---|------|-------|----------|
| 1 | null | FN    |          |
| 2 | 1    |       | FN1BU    |
| 3 | 1    |       | FN2BU    |
| 4 | 1    |       | シェアC(FN) |

検索 +

| ② | 部署id | 親部署id | 部署名      |
|---|------|-------|----------|
| 2 | 1    |       | FN1BU    |
| 3 | 1    |       | FN2BU    |
| 4 | 1    |       | シェアC(FN) |

検索 +

| ③ | 部署id | 親部署id | 部署名 |
|---|------|-------|-----|
| 7 | 2    |       | PT1 |
| 8 | 2    |       | PT2 |
| 9 | 3    |       | PT1 |

## 6-6.その他について

### (1).SQL の内部処理方法の確認

SQL を実行すると、DBMS 内でどのようにその処理を実行するか検討する。

内部でどのような計画（プラン）をたて、どう実行しているかを把握する為の文法に explain がある。

重い処理をしていないはずなのに想定以上に処理が遅い等の性能観点の課題が生じた場合には本構文を使い、解析をしていくことになるだろう。

極めて重要な文法である為、ぜひ頭に入れておいてほしい。

### SQLの動作チェック(explain)

- SQL実行後にはざっくりいうと
  1. 処理内容をどうするか決める(プラン)
  2. その処理を実行する
 という2段階を経て処理が実行される。  
 → SQLは「何をやるか」しか指定せず、それを実現する具体的な手段はDB側で決めている。
- プランは登録されているデータ情報等をまとめた統計情報をもとに決める。  
 良くないプランを選定しているとSQLが遅くなる原因となる。  
 → どういったプランを作っているのか把握する必要がある。  
 → 処理が遅かった場合等に解析でよく使用する。

#### 【プランのみ確認】

- explain select ~

#### 【実際に実行した結果も確認】

- explain analyze select ~

| 【結果】                                                                                                                                                          | 解説                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------|
| <pre>-----<br/>QUERY PLAN<br/>-----<br/>Seq Scan on foo (cost=0.00..155.00 rows=10000 width=4)<br/>Planning time: 0.077 ms<br/>Execution time: 1.543 ms</pre> | 緑枠は処理内容。<br>赤枠はexplainのみだと出力されない。<br>プランにかかった時間と実行にかかった時間。 |

Seq Scan は「シーケンシャルスキャン」とも言い、シンプルに上から順番に情報にアクセスしている動作をしている。

他にも様々な動作とそれに紐づいた表記があり、検索手法や検索対象の index の有無、postgres の設定ファイル (postgresql.conf) で設定したメモリ量、postgres が持っている管理情報(統計情報)等によって変化する。

すなわち同じ SQL 文でも状況に応じて異なる内部の処理で動く事がある( = 同じ SQL 文でも状況によって処理速度に差が出る事がある)し、一見大したことがない処理でも極めて時間がかかるケースが発生しうる。

上記のような性能課題を解消する「パフォーマンスチューニング」では explain を使って内部処理を確認し、なるべく高速に動く処理になるように状態を変化させていく事となる。

explain で表示される動作(表記)について一例を下記に記載する。

まずはどのようなものがあるか参考になれば、と思う。

## パフォーマンスチューニングについて

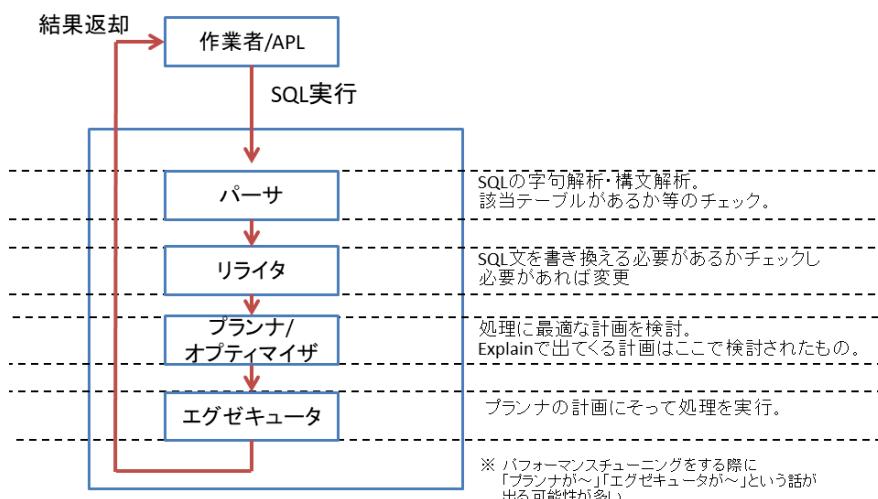
- explain 及び explain analyze で出てくる結果について。  
以下のようなものが良く出てきがち(例として記載)。  
→ postgres のメモリ使用設定を変更したり、  
index を付与して対処していく。

| タイトル            | 説明                                           |
|-----------------|----------------------------------------------|
| seq scan        | 全行を淡々と確認している。<br>Indexを使っておらず、遅い。            |
| index scan      | Indexを使用して処理。<br>Indexでスキャン後にテーブルアクセスでデータ取得。 |
| index only scan | Indexを使用して処理。<br>Indexのみでデータ取得。              |
| quick sort      | メモリのみで処理する為、早い。                              |
| external sort   | メモリとディスクを使って処理する為、時間がかかる                     |
| nested loop     | Joinで使用。<br>データが小さい場合に有効。                    |
| merge join      | 結合するテーブルをソートした後に結合する。<br>データが多い場合に有効。        |
| hash join       | メモリが十分にある場合に有効。                              |

参考までに SQL を実行した後の動作を以下に記載する。

## SQL実行の処理の流れ

- SQLをpostgresに投げたときの詳細動作について以下に示す。



前述の説明で軽く触れた、postgres が持っている管理情報(統計情報)について補足をする。

まず、管理情報は常に最新な状態とはならない。

管理情報を更新する為には、analyze という処理を実行する必要がある。

vacuum と同様に SQL 対話モードで実行できるが、標準 SQL にはない文法であり、postgres を扱う際に覚えておいてほしいものとなる。

## [参考] 統計情報とanalyzeについて

- analyzeは統計情報(どのテーブルにどんな情報が入っているか等)を更新する。  
→ explain・explan analyzeをする際に内部でどのような処理をしているか見られるが、その内部処理のどれを動かすのかを上記統計情報をもとに決定する。

### ■ 書式

[analyze]

# analyzeの実行。テーブル名省略時は全テーブルが対象となる。

• **analyze [テーブル名];**

[vacuum] ※ vacuumと同時にanalyzeも実行できる

• **vacuum analyze [テーブル名];**

※ analyzeもvacuumと同様に標準SQLにはない文。

analyzeもvacuumと同様にautovacuumの対象とできる。

上記と合わせて覚えておきたいのがシステムカタログ・統計情報コレクタである。

SQL 実行時のプランを作る際に参考とされる統計情報の他にも postgres が管理する様々な情報が確認できる。

## [参考] システムカタログ・統計情報コレクタについて

- postgresには自身が作ったテーブルやビュー以外に、全体情報管理用に自動で作られているテーブル・ビューがある(=システムカタログと統計情報コレクタ。)
- postgresで管理している様々な情報が見られる。
- pg\_XXX というテーブル・ビューノ。
- 様々なテーブル・ビューがあるが、以下で3つほど紹介する。
  - pg\_stat\_activity: 現在処理しているSQL一覧が見られる。
  - pg\_stat\_user\_tables: デッドタプル(vacuumされていない情報)がどの程度あるか確認できる
  - pg\_locks: 現在ロックされている物の情報が見られる

7章で学習するテーブル空間の確認やファンクションの確認でも使用しているものの為、記憶に留めておいてほしい。

## (2).Window関数について

グルーピング処理にしたものにWindow関数がある。

データ分析時に使う事がある為、覚えておけると良い。

### Window関数とは

- 指定したカラムでグルーピングして計算等(集約関数)を行う
- 動き的にはgroup byに近いが、グルーピングした際に指定したカラムで重複したデータを1つにまとめない。  
`select 集約関数([カラム]) over ( partition by [グルーピングカラム] ) from [テーブル名];`

【personテーブル】

| 部署名 | メンバ | 年齢 |
|-----|-----|----|
| 総務  | A   | 30 |
| 開発  | B   | 25 |
| 経企  | C   | 41 |
| 開発  | D   | 35 |

- ① `select avg(年齢) from person ;`
- ② `select 部署名, avg(年齢) from person group by 部署名 ;`
- ③ `select 部署名, avg(年齢) over (partition by 部署名 ) from person;`

【①の結果】

| avg   |
|-------|
| 32.75 |

group byを指定しないと  
グループごとの平均は  
不可。(全体の結果となる)

【②の結果】

| 部署名 | avg |
|-----|-----|
| 総務  | 30  |
| 開発  | 30  |
| 経企  | 41  |

【③の結果】

| 部署名 | avg |
|-----|-----|
| 総務  | 30  |
| 開発  | 30  |
| 開発  | 30  |
| 経企  | 41  |

開発は2つあったが  
それぞれ表示。  
(1つに  
まとめられない)

### Window関数を使った順位付け

- 計算した結果をランク付けしたい場合は「partition by」ではなく、「order by」を使用する
- ランク付け用の関数を使う  
`select [ランク関数] over ( order by [対象カラム] [asc/desc] ) from [テーブル名] ;`  
※ 何も指定しない場合はascとして動く。

【ランク関数】

| No. | 部署名          | 説明                                                      |
|-----|--------------|---------------------------------------------------------|
| 1   | row_number() | 上位から番号を振る。同着なし。                                         |
| 2   | rank()       | 同じ値なら同着とし、次は同着順位+同着対象数とする                               |
| 3   | dense_rank() | 同じ値なら同着とし、次は同着順位+1とする                                   |
| 4   | ntile(n)     | 結果をnの数だけわける<br>※ 年齢が若い順に並べたうえで近い者同士でn人のグループを組む、みたいなイメージ |

【例】`select * , [ランク関数] over (order by a.avg) from  
( select 部署名, avg(年齢) over (partition by 部署名 ) a from person);`

【No.1の結果】

| 部署名 | avg | rank |
|-----|-----|------|
| 総務  | 30  | 1    |
| 開発  | 30  | 2    |
| 開発  | 30  | 3    |
| 経企  | 41  | 4    |

30が3つあるが、同着なし

【No.2の結果】

| 部署名 | avg | rank |
|-----|-----|------|
| 総務  | 30  | 1    |
| 開発  | 30  | 1    |
| 開発  | 30  | 1    |
| 経企  | 41  | 4    |

30が3つある為、次は4位

【No.3の結果】

| 部署名 | avg | rank |
|-----|-----|------|
| 総務  | 30  | 1    |
| 開発  | 30  | 1    |
| 開発  | 30  | 1    |
| 経企  | 41  | 2    |

30が3つあるが、次2位

【No.4の結果】

| 部署名 | avg | rank |
|-----|-----|------|
| 総務  | 30  | 1    |
| 開発  | 30  | 1    |
| 開発  | 30  | 2    |
| 経企  | 41  | 3    |

nが3の場合

## Window関数を使った比較

- その他、比較も可能。  
これも同様に over (order by) で指定する。

### 【特定行取得関数】

| No. | 部署名                | 説明            |
|-----|--------------------|---------------|
| 1   | first_value(対象カラム) | 最初の値を取得       |
| 2   | last_value(対象カラム)  | 最後の値を取得       |
| 3   | nth_value(対象カラム,n) | N番目を取得        |
| 4   | lag(対象カラム,n)       | 現在行からN個前の値を取得 |
| 5   | lead(対象カラム,n)      | 現在行からN個後の値を取得 |

【例】`select * , ( first_value(a.avg) over (order by a.avg) - a.avg ) as 差分 from (select * , avg(年齢) over (partition by 部署名) from person) a;`

first\_value()を使い、一番上の平均年齢と自部署の平均年齢を比較

【副問合せ箇所で取得したもの】

| 部署名 | メンバ | 年齢 | avg |
|-----|-----|----|-----|
| 総務  | A   | 30 | 30  |
| 開発  | B   | 25 | 30  |
| 開発  | D   | 35 | 30  |
| 経企  | C   | 41 | 41  |

【上記SQLの結果】

| 部署名 | メンバ | 年齢 | avg | 差分  |
|-----|-----|----|-----|-----|
| 総務  | A   | 30 | 30  | 0   |
| 開発  | B   | 25 | 30  | 0   |
| 開発  | D   | 35 | 30  | 0   |
| 経企  | C   | 41 | 41  | -11 |

### (3).UPSERTについて

insert 時に条件に応じて insert か update か切り替わる、UPSERT と呼ばれる構文がある。

これは insert と update を組み合わせた造語である。

以降は UPSERT に関する構文について記載する。

### 特定条件の時だけ挿入 or 更新(on conflict)

- 特定の条件に合致したときだけ挿入 or 更新を行う(upsertという概念)  
具体的には以下2つがある
  - ① テーブルに挿入したいデータがない時は挿入、  
存在する時は何もしない。( = do nothing句)  
`insert into [table名] values ( 値 , ~ ) on conflict (条件列) do nothing;`
  - ② テーブルに挿入したいデータがない時は挿入、  
存在する時は該当行を更新する( = do update句)  
`insert into [table名] values ( 値 , ~ ) on conflict (条件列) do update set 条件式;`

| id | name | age |
|----|------|-----|
| 1  | 山口   | 28  |
| 2  | 鈴木   | 29  |
| 3  | 佐藤   | 35  |

↓

|   |    |    |
|---|----|----|
| 1 | 上原 | 28 |
|---|----|----|

- ① `insert into person values ( 1 , '上原' , 28 ) on conflict (id) do nothing ;`  
 ② `insert into person values ( 1 , '上原' , 28 ) on conflict (id) do update set name = '上原' ;`

挿入しようとしている行のidが1で、既に存在している(条件列にidをしている)為、  
 ①なら何もしない、②ならnameを山口から上原に更新。  
 なお、挿入しようとしているものの条件列がnameなら上原で重複はないため、  
 ①②ともに新規登録される。

## 複数upsert使用した場合(excluded)

- Insert文では複数行を挿入可能。  
その場合、upsertを使おうとするとどうなるだろうか。

  - ① do nothingの場合、重複しなければ登録するし、重複すれば登録しない為、考慮不要。
  - ② do updateの場合、重複しなければ登録するが、重複すれば更新する。  
その際の更新条件はどうなるのか。

② insert into person values (1 , '上原', 28),( 2, '横浜', 30 )  
on conflict (id) do update set name = '上原';

Id 1も2も「上原」で更新されてしまう。  
1は上原、2は横浜で更新したい。。。。

excluded句を使うと良い。以下でやりたい事ができる。

```
insert into person values (1 , '上原', 28),(2, '横浜', 30)
on conflict (id) do update set name = excluded.name;
```

postgres15 から入った構文に merge 文がある。

本書の問題には入っていないが、あわせて理解しておけると良いだろう。

## [参考] mergeについて

- PostgreSQL15から、追加されたものでinsert/update/deleteを実行できる。
- 代表的な書式を以下に記載する。
 

```
merge into [テーブル1]
using [データソース] on [条件]
when matched then
 update set [条件]
when not matched then
 insert [カラム] values [値]
```
- 長いが、太字で書いている部分がポイント。  
Using句で書いた条件に一致した場合と一致しなかった場合の動作を記載できる。

#### (4).DML のおわりとして

ここで文法の説明は終わりである。

基礎編で実施したものに加え、応用編で実施した結合や集合演算、副問合せなどを活用すれば極めて複雑な処理も実施が可能となる。

一方で複雑な処理は可読性が下がりバグの温床になる他、その処理自体の性能はもちろん、ロングトランザクションとロックにより他の処理にも影響を与える事もあり得る。

その為、長文で複雑な事を実施したい場合に「できるから長文を書く」のではなく、システム要件も考慮した上で、アプリ側で一部処理を代替できないかやSQLを分割する等、他にやりようがないのかは是非考慮してほしい。

（全てをDBでやる必要はない。余裕のあるもので処理を行うのが良いと考える）

## 7章 より高度な仕組みの活用・操作

本章では+αの内容を記載する。

より高度な事を知りたい人は参照してみてほしい。

### 7-1.本章の概要

まず、本章で触れる概念について整理する。

## +α 概要

### 【登場概念】

| タイトル          |
|---------------|
| 宣言的パーティショニング  |
| テーブル空間        |
| ドメイン          |
| 外部キー制約(アクション) |
| contrib       |
| 2相コミットメント     |
| 表明            |
| トリガー          |
| ファンクション       |
| ルール           |
| プロシージャ        |
| カーソル          |

### 【実施概要】

| 大項目              | 文法                                                                                                                                                                                                                                 |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 高度な仕組みと操作        | <pre>create table ~ partition by create table ~ partition of create tablespace create domain prepare/execute/deallocate case grouping sets/rollup/cube create extension prepare transaction/commit prepared create assertion</pre> |
| PL/pgSQLの活用、自動処理 | <pre>create trigger create function create rule create procedure</pre>                                                                                                                                                             |

本章はSQLで操作をするものもあるが、新たに登場する記法はわずかであり、どちらかというとDBの機能理解に近づく。

また、手続き言語であるPL/pgSQLを扱う箇所もある。

SQL自体の理解からはやや遠のくようにも思う為、あくまで「おまけ」「より高度な事を知りたい人向け」の内容となる。

## 7-2.+α その1：高度な操作・活用

本節では2~6章の基本編では扱わなかった、より高度な仕組み等について説明する。

### (1).宣言的パーティショニングとテーブル空間について

宣言的パーティショニングは1つのテーブルを複数のテーブルに区切り、データを複数のテーブルに配置する事。

登録する情報により、どの子テーブルに格納するかが決まる。

また、親テーブルを参照する事で全ての子テーブルの情報が確認できる。

## 宣言的パーティショニング

- ・ **宣言的パーティショニング**は1つの巨大なテーブルを情報に応じて異なるテーブルに配置すること。
  - ① 親テーブル作成  
(`create table ~ partition by文`) (下の例ならjapan)
  - ② 子テーブル作成  
(`create table ~ partition of 文`) (下の例ならkanagawa等)



#### 【例：上記パターン】

```

create table japan (pref text, city text) partition by list (pref) ;

create table kanagawa partition of japan for values in ('kanagawa') ;
create table tokyo partition of japan for values in ('tokyo') ;
create table yamanashi partition of japan for values in ('yamanashi') ;

```

上記の例ではjapan テーブルのprefの値で振り分けている。

prefが「kanagawa」「tokyo」「yamanashi」の場合、それぞれの子テーブルに格納する、という動きとなる。

どの子テーブルに格納するか、その振り分け条件の指定の仕方から以下3つの種類がある。

## パーティショニングの種類

- 前ページで触れたのは「リストパーティション」呼ばれる手法。  
他にもいくつかパーティションの手法はある。  
以下にパーティションの種類について記載する。

| 種類              | 意味                                                             | SQL文法                                                                                                                                                     |
|-----------------|----------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| リスト<br>パーティション  | 「神奈川」「東京」などのキーワードごとに配置テーブルをわける                                 | 【親】<br>create table p_tbl (~) partition by list (リストキー);<br><br>【子】<br>create table c_tbl partition of p_tbl for values in ('AAA');                       |
| レンジ<br>パーティション  | 「1~10」「11~20」というような範囲ごとに値の配置テーブルを決める。<br>日にちで区切るが多い。           | 【親】<br>create table p_tbl (~) partition by range (レンジキー);<br><br>【子】<br>create table c_tbl partition of p_tbl for values from (1) to (10);                |
| ハッシュ<br>パーティション | ハッシュ値を分割値で割った際の余りの値で配置テーブルを決める<br>なるべく特定テーブルへのアクセス集中を避けたい時に使う。 | 【親】<br>create table p_tbl (~) partition by hash (ハッシュキー);<br><br>【子】<br>create table c_tbl partition of p_tbl for values with(modulus 分割数, remainder 余り値) |

あわせて覚えておきたいのが「テーブル空間(テーブルスペース)」と呼ばれる機能である。

DBに入れたデータは通常DBクラスタとして使っているディレクトリ配下に物理データファイルとして配置されるが、その置き場を変えることができる機能である。

## テーブル空間

- テーブル空間(tablespace)はDBの実体ファイル(データファイル等)をどのディレクトリに配置するか指定できる機能。
- この機能によりデータを複数のディスクに配置が可能となる。  
DB処理でネックになりがちなのがIO処理。(書き込み処理)  
データを複数のディスクに配置する事で、IO処理ネックが改善できる事がある。
- 使い方は以下。
  - ① テーブル空間の作成  
`create tablespace tablespace名 location '/target/path' ;`
  - ② テーブルへの割り当て  
`create table table名(定義) tablespace tablespace名 ;`

※ alter table文で後から付与する事も可能

### 【例】

```
create tablespace tblspc1 location '/tmp/data01/'
create table tbl1(id int, name text) tablespace tblspc1;
```

※似た名前に「名前空間」というものがあるが別物なので注意。これはスキーマの事を指す。

テーブル空間を設定すると、設定を知らない人は物理ファイルが探しづらくなる。

例えばDBが使っているストレージ容量を確認しようとした際に、テーブル空間で使っているものが漏れてしまう可能性がある。

そのようにならないように、以下の方法で設定されたテーブル空間を確認することができる。

## テーブル空間の設定確認

- テーブル空間の一覧は以下で確認できる

```
select oid, spcname, pg_tablespace_location(oid) from
pg_tablespace ;
```

- テーブル空間とそれに紐づくディスク場所を確認するには以下となる。

```
select n.nspname , c.relname, t.spcname from pg_class c join
pg_namespace n on n.oid = c.relnamespace left join
pg_tablespace t on t.oid = c.reltablespace where c.relkind = 'r'
and n.nspname = '使用しているスキーマ名';
```

参考までにパーティショニング機能とテーブル空間を組み合わせて性能向上につなげる例を以下に記載する。

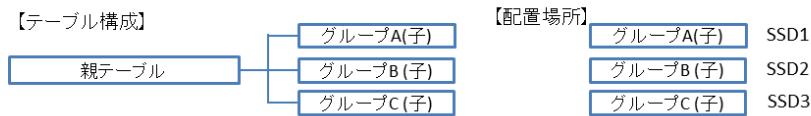
### [参考] テーブル空間と宣言的パーティショニングの組み合わせ

- どちらもディスク書き込み負荷分散に使用できる。  
Insert等が多いシステムだとそれだけディスク書き込みが多くなり、処理が重くなる。  
(DB自体の処理より、ディスク書き込みがネックになる可能性がある)  
→ 宣言的パーティショニングはテーブルを分ける事による  
他にも参照のアクセス量減少や運用の効率化、  
シャーディング(※)と組み合わせてスケールアウトしやすくする等にも使用できる  
※ シャーディング: 1つのテーブルを複数のマシンに配置すること  
→ 宣言的パーティショニングでテーブルを分割、かつ  
各テーブルにテーブル空間を設定する事で  
書き込み対象のディスクも分散し、処理高速化につながる。

#### 【例】

insert処理が多く処理時間がネックになっているシステムがあるとする。

テーブル空間と宣言的パーティショニングで書き込みディスクを分散させて対応できる。



## (2).自ら定義する型：ドメインについて

PostgreSQLには数値型や文字列型、時刻型など基本的なものから、inet型(IPアドレス型)やpoint型(平面の座標点。幾何データ型)のようにユニークなものも存在する。

上記のように事前に用意されているもの以外で、入力を特定の型で制限したい場合、ドメインを活用する事で独自の型を作ることができる。

## 自分で指定する型(domain)について

- テーブル作成時には列名(カラム名)とあわせて、型を指定した。  
型は整数型や文字型などがあるが、都合が良いものがない場合は自身が定義する事もできる。
- 定義にはcreate domainを使い、正規表現で条件を記載する  
`create domain [型名] as [ベースとなる型] check ( value ~ '条件');`
- 作成したdomainはcreate tableで初期からある型と同様に使用できる

### 【email型を作る】

```
create domain email as text check (
 value ~ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');

create table person (id int , name text, email email);
```

### (3). プリペアドステートメントについて

## プリペアドステートメントについて

- ・ アプリとの連携でよく使われるものに「プリペアドステートメント」と呼ばれるものがある。
- ・ これはよく使うSQLを事前に登録しておく機能。  
このメリットは高速化(2回目以降は単発でSQLをたたくより高速になる)などがある。
- ・ 文法は以下の通り。

定義: `prepare prepare名 as select ~ ;`

実行: `execute prepare名();`

削除: `deallocate prepare名 ;`

#### 【例】

```
prepare p_statement as select * from person where id = $1 ;
execute p_statement(10);
deallocate p_statement;
```

実行時に受け取る変数として記載

\* プリペアドステートメントはアプリがなくても試すことができる為、本書の問題はアプリなしで試すものとなる。

#### (4).外部キー制約の細部アクション設定について

2章の「制約」で外部キー制約について触れた。

問題(DDL)でも外部キー制約をかけた上で動作検証をする問題があるが、参照先がデータを変えようとした際には何も指定していないと通常エラーとなる。

これはデフォルト動作（アクション）が「エラーとする」ものだからであり、細部指定をすると動きを変える事ができる。

#### 外部キー制約のアクションについて(1)

- 外部キー制約は他テーブルにある情報しか登録できない

【部署】

| 部署名 | 区分   |
|-----|------|
| 総務  | スタッフ |
| 開発  | 事業   |
| 経企  | スタッフ |
| 運用  | 事業   |

【社員】

| 部署名 | メンバ | 年齢 |
|-----|-----|----|
| 総務  | A   | 30 |
| 開発  | B   | 25 |
| 経企  | C   | 41 |
| 運用  | D   | 35 |

社員テーブルの部署名は部署テーブルの部署名を参照している為、  
部署テーブルの部署名にあるものしか登録できない



参照先の部署テーブルの情報かえたら困る。。。  
例) 部署テーブルの 総務を「人事」に変えると  
    社員テーブルのAさんの部署名は総務部のままだと困る。  
    → **アクション**で動作を指定できる。

## 外部キー制約のアクションについて(2)

- アクションは以下の文法で指定する。

`~ references [テーブル名] ([カラム名]) [on delete action] [on update action]`

削除(delete)された時と更新(update)された時、  
それぞれでアクションを指定できる。  
なお、記載しなくても良い。

### 【アクションの種類】

| Action      | 説明                                                                          |
|-------------|-----------------------------------------------------------------------------|
| no action   | 更新 or 削除を使用しようとするとエラーとする<br>デフォルト(アクションを指定しない場合)これ。                         |
| cascade     | 更新した場合、更新されたデータを参照しているデータも全て同じ更新をする。<br>削除した場合、削除されたデータを参照しているデータもすべて削除する   |
| set null    | 更新 or 削除されたデータを参照していたデータを null とする                                          |
| set default | 更新 or 削除されたデータを参照していたデータを default 値とする<br>※ default 値は「シーケンスとは」ページにて説明。(後述) |

なお、テーブル定義の最後にまとめて記載する方法もある。

## 外部キー制約の別の指定方法

- これまで列名に制約をつけていた。

### 【部署】

| 部署名 | 区分   |
|-----|------|
| 総務  | スタッフ |
| 開発  | 事業   |
| 経企  | スタッフ |
| 運用  | 事業   |

### 【社員】

| 所属部署 | メンバ | 年齢 |
|------|-----|----|
| 総務   | A   | 30 |
| 開発   | B   | 25 |
| 経企   | C   | 41 |
| 開発   | D   | 35 |

社員テーブルを作成時に以下を実行する  
`create table 社員 (  
 所属部署 text references 部署 (部署名),  
 メンバ text,  
 年齢 int );`

以下のような書き方もできる

`create table 社員 ( 部署名 text ,メンバ text ,年齢 int ,  
 foreign key (所属部署, ~) references 部署(部署名, ~) );`

複数指定可。(複数指定した場合、複合外部キー制約となる)

ちなみに DB に便利な機能があるから絶対使うべき、と言う事ではない点は注意してほしい。

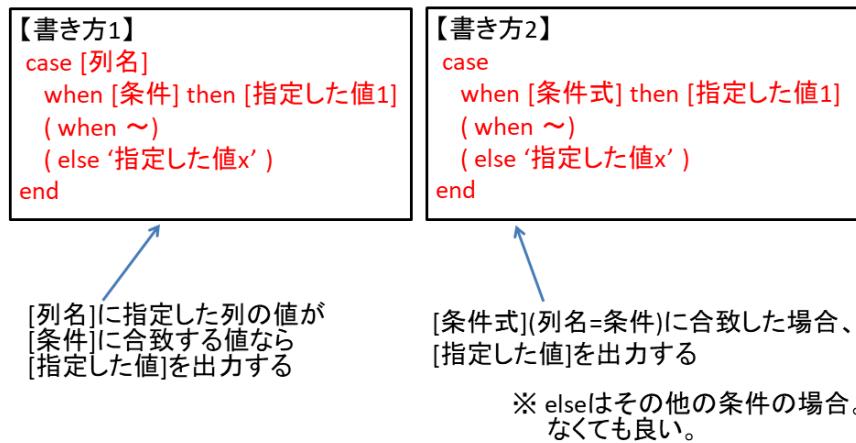
(度々触れておりくどいようだが)DB とアプリの余裕のある方が処理するべきであり、高速にデータを扱いたい場合に外部キー制約も処理時間増しにつながる為、性能を求められておりアプリで制御したほうが性能が早くなるならそのようにすべきである点、改めてあげておきたい。

## (5). データ分析の高度化に向けて(case文/grouping sets, rollup, cube)

条件に応じて出力する結果を変える、条件分岐のような処理を case 文で実施できる。  
これにより条件をつけたデータ分析などがしやすくなる。

### 条件判定 (case)

- 特定の条件の時に指定した値を出力したい場合、case文を使用する  
以下のどちらで書いても良い。



### case文の具体例

- select文を活用して以下のようなことができる  
※ 例1と例2は同じ

| 【例1】                                                                                                     | 【例2】                                                                                                          |
|----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| <pre>select 部署名 , case 部署名 when '開発' then '事業' when '総務' then 'スタッフ' else '不明' end as 区分 from 部署 ;</pre> | <pre>select 部署名, case when 部署名 = '開発' then '事業' when 部署名 = '総務' then 'スタッフ' else '不明' end as 区分 from 部署</pre> |

| 部署名 | メンバ | 年齢 |
|-----|-----|----|
| 総務  | A   | 30 |
| 開発  | B   | 25 |
| 経企  | C   | 41 |
| 開発  | D   | 35 |



| 部署名 | 区分   |
|-----|------|
| 総務  | スタッフ |
| 開発  | 事業   |
| 経企  | 不明   |
| 開発  | 事業   |

例では select で実行しているが、insert 文(insert ~ select ) や update 文、ビューの作成や関数（ファンクション、後述）等でも使用でき、ユースケースは広い。

もう1つ、データ分析に向けて便利な記法を紹介する。

それがグルーピングと集計処理に関連する grouping sets, rollup, cube である。

これまでグルーピングは group by や Window 関数（、必要に応じて相関副問合せ）を紹介してきた。  
より細かいパターンでの集計結果も見たい場合には、これらを使用すると良い。

## group by の課題

- group by では複数列指定できる。  
その場合は以下のようになる

| 事業部 | BU    | PT  | 区分 | 売上 |
|-----|-------|-----|----|----|
| FN  | FN1BU | PT1 | 受託 | 10 |
| FN  | FN1BU | PT2 | 受託 | 20 |
| FN  | FN1BU | PT3 | 製品 | 30 |
| FN  | FN2BU | PT1 | 受託 | 40 |
| FN  | FN2BU | PT2 | 製品 | 50 |
| FN  | FN3BU | PT1 | 受託 | 60 |
| DT  | DT1BU | PT1 | 受託 | 70 |
| DT  | DT1BU | PT2 | 受託 | 80 |

select 事業部, BU, 区分, sum(売上)  
from 部署  
group by 事業部, BU, 区分;

| 事業部 | BU    | 区分 | 売上 |
|-----|-------|----|----|
| FN  | FN1BU | 受託 | 10 |
| FN  | FN1BU | 製品 | 20 |
| FN  | FN2BU | 受託 | 40 |
| FN  | FN2BU | 製品 | 50 |
| FN  | FN3BU | 受託 | 60 |
| DT  | DT1BU | 受託 | 70 |

Group by で指定した3列全ての全パターン整理。

しかし事業部だけでの売上や区分だけでの売上、  
事業部と区分の組み合わせや、事業部とBUでの組み合わせでも見てみたい。  
その場合は重複した資料をまたやらせる？

## まとめて様々なパターンを集計 (grouping sets, rollup, cube )

- 実例をもとに動作を以下に示す。

| 事業部 | BU    | PT  | 区分 | 売上 |
|-----|-------|-----|----|----|
| FN  | FN1BU | PT1 | 受託 | 10 |
| FN  | FN1BU | PT2 | 受託 | 20 |
| FN  | FN1BU | PT3 | 製品 | 30 |
| FN  | FN2BU | PT1 | 受託 | 40 |
| FN  | FN2BU | PT2 | 製品 | 50 |
| FN  | FN3BU | PT1 | 受託 | 60 |
| DT  | DT1BU | PT1 | 受託 | 70 |
| DT  | DT1BU | PT2 | 受託 | 80 |

① 複数列を完全縦割りで集計  
select 事業部, BU, 区分, sum(売上)  
from 部署  
group by grouping sets(事業部, BU, 区分);

| 事業部 | BU    | 区分 | 売上  |
|-----|-------|----|-----|
| FN  |       |    | 210 |
| DT  |       |    | 150 |
|     | FN1BU |    | 60  |
|     | FN2BU |    | 90  |
|     |       | ～  |     |
|     |       | 受託 | 280 |
|     |       | 製品 | 80  |

② rollup の結果  
FN 受託 130  
FN 製品 80  
DT 受託 150  
FN1BU 受託 30  
FN1BU 製品 30  
～

③ 複数列の全組み合わせパターンで集計  
select 事業部, BU, 区分, sum(売上)  
from 部署  
group by cube(事業部, BU, 区分);

| 事業部 | BU    | 区分 | 売上  |
|-----|-------|----|-----|
| FN  | FN1BU | 受託 | 30  |
| FN  | FN1BU | 製品 | 30  |
| FN  | FN1BU |    | 60  |
| FN  |       |    | 210 |
| FN  | FN2BU | 受託 | 40  |
|     |       | ～  |     |
|     |       |    | 360 |

全てを条件に集計

## (6).contrib と dblink の活用、2相コミットメントについて

contribは後から追加する拡張モジュールのようなイメージのもので、新たな機能を提供する。 基本はcreate extensionで有効化できるが、できないcontribモジュールもある。

以降はcontribの1つ、dblinkについて説明したいと思う。

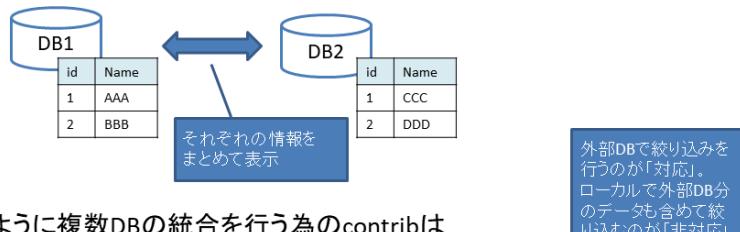
### contribについて

- contribとは簡単にいえば「拡張モジュール」の事。
- postgresインストール時には入っていないが、必要に応じて contribを後から追加で入れてpostgresの機能を拡張できる。
- 入れ方はcontribにより異なるが、特定のフォルダに入れた後に **create extension [モジュール名]** を実行するケースが多い。
- contribには以下のようなものがある(一部紹介)
  - dblink: DB間のテーブル情報を参照
  - pgcrypto: データの暗号化・複合に関するモジュール。

※ 本書ではdblinkについて説明する

### dblinkについて

- それぞれ単独で動くDBを1つのDBとして動いているように見せるもの。



- 上記のように複数DBの統合を行う為のcontribは以下の2つがある。違いは以下。

| contrib名     | 実施可能なSQL  | トランザクション                | プッシュダウン |
|--------------|-----------|-------------------------|---------|
| postgres_fdw | DMLのみ     | PREPARE TRANSACTION 非対応 | 対応      |
| dblink       | DDL, DML等 | PREPARE TRANSACTION 対応  | 非対応     |

- 上記の通り、単純なデータ取得や登録のみならpostgres\_fdwの方が良い。 テーブル作成(create table)等を行うならdblinkが選択肢となる。

※ 以降はdblinkについて説明する

## dblinkの使い方

- dblinkを使う場合は以下の手順となる

① dblinkを使う事を宣言

`create extension dblink;`

② 外部サーバを指定した上で操作を実施。

dblink使用時には `dblink('接続情報', 'SQL文')` と指定する。

外部サーバ情報には以下がある

| 指定する情報      | 指定key    |
|-------------|----------|
| データベース名     | db_name  |
| ユーザ名        | user     |
| ユーザパスワード    | password |
| 外部DBのIPアドレス | host     |
| 外部DBのポート番号  | port     |

selectを実施する場合はdblink。  
createやdrop,  
Insert/update/deleteはdblink\_exec。

【例: ローカルの情報を外部サーバの情報を統合して出力する】

```
select id , name from person
union all
select id , name from dblink(
'dbname=testdb01 user=user01 password=AAA host=192.168.10.11 port=5432',
'select id , name from person')
as external_person(id int, name text);
```

後半がdblinkを使用した例。

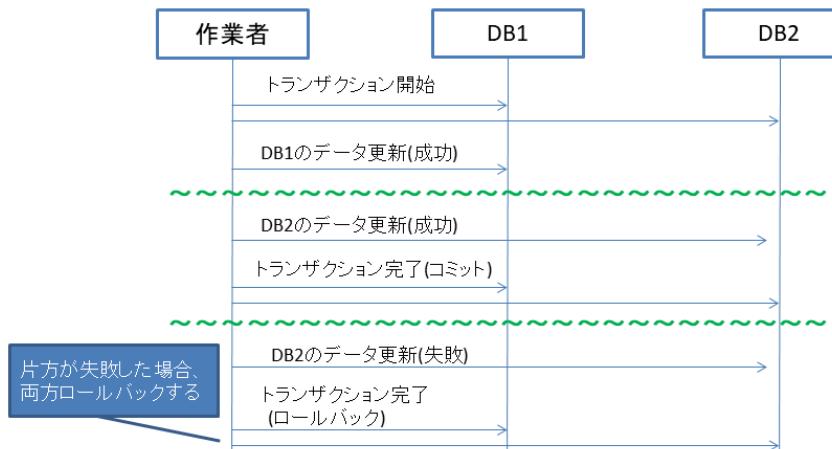
dblink でできる事の 1 つに 2 相コミットメントがある。

あわせて概念を理解しておけると良い。

## 2相コミットメントとは

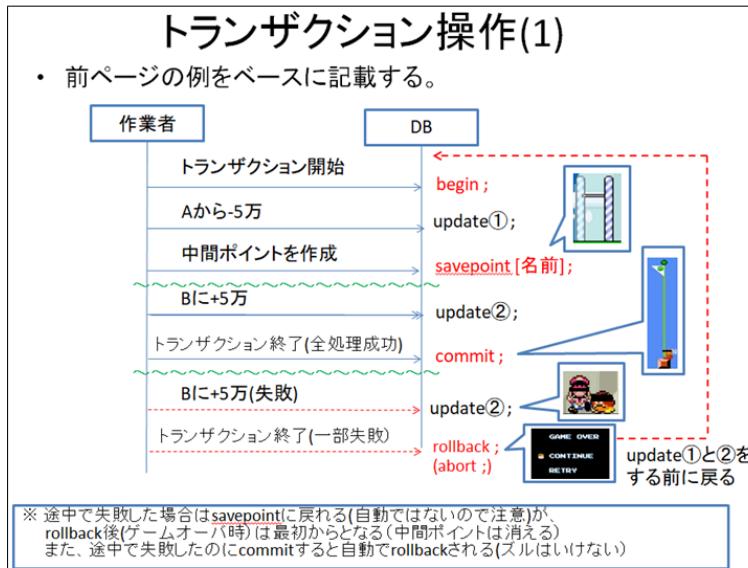
- 「PREPARE TRANSACTION」構文は2相コミットメントに使うもの。  
2相コミットメントについて以下で説明する。
- 2相コミットメントとは、2つのDBを使ったトランザクションの事を指す。

【2相コミットメントイメージ概略】



## [参考] 1相コミットメント

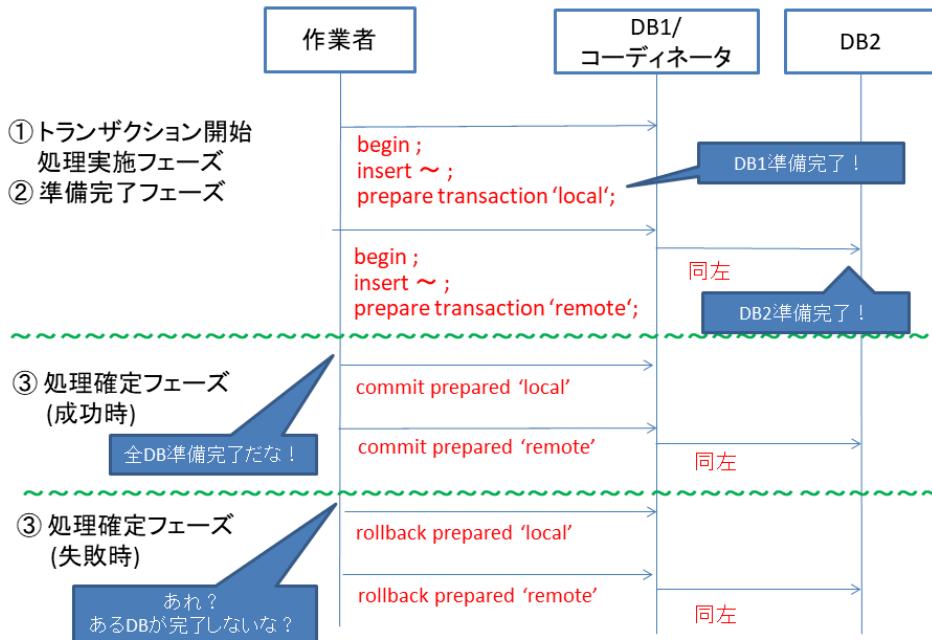
- 本書の4章で扱ったトランザクションは「1相コミットメント」だった。



## 2相コミットメントの考え方

- 各DBへのコミットやロールバックはそれぞれSQLで実施する必要がある。  
また操作は1つのDBから実施する。  
(指揮DBのようなイメージ。コーディネータと呼ぶ)
- 2相コミットメントは以下の3段階がある。
  - ① トランザクション開始と処理の実施  
# コーディネータから各DBに向けて実施  
`begin;`  
`insert/update/delete 等の処理`
  - ② 2相コミットメントの準備完了コマンドの実施  
# コーディネータから各DBに向けて実施  
`prepare transaction 'tx名';`  
※ tx名はDBごとに異なる名前にする事
  - ③ 全てのDBで準備完了となったら、各DBにコミット処理  
# コーディネータから各DBに向けて実施  
`commit prepared 'tx名'`

## 2相コミットメントの例



## 2相コミットメントの具体的なSQL例

- 前ページのように基本は1つのDB(コーディネータ)から各DBの処理を実施する。  
つまり異なるDBの操作を行う必要がある。  
= dblinkを使う。

```
【2相コミットSQL例】
DB1(コーディネータ)のトランザクション開始/準備完了
begin ;
insert into tbl1 (id, name) values (1 , 'AAA');
prepare transaction 'local';

DB2のトランザクション開始/準備完了
select dblink_exec(dbname=testdb01 user=user01 password=AAA ,host=192.168.10.11 port=5432 ,
$$ begin; insert into tbl2 (id , name) values (2 , 'BBB') ; prepare transaction 'remote' : $$);

各DBのコミット実施
commit prepared 'local';
select dblink_exec(dbname=testdb01 user=user01 password=AAA ,host=192.168.10.11 port=5432,
'commit prepared "remote"');
```

- コーディネータと他DBの連携は基本手動で行う。  
その為、アプリやファンクション等で2相コミットメントを実現する事が多い。

## 2相コミットメントのその他

- コーディネータからの処理は成功(commit)、失敗(rollback)含め手動で対応する。  
よって明に rollback prepared 'tx名' をしないと  
2相コミットメント用トランザクションが残り続ける。
- 上記のように残った2相コミットメント用  
トランザクションを確認する関数に以下がある。  
`select * from pg_prepared_xacts;`
- 2相コミットメントを行うには、コーディネータと  
外部DBそれぞれでpostgresql.confの  
「`max_prepared_transactions`」を有効化する必要がある。

なお、contrib は、できる事を拡張するモジュールと説明したが、他にも SQL では実現できない事を可能とするツールはある。

例えばバックアップやリストアを実現する（コマンドとして扱える）`pg_dump` や `pg_basebackup`（これは `postgres` をインストールすると最初から使用できる）、ベンチマークツール（性能観点を確認できるツール）の `pgbench`、PostgreSQL に GUI 機能を提供する `pgAdmin` などがある。

参考までに記憶に留めておいてほしい。

## (7). 表明について

RDB の世界には表明という概念がある。

PostgreSQL には実装されていないが、IPA の資格には登場する場合がある。

やや試験対策という趣きが強いが、参考程度に捉えてもらえば、と思う。

### [参考] create assertion文について

- PostgreSQLには現状実装されていない(PostgreSQL15までとする)が、SQL標準には「create assertion」文というものがある。  
⇒ 他のDBMS(RDB)では使用できる場合があり、IPAの資格でも登場する為、押さえておくと良い。
- ユースケースは「複数テーブルでの制約を独自に実現する」もの。  
例えば部署テーブルに、その部署に最大何人まで収容できるかの列を登録し、それ以上人数にならないように従業員テーブルの所属部署列のデータを変えられないようにする、など。

【deptテーブル】

| 部署Id | 部署名 | 収容人数 |
|------|-----|------|
| 1    | 総務  | 100  |
| 2    | 運用  | 120  |

【empテーブル】

| Id | 名前 | 部署id |
|----|----|------|
| 1  | 鈴木 | 2    |
| 2  | 佐藤 | 3    |

各部署ごとの収容人数  
(上限)を超えないように  
チェック

- PostgreSQLではcreate assertionによる制約(表明と呼ぶ)を使わずトリガー・ファンクションなどで実現する。

一般的な文法は以下。

`create assertion 表明名 check ( 条件式 );`

## 7-3.+αその2: PL/pgSQLの活用 及び 自動処理について

本節ではトリガー/ファンクション等の自動処理や、ファンクションに関わりの強いものについて触れていく。  
なお、ファンクション(関数)関連は手続き言語(PL)の理解が必要であり、PostgreSQLはいくつかの手続き言語に対応しているが、本書ではPL/pgSQLを対象に記載する。

PL/pgSQLはPostgreSQLに向けて用意されたSQLが使えるプログラミング言語、というイメージを持ってもらいうと良い。

Excel等の関数が使えるVisual basicに扱いがやや似ているかもしれない。

Oracle DBの手続き言語であるPL/SQLと文法が似ていると言われている。

DB、特にPostgreSQLを扱う際には必要な知識となる為、興味ある人は是非確認してみてほしい。

### (1). トリガーファンクションと、ルールについて

ファンクション(関数)とは、1つの機能を提供する処理のまとめである。

例えばsumやavg等の集約関数も、与えられたものの合計値・平均値を出す、という機能を提供する関数である。  
sumやavgは最初から提供されているが、自分で関数を作ることもできる(ユーザ定義関数、と呼ぶ)。  
この際にPL/pgSQLという手続き言語を使用する。

自身が作るファンクションには使われ方によって大きく以下の2つがある。

- ① トリガーファンクション
- ② ストアドファンクション

まずは動作が文法が少なくて済む①トリガーファンクションから説明する。

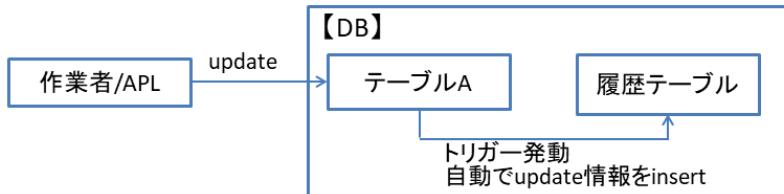
トリガーファンクションは、トリガーから呼ばれるファンクションである。

トリガーとは指定した動作が起こった際に起動する仕組みである。

トリガーとファンクションの関係を以下に記載する。

## トリガーの概要

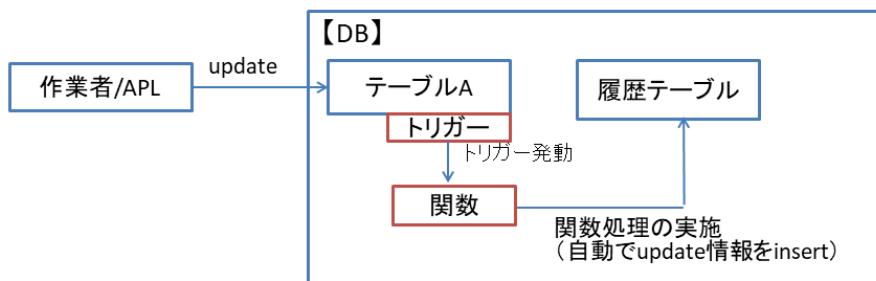
- ・ トリガーとは、特定の処理をされた時に同時に別処理を行う設定の事。
- ・ 例えば特定テーブルに処理が入った後、履歴としてその操作情報を履歴テーブルにinsertする、など。



- ・ 詳細としては、以下のようないみでる。
  1. トリガーが呼び出すファンクション(関数)を作成。  
`create function文`  
# 独自の記述法(PL/pgSQL等)がある。プログラミングチック(ifやforが使える)。
  2. トリガーを作成し、テーブルに付与。  
`create trigger文`

## トリガーのイメージ詳細化

- ・ トリガーをテーブルに付与し、テーブル更新等のイベントを検知したら指定のファンクションを起動する、といいうイメージ。



- ・ 仕組みは先に処理を行う関数を作り、次に呼び出し条件等を指定するトリガーを作成する。
  1. トリガーが呼び出すファンクション(関数)を作成。  
`create function ~ begin 処理 end ~ $$ language 使用言語(pgsql)`
  2. トリガーを作成し、テーブルに付与。  
`create trigger ~ {after|before (いつ実行か)} on table名 ~ execute function funcstion名`

上記の通り、登場人物は「トリガー」と「ファンクション」があるが、まずはファンクションの記法について触れる。

前述の通り、PL/pgSQLを使った書式等を以下に記載する。

## 関数の書き方について①

- 「plpgsql」形式での書式を以下に記載する

```
create function 関数名() returns trigger as $任意の文字$
begin
 处理
 return 戻り値;
end;
$任意の文字$
language plpgsql;
```

【例】呼び出されたらtbl1に「get up func」と入れる

```
create function func_history() returns triggers as $func_his$
begin
 insert into tbl1(ins_time, event_txt) values (now(), 'get up func') ;
 return null ;
end ;
$func_his$ language plpgsql ;
```

## 関数の書き方について②

- 関数内では特別な意味を持つキーワードがある。  
それを活用する事で更なる高度な処理が可能。

| キーワード         | 意味                                                                 |
|---------------|--------------------------------------------------------------------|
| new           | 対象テーブルにinsert/updateをしてトリガーが起動した際の<br>挿入/更新後した新しい値                 |
| old           | 対象テーブルにupdate/deleteをしてトリガーが起動した際の、<br>更新前/削除前の古い値                 |
| tg_name       | 呼び出し元トリガーの名前                                                       |
| tg_when       | 呼び出し元トリガーがどのタイミングで関数を呼んだか(before,after)                            |
| tg_level      | 呼び出し元トリガーで行ごとに関数呼ぶ設定か、<br>SQL文ごとに関数を呼ぶ設定か                          |
| tg_op         | 呼び出し元トリガーが起動した操作。<br>テーブルにinsert/update/delete/truncateのどの処理をした時か。 |
| tg_table_name | トリガー呼び出し元のテーブル名。                                                   |

## 関数の書き方について③

- 関数内では IF 文を指定して条件に応じた処理ができる。  
前ページのキーワードと組み合わせて使うと良い。

【例】

```
create function func_history() returns trigger as $func_his$
begin
if (tg_op = 'INSERT') then
 insert into tbl1(ins_time, ivent_txt, new_data)
 values (now(), 'insert', new.name) ;
elseif (tg_op = 'UPDATE') then
 insert into tbl1(ins_time, ivent_txt , old_data, new_data)
 values (now(), 'update', old.name, new.name) ;
elseif (tg_op = 'DELETE') then
 insert into tbl1(ins_time, ivent_txt , old_data)
 values (now(), 'delete', old.name) ;
end if ;
return null ;
end ;
$func_his$ language plpgsql ;
```

If文で、insert/update/delete  
で処理をかえる

new,oldを使い、  
変更前の名前データと  
変更後の名前データを  
それぞれ履歴に登録する

ファンクションに変更があった場合、毎回削除して作り直すのが面倒である。

create or replace function 構文を使用すると、指定した名前のファンクションがなければ作成、存在すれば更新、という動作となる。

## ファンクションの更新と確認について

- ファンクションの更新は  
**create or replace function ~ 文**  
を使用する。  
※ ~ 以降はcreate functionと同じ

pg\_XXXとつくテーブルは最初からある  
テーブルと思ってもらってよい。  
システムカタログと呼び、  
PostgreSQLがシステム的に管理している  
情報などが見れる。

- 上記は「存在しなければ作成」し、「存在すれば更新」する。
- 作ったファンクションの名前と定義の確認は  
以下を実行する。  
**select proname, prosrc from pg\_proc  
where proname = 'func名';**

次にトリガーの記載方法を示す。トリガーはSQLで作成する。

## トリガーの書き方

- 関数の呼び元のトリガーは以下のように記載する

```
create trigger トリガー名 {before|after} {イベント}
on table名 { for each row | statement}
execute function function名
```

ターゲットのtableで  
影響の当たった行ごとに  
関数起動するか、SQL文で  
1回のみ起動とするか。

※ イベント: insert, update, delete, truncate

### 【例: func\_history()を起動するトリガー】

```
create trigger func_his_tg after
insert or update or delete on targetTbl
for each row
execute function func_history();
```

これにより、targetTblに  
insert/update/deleteが発生したら  
その記録をtbl1に記録されるように  
なる

ここまで「何かの動作があった際に」自動で起動して処理を行うトリガーとファンクションについて説明してきた。  
これによく似た「ルール」と呼ばれる仕組みがある。

ルールは該当のテーブルに対し、指定された処理がされた場合に「追加で処理する」か「指定された処理を別の処理に置換で処理」することを定義するものである。

動作的にはトリガーファンクションに似ているが、PL/pgSQLを使わず、SQLのみで定義する。

## ルールについて

- 特定のSQL処理を使用とした際に、他のSQL操作に切り替える事を主として使うルールという仕組みもある。  
書式は以下。

```
create rule rule名 as on { select | insert | update | delete }
to table名 do { also | instead } 変換SQL文 ;
```

※ alsoは特定操作を受けた際に他SQLも実施、  
insteadは特定操作の代わりに他SQLを実施

### 【例 tbl1へのupdateを禁止しており、その記録を残す場合】

```
create rule upt_rule as on update
to tbl1 do instead insert into tbl1(ivent_time, ivent_txt)
values (now() , 'upt_rule get up') ;
```

#### 【補足】

イメージはビュー(view)に近い。  
ビューは既存テーブルから一部の情報しか表示されない  
仮想テーブルを定義するものだが、テーブルを実際に作っている  
わけではなく、呼び出す際に毎回ビュー用SQL文を打っている。

最後にトリガーやルールの確認方法や削除方法を記載する。

## トリガー、ルールの更新、確認方法

- トリガーとルールは alter 文や replace 文がない  
よって一度古いものを削除した後に  
新たなものを作成する、という手順となる。

- トリガーのトリガー名、トリガー起動テーブル、  
呼び出しファンクション名の確認方法は以下。

```
select tgname, t.tgrelid::regclass, proname from pg_trigger t
join pg_proc p on t.tgfoid = p.oid where tgname = 'trigger名';
```

- ルールの確認方法は以下。

```
Select * from pg_rules where rulename = 'rule名'
```

## トリガー/ファンクション/ルールの削除

- それぞれ以下を実施。

### 【トリガー】

```
drop trigger trigger名 on table名 ;
```

### 【ファンクション】

```
drop function function名 ;
```

### 【ルール】

```
drop rule rule名 on table名 ;
```

## (2).ストアドファンクションとストアドプロシージャ

次にストアドファンクションについて説明する。

トリガーファンクションはトリガーに呼び出されるファンクションだったが、ストアドファンクションはユーザから呼び出されるファンクションである。

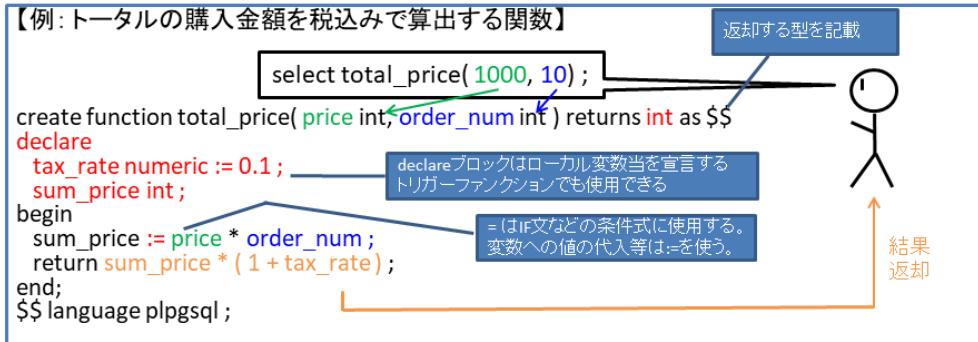
ユーザが呼び出す際に、関数に渡す値（引数）を指定する事で、関数内で決められた処理をして結果をユーザに返す。

アプリを主でやる方には（プログラムの世界でいう）オブジェクト指向とメソッドの考え方方に近い、というと伝わりやすいかもしれない。

使い方は、sum や avg 等の PostgreSQL に用意してある関数と同じようなイメージである。

### ストアドファンクションについて

- トリガーと合わせてファンクションについて触れた。  
これはトリガーから呼びだされることから「トリガーファンクション」と呼ばれる。  
ファンクションの使い方はトリガー以外にアプリ等から呼び出す「ストアドファンクション」がある。
- トリガーファンクションとの違いは「引数」を渡すこと。  
かわりにnew, old, tg\_table\_nameといったキーワード（トリガ変数）は使えない  
→ 良く実行する処理などをまとめておく。ユーザ定義関数と思えば良い。



ストアドファンクションとあわせておさえておきたいのが、(ストアド)プロシージャである。

ストアドファンクションと同じようにユーザ定義関数が作られるが、大きな違いはプロシージャ内でトランザクション操作を指定できる点にある。

## ストアドプロシージャの定義

- ・ ファンクションをより高度化したものに(ストアド)プロシージャがある。  
大きな違いは関数内でトランザクション操作が実行可能かという点にある。  
構文は以下。  
`create procedure procedure名(引数) language plpgsql as $$ begin 処理 end $$;`
  - ・ よく使う機能をプロシージャとしてまとめておくことができる。  
例えば以下でトランザクションでよく出てきがちな  
銀行口座システムの入金処理のプロシージャを記載する。

```
【例】
create or replace procedure transfer_money(from_account int, to_account int, money int)
language plpgsql as $$

begin
if from_account > 0 and to_account > 0 then
 update accounts set account_money = account_money - money where account_id = from_account;
 update accounts set account_money = account_money + money where account_id = to_account;
 commit ;
else
 rollback ;
end if ;
end;
$$;
```

口座(accounts)テーブルの、  
from\_accountさんの口座金額(account\_money)から入金額(money)をひいて、  
to\_accountさんの口座の金額に入金分を増やす

トランザクション処理。  
条件に一致するものだけ、まとめて登録。

プロシージャの呼び出しには call 文を使用する。

## ストアドプロシージャの呼び出し

- 定義したプロシージャはcall文で呼び出す。  
以下、文法。

call プロシージャ名(引数)

※ 引数はプロシージャに渡す値。

**【例】**  
call transfer\_money( 1, 2, 1000)

アカウント1さんから、アカウント2さんに  
1000円の入金があった、という処理。

※ ファンクションは、トリガーからの呼び出しとユーザ/アプリからの呼び出し(select等)の2つがユースケースとしてあった。プロシージャはユーザ/アプリからの呼び出し(call)のみで、トリガー呼び出しはできない。

プロシージャに関するその他処理を以下に記載する。

## ストアドプロシージャのその他処理

- 変更はfunctionと同様にreplaceを使う。

`create or replace procedure～文`

- 確認もfunctionと同様にpg\_procテーブルを確認する

`select * from pg_proc where proname = 'procedure名';`

- 削除は以下

`drop procedure procedure名 ;`

### (3).カーソルについて

各ファンクション、およびプロシージャとあわせて覚えておくと良いのがカーソルである。

カーソルをファンクションやプロシージャに埋め込むことができる。

### カーソルについて①

- ファンクション(関数)やプロシージャとあわせて覚えておくと良いものに「カーソル」がある。
- カーソルとは、取得した結果を1行ずつ取得する仕組みの事である。
- 基本的な使い方は以下。(トランザクション内で使用する)  
定義: `declare カーソル名 cursor for select ~ ;`  
情報取得: `fetch [next/prior/first/last] from カーソル名 ;`  
カーソルのクローズ: `close カーソル名 ;`
- ファンクションやプロシージャでは「declare」ブロックにて  
カーソルを定義し、beginブロック内で情報取得等を実施する。  
(例は次ページ)  
⇒ 1行ごと取得する事で、取得した値によって処理を変化  
させる事ができる。(より細かい制御が可能となる。)
- ファンクション・プロシージャ以外に埋め込みSQLとして使われることもある。

#### [補足]埋め込みSQLとは

CやCOBOL等の歴史あるプログラムでDBを操作する為の記載法。  
`exec sql begin declare section` 「get descriptor」などがある。

## カーソルについて②

### 【ファンクションでの使用法】

```
create function 関数名 returns 戻り値 as $$
declare
 カーソル名 cursor for select ~
begin
 open カーソル名 ;
 return next カーソル名 ;
 close カーソル名 ;
end ;
$$ language plpgsql ;
```

Declare ブロックで指定する為、先頭の declare を再指定する必要なし

関数やプロシージャ内では open を指定する必要あり

### 【プロシージャでの使用法】

```
create procedure プロシージャ名 as $$
declare
 以下、ファンクションと同様
```

### (4).教科書編のおわりに

以上をもって教科書編は完了となる。

これまで様々な操作を記載してきたが、1章でも触れた通り、本書はDBの操作に焦点を当てている点は改めて認識と理解をお願いしたい。

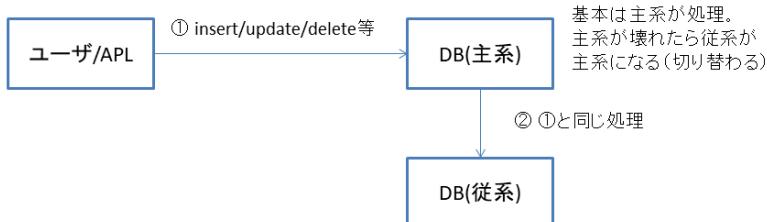
DBに関しては他にも設計観点(テーブル設計や冗長構成(レプリケーション※)の有無、各種設定等)や運用観点(バックアップ・リストア等)の知識も重要である。

もし本書を通してDBに興味をもってもらえたなら、次のステップに向けて今度は上記領域についても調べてみると良いだろう。

\* 参考までに次ページにレプリケーションのイメージを記載する。

## [参考] レプリケーションとは

- DBサーバを2つ以上用意し、冗長構成をとった際に、主系(メインで使うDB)に不具合が生じた際に、すぐに従系(予備DB)に切り替えられるようにDBの仕組みを使って、2台のDBが同じ状況になるように連携させる仕組み。  
実現にはコマンドやSQLではなく、設定を変更して対応する。(postgresql.conf等)



- レプリケーションにも種類がある。  
PostgreSQLでは以下2つがある。
  - ストリーミングレプリケーション(SR)
  - ロジカルレプリケーション

またこのまま終わるのではなく、第二部の問題を解いて、実際に手を動かす事で理解を深めてほしい。  
深く理解できた分だけ、現場で活用する機会も広がるだろうと思う。

本書が読者の皆様に少しでも役立つ事を願っている。

# 問題編

以降ではハンズオン形式の問題を実行する為の環境構築と、各問題等を記載する。  
教科書編を見直しつつ、是非取り組んでみてほしい。

## 環境構築

以降でハンズオン形式の問題を動かしながら解く為の環境構築手順を記載する。

### (1).本書の前提

PostgreSQLを扱いたいとき、昨今は以下のように様々な手段がとれる。

- ① Windows 端末に Windows 版 PostgreSQL をインストールする
- ② Windows 端末に WSL2 で Linux 環境を作り、その上に PostgreSQL をインストールする
- ③ Linux の仮想マシンを用意し、その上に PostgreSQL をインストールする
- ④ Linux マシンを用意(物理マシンや Raspberry Pi 等)して、その上に PostgreSQL をインストールする
- ⑤ コンテナ技術(docker コンテナ等)を活用して、PostgreSQL を扱う
- ⑥ Web ブラウザ(Chrome や Edge)から PostgreSQL を扱えるサービス(SQL Fiddle 等)を使う

本書の動作確認は③で実施している。

①②⑤⑥では一部の手順が実施できない可能性がある為、注意してほしい。

③は AWS 等のパブリッククラウドでも、Virtual box や VMware、Vagrant 等の仮想化ソフトウェアを使って自身の端末に仮想マシンを用意する案でも、どちらでも構わない。

以降で Linux OS の仮想マシンを用意した状態からの PostgreSQL インストール 及び セットアップ手順を記載する。

なお、動作確認した Linux OS は Ubuntu20.04、PostgreSQL は 12 版である。

OS バージョンの違いによりインストール手順が変わる他、PostgreSQL のバージョン差分で動作が異なる可能性がある為、他バージョンで実施する場合は注意してほしい。

### (2).Ubuntu20.04 で PostgreSQL12 をインストールする方法

Ubuntu20.04 のマシンにて以下コマンドを 1 行ごとに実行する。

```
sudo apt update

sudo apt install postgresql-12 postgresql-client-12 postgresql-contrib
```

\* 2 つ目のコマンドで「Do you want to continue? [Y/n]」と表示されたら、「Y」を入力する。

### (3).インストール後のセットアップ

問題なく PostgreSQL がインストールされた場合、PostgreSQL は自動起動している。

一度 PostgreSQL を停止した上、本研修用の環境を作っていく。

以下のコマンドを 1 行ずつ実施する。

#### PostgreSQL の停止

```
sudo su - postgres

echo 'export PATH=$PATH:/usr/lib/postgresql/12/bin' >> ~/.bashrc

source ~/.bashrc

pg_ctl stop -D /var/lib/postgresql/12/main/
```

1 つ目のコマンドで、コマンド入力欄の横の表示が変わっていること。

以下に例を記載する。

```
vagrant@ubuntu-focal:~$
vagrant@ubuntu-focal:~$
vagrant@ubuntu-focal:~$ sudo su - postgres
postgres@ubuntu-focal:~$
postgres@ubuntu-focal:~$
```

上記では「vagrant」ユーザで apt によるインストールをし、postgres ユーザに切り替えている。

上記のように表示が変わっていれば問題ない。

2 つ目と 3 つ目のコマンドで PostgreSQL のパスを設定している。

これを実行しないと、pg\_ctl を実行した際に「pg\_ctl: command not found」と表示され失敗する。

\* 直接 /usr/lib/postgresql/12/bin/pg\_ctl のように指定して実行する手もあるが、毎回指定するのは面倒なのでパス指定をする。

4 つ目のコマンドで初期セットアップされている DB サーバ(PostgreSQL のプロセス)を停止する。 \* /var/lib/postgresql/12/main/ に初期セットアップされた DB クラスタがある

4 つ目のコマンドで「server stopped」と出れば問題ない。

## 本書の問題用に初期セットアップと動作確認

以下コマンドを1行ずつ実施する。

```
initdb -D /var/lib/postgresql/12/data01 --no-locale --encoding=utf8
pg_ctl -D /var/lib/postgresql/12/data01 start
```

1つ目のコマンドで「Success. You can now start the database server using:」と出れば問題ない。  
また、2つ目のコマンドで「server started」と表示されれば想定通り動かせている。

### 動作確認

以下を実行し、SQL実行モードに接続できるか確認する。

```
psql
```

コマンドを打つ左側の表示が以下のように切り替わっていれば、問題ない。

```
postgres@ubuntu-focal:~$ psql
psql (12.20 (Ubuntu 12.20-0ubuntu0.20.04.1))
Type "help" for help.
```

```
postgres=#
postgres=# █
```

これでDBへの接続確認は完了である。

必要に応じて「\q」で抜ける。

\* このまま問題に入っても良い。

なお、本書ではPostgreSQL自動起動設定はしないものとする。

マシン再起動時には再度pg\_ctlで起動する事。

### (4). 好みの設定

以降は好みに併せて確認してほしい。(必須ではない)

これまでPostgreSQLのセットアップは念のためフルパスで指定しているが、postgresユーザのホームディレクトリは「/var/lib/postgresql」のはずである。

以下を実行して問題なく実行できれば、以降は~/を活用しても良い。

```
pg_ctl restart -D ~/12/data01
```

また pg\_ctl で毎回-D を指定するのが面倒な場合、PGDATA 変数に上記を覚えさせると良い。  
以下、参考に手順を記載するが、本書では PGDATA を登録しない想定で以降を記載する。  
これを実行しておけば「pg\_ctl 操作(start/restart/stop)」だけで実行が可能となる。

```
export PGDATA=~/12/data01
```

\* 上記は一時的な設定であり、接続を閉じると消えてしまう。設定永続化したい場合は PostgreSQL のパスを張った時同様、.bashrc に記載すること。

#### (5). PostgreSQL が上手く動かなくなってしまった場合

万が一 PostgreSQL が上手く動かなくなったり起動できなくなった場合、以下手順で初期化できる。

必要に応じて活用してほしい。

\* ただしデータも含めて初期化される。また下記手順は「/var/lib/postgresql/12/data01」に DB クラスタを設定した場合を想定している。問題により data01 ではない箇所に DB クラスタを配置する場合もある為、異なる DB クラスタに不具合が生じた場合は下記手順を適宜読み替えて実施する事。

```
sudo su - postgres

pg_ctl -D /var/lib/postgresql/12/data01 stop

rm -rf /var/lib/postgresql/12/data01/*

initdb -D /var/lib/postgresql/12/data01 --no-locale --encoding=utf8

pg_ctl -D /var/lib/postgresql/12/data01 start
```

1つ目のコマンドは postgres ユーザに切替済であれば、不要である。

また、3点目のコマンドはファイルをまとめて削除するコマンドであるが、指定するディレクトリを誤ると仮想マシン自体が故障する可能性もある為、特に注意して実行する事。

# 各種問題

以降にハンズオンで動かせる問題を記載する。

## はじめに

### 【問題の読み方と問題を解いていくまでの前提】

問題編の読み方は以下の通り。

## 問題編の読み方

- 章ごとにテーブルは分ける為、章の問題を全て解かなくても次に進めます。
- ヒントは章をわけて、最後に記載しています。  
問題に着手し、全くわからなければ適宜確認ください。
- 回答が思い浮かばなければ正解をみて動かしてみてください。  
なお、回答はあくまで「例」ですので他案でも同様の事(期待出力)となれば正解です。  
回答は「はじめに」に記載のサイトからダウンロードください。  
→コピペだけで済ますのはあまりお勧めしません。  
書いて覚えるのが一番かと思います。
- 問題の中には失敗するものもあります。期待出力欄を確認しましょう。  
失敗した場合、次の問題に行く前に「なぜ失敗したか」少し考えてみましょう。  
→期待出力は「上から順番に問題を実施した場合」を想定しています。  
また、時間やプロセスID等は環境によっても変わります。
- 赤字の問題は余裕がある人・より細部の動きを知りたい人向けです。  
飛ばしても構いません。
- 【advance】についているものは教科書に記法を意図的に記載していないものとなります。  
調べてみる等、工夫しながら進めてみてください。  
→次問題以降もadvance含めて実施した事を前提に  
問題を作っています。悩んで難しそうなら答えを見て、  
別途なぜそうなるのか考えてみてください。

各章(DDL や DCL 等)はテーブルを分けている為、並行して実施できるが、各章内は上から問題を実施する事を前提としている。

なお、各章の問題では(DDL 等の DML の前の問題であっても)簡易な DML ( insert/update/delete)を実行する場合がある。データの登録・更新・削除をする旨の問題が登場した場合は、教科書編「5 章 5-2 節」を参照してほしい。

### 【ヒントの使い方】

各問題とヒント欄には「● Q 数値 タイトル」とついている為、これをコピペして検索すると、問題とヒント間で即座に移動できる。

例えば最初の問題には「● Q1 DB 作成」とついている為、問題を解いている際にヒントが見たくなれば、これで検索すればすぐヒントに飛んで確認できるし、ヒントを見た後、再度検索すればすぐ問題まで戻ってこれる。

この方法を活用して、うまく問題を解いてもらいたい。

ただしヒント自体がない問題もある。ないものは問題の丸数字(①や②等)自体の記載がない。

### **【補足の使い方】**

問題に関する「補足」がある場合には「※補足あり」と記載する。

補足の内容は、回答に関する補足等であり、「問題回答後」に確認する事を前提として記載している。

補足には「★ Q 数値 タイトル」とつけており、問題番号(Q 数値)とタイトルは問題と同じである。

その為、問題を解いた直後に補足を確認する場合、ヒントと同様に検索で飛んでほしい。(その際には先頭の「●」を「★」に変えて検索する事)

### **【問題分布について】**

問題分布は以下の通り。

問題数は多いが全て解く事が目的なのではなく、理解する事が目的である。

「問題の読み方と問題を解いていくまでの前提」にも記載の通り、各章の問題は並行して進められる為、上の章から順番に進めていくでも良いし、気になった箇所を触れる、という進め方でも良いのではないかと考える。

## **問題構成**

以下のような構成で問題を用意。

| 章       | 問題範囲     | 問題数(割合)   |
|---------|----------|-----------|
| DDL     | Q1～17    | 17問 (10%) |
| DCL     | Q18～25   | 8問 (5%)   |
| TCL     | Q26～45   | 20問 (12%) |
| DML(基礎) | Q46～76   | 31問 (18%) |
| DML(応用) | Q77～114  | 38問 (22%) |
| 高度な操作   | Q115～156 | 42問 (24%) |
| +α      | Q157～171 | 15問 (9%)  |

## 1章 DDL問題

---

### ● Q1 DB 作成

開始の前提：「環境構築」章をみて initdb や pg\_ctl 実行し、psql で接続完了した状態を問題開始状態とする

#### 【問題】

- ① 演習用のDB(test01)を作成
- ② DBが作られたかメタコマンドを使用し確認

#### 【期待出力】

- ① CREATE DATABASE

②

| List of databases |          |          |         |       |                   |
|-------------------|----------|----------|---------|-------|-------------------|
| Name              | Owner    | Encoding | Collate | Ctype | Access privileges |
| test01            | postgres | UTF8     | C       | C     |                   |

### ● Q2 table 作成(public)

## 【問題】

- ① postgres DBから、test01 DBにインし直す
- ② test01内で以下(左)のような新テーブル(class)を作成
- ③ test01内で以下(右)のような新テーブル(student)を作成

|       |             |            |       |                                           |  |                    |          |
|-------|-------------|------------|-------|-------------------------------------------|--|--------------------|----------|
| 名前    | class_id    | class_name | 名前    | id                                        |  | name               | class_id |
| 型     | int         | varchar(5) | 型     | int                                       |  | text               | int      |
| その他指定 | Primary key |            | その他指定 | Primary key<br>(別名:student_id_pkey_alias) |  | not null<br>unique |          |

④ 作成したテーブル情報を参照する。

## 【期待出力】

- ① プロンプトが変化  
インし直し前: postgres=#  
インし直し後: test01=#
- ②③ CREATE TABLE
- ④ 以下の枠参照

| Table "public.class"                       |                      |           |          | Table "public.student" |         |           |          |
|--------------------------------------------|----------------------|-----------|----------|------------------------|---------|-----------|----------|
| Column                                     | Type                 | Collation | Nullable | Column                 | Type    | Collation | Nullable |
| class_id                                   | integer              |           | not null | id                     | integer |           | not null |
| class_name                                 | character varying(5) |           |          | name                   | text    |           | not null |
| Indexes:                                   |                      |           |          |                        |         |           |          |
| "class_pkey" PRIMARY KEY, btree (calss_id) |                      |           |          |                        |         |           |          |

※ ④は赤字の箇所を特に注意して確認する事

### ● Q3 table 情報更新

## 【問題】

- ① classテーブルに「nickname」列(型: text)を追加。
- ② class テーブルのclass\_name列をname列に変更。
- ③ classテーブルのname列をvarchar(5)からtextに変更
- ④ student テーブルのname列の「not null」制約解除。
- ⑤ student テーブルのname列の「unique」制約解除。
- 【advance】⑥ studentテーブルのclass\_idに、classテーブルのclass\_idを外部参照する設定追加。
- 【advance】⑦ classテーブルのname列に常に3文字であるチェック制約を追加

|       |             |       |          |
|-------|-------------|-------|----------|
| class | class_id    | name  | Nickname |
| 型     | Int         | text  | text     |
| その他指定 | Primary key | check |          |

|         |                                           |                    |            |
|---------|-------------------------------------------|--------------------|------------|
| student | id                                        | name               | class_id   |
| 型       | int                                       | text               | int        |
| その他指定   | Primary key<br>(別名:student_id_pkey_alias) | not null<br>unique | references |

↑ 参照

## 【期待出力】

- ①②③④⑤⑥⑦ ALTER TABLE

## ■ Q3 完了時点のテーブル定義

### [参考] ここまでテーブル定義期待出力

- ¥d [table名] の結果(期待出力)を以下に示す。

| Table "public.class"                               |                    |                       |            | テーブル名           |
|----------------------------------------------------|--------------------|-----------------------|------------|-----------------|
| Column                                             | Type               | Collation             | Nullable   | Default         |
| class_id                                           | integer            |                       | not null   |                 |
| name                                               | text               |                       |            |                 |
| nickname                                           | text               |                       |            |                 |
| Indexes:                                           |                    |                       |            |                 |
| "class_pkey"                                       | PRIMARY KEY, btree | (class_id)            |            |                 |
| Check constraints:                                 |                    |                       |            |                 |
| "class_name_check"                                 | CHECK              | (name ~~ '____':text) |            |                 |
| Referenced by:                                     |                    |                       |            |                 |
| TABLE "student" CONSTRAINT "student_class_id_fkey" | FOREIGN KEY        | (class_id)            | REFERENCES | class(class_id) |

| Table "public.student"   |                    |            |            |                 |
|--------------------------|--------------------|------------|------------|-----------------|
| Column                   | Type               | Collation  | Nullable   | Default         |
| id                       | integer            |            | not null   |                 |
| name                     | text               |            |            |                 |
| class_id                 | integer            |            |            |                 |
| Indexes:                 |                    |            |            |                 |
| "student_id_pkey_alias"  | PRIMARY KEY, btree | (id)       |            |                 |
| Foreign-key constraints: |                    |            |            |                 |
| "student_class_id_fkey"  | FOREIGN KEY        | (class_id) | REFERENCES | class(class_id) |

## ● Q4 シーケンス追加とテーブル情報変更

### 【問題】

① シーケンス(student\_id\_seq)を作成(細かい指定はなし)。

【Advance】② studentテーブルのidにデフォルト値でシーケンスを設定。

③ メタコマンドでstudentテーブル状態確認。

### 【期待出力】

- CREATE SEQUENCE
- ALTER TABLE
- 以下の青枠

| Table "public.student"   |                    |            |            |                                     |
|--------------------------|--------------------|------------|------------|-------------------------------------|
| Column                   | Type               | Collation  | Nullable   | Default                             |
| id                       | integer            |            | not null   | nextval('student_id_seq'::regclass) |
| name                     | text               |            |            |                                     |
| class_id                 | integer            |            |            |                                     |
| Indexes:                 |                    |            |            |                                     |
| "student_id_pkey_alias"  | PRIMARY KEY, btree | (id)       |            |                                     |
| Foreign-key constraints: |                    |            |            |                                     |
| "student_class_id_fkey"  | FOREIGN KEY        | (class_id) | REFERENCES | class(class_id)                     |

### ● Q5 シーケンス関連動作の確認

#### 【問題】

- ① 新シーケンス(test\_seq)作成。  
条件は10から開始で、10ずつインクリメント。30で終わり、とする。
- ② test\_seqの現在の値を確認。
- ③ test\_seqの次の値を取得。
- ④ test\_seqの現在の値を確認。
- ⑤ ③の次値の取得を2回実施し、値を使い切り、その後再度次値取得するとどうなるか確認。
- ⑥ test\_seqの値を0に戻す。
- ⑦ test\_seqの値を1に戻す。
- ⑧ test\_seq削除。

#### 【期待出力】

```

① CREATE SEQUENCE
② ERROR: curval of sequence "test_seq" is not yet defined in this session
③ ④ curval
nextval -----
----- 10
----- 10

⑤ ERROR: nextval: reached maximum value of sequence "test_seq" (30)
⑥ ERROR: setval: value 0 is out of bounds for sequence "test_seq" (1..30)
⑦
setval

1

⑧ DROP SEQUENCE

```

---

### ● Q6 index付与

#### 【問題】

- ① studentテーブルのnameにindex(student\_name\_index)付与
- ② classテーブルのclass\_idにindex(class\_id\_manual\_index)付与
- 【advance】③ studentテーブルのidとclass\_idで1つの  
index(multi\_column\_index)付与(それぞれに付与ではない)
- ④ ②と③のindex削除

| class | class_id    | name  | nickname |
|-------|-------------|-------|----------|
| 型     | Int         | text  | text     |
| その他指定 | Primary key | check |          |

| student | id                                                                          | name                          | class_id   |
|---------|-----------------------------------------------------------------------------|-------------------------------|------------|
| 型       | int                                                                         | text                          | int        |
| その他指定   | Primary key<br>(別名:<br>student_id_pkey_alias)<br><br>default student_id_seq | Index<br>(student_name_index) | references |

参照

#### 【期待出力】

- ①②③ CREATE INDEX
  - ④ DROP INDEX
-

### ● Q7 制約チェック

#### 【問題】

- ① 外部参照制約のチェックを行う。Studentテーブルにname(AAA),class\_id(1)を指定して投入する。
- ② primary key(not null)のチェックを行う。classテーブルでname(1-1)を入れるinsert文を投入する
- ③ check制約のチェックを行う。Classテーブルでclass\_id(1), name(1-1-1)を入れるinsert文を投入する。
- ④ classテーブルに正しい値を入れる(class\_id 1, name 1-1)
- ⑤ primary key(unique)のチェックを行う。Classテーブルで class\_id(1), name(1-2)を入れるinsert文を投入する。
- ⑥ default指定のチェックを行う。studentテーブルにname(AAA)とclass\_id(1)を入れるinsert文を投入する。
- ⑦ 外部参照制約のチェックを行う。Classテーブルで、class\_id(1)のレコードを削除する
- ⑧ studentテーブルのclass\_id(1)レコードを削除した後、classテーブルでclass\_id(1)を削除する

#### 【期待出力】

- ① ERROR: insert or update on table "student" violates foreign key constraint "student\_class\_id\_fkey"  
DETAIL: Key (class\_id)=(1) is not present in table "class".
- ② ERROR: null value in column "class\_id" violates not-null constraint  
DETAIL: Failing row contains (null, 1-1, null).
- ③ ERROR: new row for relation "class" violates check constraint "class\_name\_check"  
DETAIL: Failing row contains (1, 1-1-1, null).
- ④ INSERT 0 1
- ⑤ ERROR: duplicate key value violates unique constraint "class\_pkey"  
DETAIL: Key (class\_id)=(1) already exists.
- ⑥ INSERT 0 1
- ⑦ ERROR: update or delete on table "class" violates foreign key constraint "student\_class\_id\_fkey" on table "student"  
DETAIL: Key (class\_id)=(1) is still referenced from table "student".
- ⑧ DELETE 1 × 2

| class | class_id    | name  | nickname |
|-------|-------------|-------|----------|
| 型     | int         | text  | text     |
| その他指定 | Primary key | check |          |

| student | id                                                                   | name                      | class_id   |
|---------|----------------------------------------------------------------------|---------------------------|------------|
| 型       | int                                                                  | text                      | int        |
| その他指定   | Primary key<br>(別名: student_id_pkey_alias)<br>default student_id_seq | Index(student_name_index) | references |

参照

### ● Q8 indexメンテナンス

#### 【問題】

- ① studentテーブルのnameにある  
index(student\_name\_index)の並び替えの実施。
- ② studentテーブルのnameにある  
index(student\_name\_index)の作り直しを実施。

#### 【期待出力】

- ① CLUSTER
- ② REINDEX

### ● Q9 ビューとマテリアルズドビューの作成

**【問題】**

- ① classテーブルに「1, 1-1, view」(class\_id, name, nickname)という情報を登録する
- ② classテーブルからclass\_idとnicknameのみ表示するビュー(test\_view)を作成する
- ③ ②と同じ条件のマテリアルズドビュー(test\_mate\_view)を作成する
- ④ ②③で作成したビュー、マテビューを確認する

[class]

| class_id | name | nickname |
|----------|------|----------|
| 1        | 1-1  | view     |

[test\_view]

| class_id | nickname |
|----------|----------|
| 1        | view     |

[test\_mate\_view]

| class_id | nickname |
|----------|----------|
| 1        | view     |

**【期待出力】**

- ① INSERT 0 1
- ② CREATE VIEW
- ③ **SELECT 1** ※ CREATE系の出力ではなくselectなので注意

④ [test\_view]  
class\_id | nickname  
-----+-----  
1 | view

④ [test\_mate\_view]  
class\_id | nickname  
-----+-----  
1 | view

### ● Q10 ビューとマテリアルズドビューの動作確認: 参照元の変更とその影響

**【問題】**

- ① classテーブルに「2, 2-1, mate\_view」(class\_id, name, nickname)という情報を登録する
- ② test\_view, test\_mate\_viewそれぞれのビューから上記変更が見えるか確認する
- ③ classテーブルにてclass\_idが2のレコードのnicknameを「view\_r2」に変更する  
その後、更にclassテーブルからclass\_idが1のレコードを削除する
- ④ test\_view, test\_mate\_viewそれぞれのビューから上記変更が見えるか確認する
- ⑤ マテビュー(test\_mate\_view)に対し、情報の最新化を行う  
その後、マテビューを確認する

**【期待出力】**

- ① INSERT 0 1

- ②④ 青枠参照

- ③ UPDATE 1  
DELETE 1

- ⑤ REFRESH MATERIALIZED VIEW

マテビュー参照結果は青枠参照

② [test\_view]  
class\_id | nickname  
-----+-----  
1 | view  
2 | mate\_view

② [test\_mate\_view]  
※ マテビューは変更が見えない  
class\_id | nickname  
-----+-----  
1 | view

④ [test\_view]  
class\_id | nickname  
-----+-----  
2 | view\_r2

④ [test\_mate\_view]  
※ マテビューは変更が見えない  
class\_id | nickname  
-----+-----  
1 | view

⑤ [test\_mate\_view]  
※ 表示が最新化されている  
class\_id | nickname  
-----+-----  
2 | view\_r2

● Q11 ビューとマテリアライズドビューの動作確認: ビューに対する変更処理

【問題】

- ① test\_viewビューに「3, view01」(class\_id, nickname)という情報を登録する
- ② test\_mate\_viewに「4, mate\_view01」という情報を登録しようとする
- ③ test\_viewビューにてclass\_idが3のレコードのnicknameを「view02」に更新する
- ④ test\_mate\_viewにてclass\_idが2のレコードのnicknameを「mate\_view99」に更新
- ⑤ test\_viewと、classテーブルを確認する。  
(test\_viewの更新が、参照元のclassテーブルに影響を与えていたか確認する)
- ⑥ test\_viewビューからclass\_idが3のレコードを削除
- ⑦ test\_mate\_viewからclass\_idが2のレコードを削除

【期待出力】

① INSERT 0 1

②④⑦ ERROR: cannot change materialized view "test\_mate\_view"

③ UPDATE 1

⑤ 青枠参照

⑥ DELETE 1

| ⑤ [test_view] |          | ⑤ [class] |        |
|---------------|----------|-----------|--------|
| class_id      | nickname | class_id  | name   |
| 2             | view_r2  | 2         | 2-1    |
| 3             | view02   | 3         | view02 |

\* 補足あり。問題確認後、参考推奨。本問題はヒントなし。

---

● Q12 ビューとマテリアライズドビューの作成

【問題】

- ① test\_view, test\_mate\_viewをそれぞれ削除する
- ② classテーブルを確認し、ビューの削除が参照元のテーブルに影響を与えていないか確認する
- ③ classのレコードを全て削除する(テーブル削除ではないので注意。)

【期待出力】

① DROP VIEW

DROP MATERIALIZED VIEW

② 青枠参照

③ DELETE 1

| ② [class]                |         |
|--------------------------|---------|
| ※ ビューの削除が参照元テーブルに影響を与えない |         |
| class_id                 | name    |
| 2                        | 2-1     |
| 3                        | view_r2 |

### ● Q13 table 作成(新スキーマ)

#### 【問題】

- ① 新スキーマ(test\_schema01, 02)を作成する
- ② test\_schema01ににおけるテーブル(test001)を作成する(カラム設定はid int, name text )
- ③ test\_schema01に作ったtest001をselectで参照する
- 【advance】④ 参照スキーマをtest\_schema02に切り替える
- ⑤ 参照スキーマを切り替えた後にtest\_shema02にテーブル(test001)を作成する  
(カラムは id int, name text)  
※ テーブル作成時にスキーマ名を省略して実施する事
- ⑥ test\_schema02に作ったtest001をselectで参照する(スキーマ名省略)
- ⑦ ⑤と同じテーブル作成を再度実施
- ⑧ 作ったスキーマとテーブル情報取得

#### 【期待出力】

- ① CREATE SCHEMA × 2
- ②⑤ CREATE TABLE
- ③⑥ 右記の青枠参照。
- ④ SET
- ⑦ ERROR: relation “test001” already exists
- ⑧ 右記の青枠参照。(作ったもののうち一部しか表示されない)

③⑥  
id | name  
---+---  
(0 行)

⑧ List of relations  
Schema | Name | Type | Owner  
-----+-----+-----+-----  
test\_schema02 | test001 | table | postgres

#### 【問題】

- ⑨ 参照スキーマをpublic、test\_schema01 , test\_schema02 とする
- ⑩ 作ったスキーマとテーブル情報取得
- ⑪ test\_schema02 のtest001 を test002にテーブル名変更
- ⑫ 作ったスキーマとテーブル情報取得
- ⑬ test\_schema02 のtest002 にコメントを記載する。
- ⑭ つけたコメント参照。
- ⑮ test\_schema02 のtest002 のコメントを削除する。
- ⑯ つけたコメント参照。

#### 【期待出力】

- ⑨ SET
- ⑩ 右記の青枠参照。(test\_schema02の値が出ない)
- ⑪ ALTER TABLE
- ⑫ 右記の青枠参照。(全て表示される)
- ⑬ COMMENT
- ⑭ 下記オレンジ枠。(箇所のみ記載。)

⑩ List of relations  
Schema | Name | Type | Owner  
-----+-----+-----+-----  
public | class | table | postgres  
public | student | table | postgres  
public | student\_id\_seq | sequence | postgres  
test\_schema01 | test001 | table | postgres

⑭ List of relations  
Schema | Name | Type | Owner | Size | Description  
-----+-----+-----+-----+-----+-----  
test\_schema02 | test002 | テーブル | postgres | 8192 bytes | sample

- ⑮ COMMENT
- ⑯ 下記オレンジ枠(必要箇所のみ記載)

⑯ List of relations  
Schema | Name | Type | Owner | Size | Description  
-----+-----+-----+-----+-----+-----  
test\_schema02 | test002 | テーブル | postgres | 8192 bytes |

⑫ List of relations  
Schema | Name | Type | Owner  
-----+-----+-----+-----  
public | class | table | postgres  
public | student | table | postgres  
public | student\_id\_seq | sequence | postgres  
test\_schema01 | test001 | table | postgres  
test\_schema02 | test002 | table | postgres

● Q14 スキーマ削除

【問題】

- ① 新スキーマ(test\_schema01)を削除
- ② test\_schema01のtest001テーブルを削除
- ③ 新スキーマ(test\_schema01)を削除

【期待出力】

- ① ERROR: cannot drop schema test\_schema01 because other objects depend on it  
DETAIL: table test001 depends on schema test\_schema01  
HINT: Use DROP ... CASCADE to drop the dependent objects too.
  - ② DROP TABLE
  - ③ DROP SCHEMA
- 

● Q15 スキーマ削除(まとめて削除)

【問題】

- [advance]① 新スキーマ(test\_schema02)を削除  
前問題のようにテーブルを削除せずに消すこと。

【期待出力】

- ① NOTICE: drop cascades to table test002  
DROP SCHEMA
-

● Q16 drop でエラーとさせない

【問題】

- ①これまでの問題で削除したはずのtest001を再度削除しようとする
- ②①のエラーを出力されないように処理させる

【期待出力】

- ① ERROR: table "test001" does not exist
  - ② NOTICE: table "test001" does not exist, skipping
- DROP TABLE
- 

● Q17 DB名変更とDB削除

【問題】

- ① DB名(test01)を変更(ex\_db)
- ② postgres DBに移動
- ③ DB名(test01)を変更(ex\_db)
- ④ DB(ex\_db)削除

【期待出力】

- ① ERROR: current database cannot be renamed
- ② プロンプトが「test01=#」から「postgres=#」に変化
- ③ ALTER DATABASE
- ④ DROP DATABASE

## 2章 DCL問題

### ■ 事前準備

練習問題実行前に以下を実行する事。

```
psql -U postgres -d postgres
create database test02 ;
\q

psql -U postgres -d test02

create table student(id int , name text) ;
insert into student(id, name) values (1, 'AAA'), (2, 'BBB'), (3, 'CCC') ;
```

### ● Q18 ロール作成

#### 【問題】

- ① ログイン、スーパーユーザ属性をつけたロール(superuser01)を作成。
- ② ログイン、パスワード属性をつけたロール(user01)を作成。
- ③ superuser01でtest02 DBにつなぎ直し。
- ④ user01でtest02 DBにつなぎ直し。
- ⑤ user01でtest02\_02 DBを作成。
- ⑥ postgresユーザでtest02 DBにアクセスし、superuser01にパスワード付与
- ⑦ superuser01でtest02にアクセスし直し、test02\_02 DB作成。

#### 【期待出力】

- ①② CREATE ROLE
- ③ イン成功
- ④ イン成功
- ⑤ ERROR: permission denied to create database
- ⑥ ALTER ROLE
- ⑦ CREATE DATABASE

\* 補足あり。問題回答後に参照推奨。

● Q19 権限付与

【問題】

- ① superuser01でstudentテーブルにselect実施。
- ② user01 でtest02 DBにつなぎ直して①と同様の select実施。
- ③ superuser01で入り直して、user01に全権限付与。
- ④ user01 でtest02 DBに入り直して、①と同様のselect実施。

【期待出力】

- ① 結果取得できる
  - ② ERROR: permission denied for table student
  - ③ GRANT
  - ④ 結果取得できる
- 

● Q20 ロールにロール付与

【問題】

- ① superuser01でtest02に入り直す。
- ② user02をログイン属性・パスワード属性(aaa)を付与して作成。
- ③ user02でtest02に入り直す。
- ④ studentテーブルのデータを取得。
- ⑤ superuser01でtest02に入り直す。
- ⑥ studentテーブルを参照できるuser01に、user02を参加させる。
- ⑦ user02でtest02に入り直す。
- ⑧ studentテーブルを参照する。

【期待出力】

- ② CREATE ROLE
  - ④ ERROR: permission denied for table student
  - ⑥ GRANT ROLE
  - ⑧ 結果取得できる事
-

● Q21 権限削除

【問題】

- ① superuser01でtest02 DBに入り直し、  
user01の権限のselectのみ排除。
- ② user01 で入って、Studentテーブルに対してselectを実施。
- ③ user01 でstudentテーブルのid1 を4 に更新
- ④ user01 でstudentテーブルのname を全て ABC に変更

【期待出力】

- ① REVOKE
- ② ERROR: permission denied for table student
- ③ ERROR: permission denied for table student
- ④ UPDATE 3

\* 補足あり。問題回答後に参照推奨。

---

● Q22 所有者変更

【問題】

- ① user01 で test02\_02 DB の名前をtest02\_03に変更
  - ② superuser01でtest02に入り直し、test02\_02の所有者をuser01にする
  - ③ test02 にuser01で入り直し、再度test02\_02をtest02\_03に変更
- [advance] ④ superuser01で、user01に足りない設定を追加
- ⑤ 再度user01でtest02 に入り、test02\_02テーブルをtest02\_03に変更
  - ⑥ superuser01で入り直し、test02\_03 を test02\_02 に変更

【期待出力】

- ① ERROR: must be owner of database test02\_02
  - ② ALTER DATABASE
  - ③ ERROR: permission denied to rename database
- ④ ヒントに記載
- ⑤⑥ ALTER DATABASE
-

### ● Q23 ロール削除

【問題】

- ① postgresでtest02テーブルに入り直し、superuser01を削除
- ② 続けて、user01を削除
- ③ user01を消すのに必要な対処を実施
- ④ user01削除
- ⑤ user01に参加していたuser02が消えているか確認する。  
user02を削除する。(残っていなければエラーになるはず)

【期待出力】

- ① DROP ROLE
  - ② ERROR: role "user01" cannot be dropped because some  
objects depend on it  
DETAIL: privileges for table student  
owner of database test02\_02
  - ③ ヒント欄に記載
  - ④ DROP ROLE
  - ⑤ DROP ROLE ※ 参加していたroleが消されても、role自体は残る事がわかる
- 

### ● Q24 ビューと組み合わせたロールの活用

【問題】

- ① id(int), name(text), status(text)を持つpersonテーブルを作成する。  
また上記テーブルに以下の情報を登録する。  
(1, "AAA", 'Active'), (2, "BBB", 'Ended'), (3, "CCC", 'Active')
- [advance] ② personテーブルのstatusはActiveが契約している状態、Endedは契約完了を想定する。  
(契約中のステータスはActive以外にもありえるものとする)  
契約中の人のIDのみが表示されるビューを作成する  
(ビュー名は「ope\_view」とし、唯一みれるIDの列名は「active\_id」とする)
- ③ 上記の限られた情報のみをオペレーション部門は使う事とする。  
オペレーション部門のロールを作成(opeとする)  
上記ビューを確認できるようにする。(権限はselectのみとする)  
なお上記ロールはグループのように使う為、ログインとパスワード属性を付与しない。
- ④ opeに参加するユーザとなるロール(ope001)を作成する。(ログイン・パスワード属性を付与する)  
作成後、opeに参加させる。
- ⑤ ope001でDBにつなぎなおし、personテーブルの確認と、ope\_viewの確認を行う。

【期待出力】

- ② CREATE VIEW
- ③④ ヒント欄に記載
- ⑤ 繋ぎなおしたうえの情報取得結果は以下  
personテーブル: ERROR: permission denied for table person  
ope\_viewビュー(右青枠)

| active_id |
|-----------|
| 1         |
| 3         |

\* 補足あり。問題回答後に参照推奨。

---

● Q25 ロールに参加したロールの解除

【問題】

- ① test02にpostgresで繋ぎ直す
- ② ope001ロールを、opeロールから取り除く
- ③ 必要事項を実施し、opeとope001をまとめて削除する  
(drop文は1文で実施する)

【期待出力】

- ② REVOKE ROLE  
※ WARNING: role "ope" is not a member of role "ope001"  
と表示されない事
- ③ DROP ROLE

## 3章 TCL問題

---

### ■ 事前準備

### 事前準備

練習問題実行前に以下を実行する事。

```
psql -U postgres -d postgres
create database test03;
\q

psql -U postgres -d test03
create table student(id int , name text);
insert into student(id, name) values (1, 'AAA'), (2, 'BBB'), (3, 'CCC');

create schema test_schema;
```

トランザクション関連ではターミナルを2つ使用するケースがある。  
必要に応じて起動すること。

---

### ● Q26 トランザクション(正常)

【問題】シンプルな例でトランザクションの実行を試してみる

- ① 端末A: studentテーブルをselectで取得
- ② 端末A:トランザクション開始
- ③ 端末A: updateでstudentテーブルのidが1のnameをZZZに変更
- ④ 端末B: studentテーブルをselect
- ⑤ 端末A:コミット
- ⑥ 端末B: studentテーブルをselect

【期待出力】

- ① 結果が出力される。その値をメモしておく事。
- ② BEGIN
- ③ UPDATE 1
- ④ 端末Aで更新した情報が反映されていない事
- ⑤ COMMIT
- ⑥ 端末Aで更新した情報が反映されていること

● Q27 トランザクション(失敗)

【問題】トランザクション内で失敗した場合の挙動について確認する

- ① トランザクション開始
- ② select文で失敗(誤記)
- ③ 正しいselect文でstudentテーブル確認
- ④ コミットを実行

※ ロールバック系のコマンドでも可。結果は変わらない。

【期待出力】※ 前ページと重複する点は省略

- ② ERROR
- ③ ERROR: current transaction is aborted,  
commands ignored until end of transaction block
- ④ ROLLBACK

※ 本問題はヒントはなし

---

● Q28 トランザクション(savepoint 復帰)

【問題】savepointを使った復旧を確認する

- ① トランザクション開始
- ② studentテーブルのid 2のnameを YYY に変更
- ③ セーブポイント(save01)を作成
- ④ update文で失敗
- ⑤ セーブポイント(save01)に復帰
- ⑥ studentテーブルのid 3 のnameを XXX に変更
- ⑦ セーブポイント(save01)削除
- ⑧ コミット
- ⑨ selectでstudent確認。  
id 2と3 の nameが両方とも変わっていること

【期待出力】

- ③ SAVEPOINT
  - ⑤ ROLLBACK
  - ⑦ RELEASE
-

● Q29 何もない所でトランザクションコマンドを打つと？

【問題】トランザクションをかけていない時にトランザクション系コマンドを打つとどうなるか確認する

- ① トランザクションID確認
- ② トランザクションID確認
- ③ コミット実施
- ④ トランザクションID確認
- ⑤ ロールバック
- ⑥ セーブポイント取得
- ⑦ セーブポイントへの復帰
- ⑧ セーブポイント削除

【期待出力】

- ① IDが払い出される(数値をメモっておく)
  - ② IDが1進む。(取得するたびに1進む)
  - ③ WARNING: there is no transaction in progress  
COMMIT
  - ④ IDが②の時から1進んでいていること(空撃ちだとID消費しない)
  - ⑤ WARNING: there is no transaction in progress  
ROLLBACK
  - ⑥ ERROR: SAVEPOINT can only be used in transaction blocks
  - ⑦ ERROR: ROLLBACK TO SAVEPOINT can only be used in transaction blocks
  - ⑧ ERROR: RELEASE SAVEPOINT can only be used in transaction blocks
- 

● Q30 トランザクションID確認

【問題】トランザクションIDの払い出され方を確認する

- ① トランザクションID確認
- ② updateでstudentテーブルのid 1 の name を AAA に変更
- ③ トランザクションID確認
- ④ トランザクションを開始、studentテーブルのid が 1 の  
name を ABC に変更し、コミット
- ⑤ トランザクションID確認

【期待出力】

- ① IDが払い出される(数値をメモっておく)
- ③ IDが2進む。  
(updateで1進み、取得するたびに1進むので合計2進む)
- ⑤ IDが2進む。  
(明示的トランザクションで1進み、取得で1進むので、  
合計2進む)

\* 本問題はヒントなし

---

● Q31 トランザクション(ロック待ち)

【問題】トランザクションを実施するとselectは待たされないが、同じ行を更新すると待たされることを確認する

- ① 端末A: トランザクション開始
- ② 端末A: studentテーブルのid1 の name を AAA に変更
- ③ **端末B:** トランザクション開始
- ④ **端末B:** selectでstudent確認
- ⑤ **端末B:** id 2 の name を BBB に変更
- ⑥ **端末B:** id 1 の name を CCC に変更
- ⑦ 端末A:コミット
- ⑧ **端末B:**コミット。その後、selectでstudent確認

【期待出力】

- ④ 問題なく結果が見れること(ただし、id1はAAAではない(古い情報))
- ⑤ 即座に完了(待たされない)
- ⑥ 端末Aがコミットするまで待たされる(トランザクションで行ロックがかかる)
- ⑦ 端末Aがコミット完了後、待たされていた端末Bの処理が即座に完了
- ⑧ id 2は BBB、id 1 は CCCである事

\* 本問題はヒントなし

---

● Q32 明示的テーブルロック

【問題】テーブルロックをするとselect含めて待たされることを確認する

- ① 端末A: トランザクション開始。開始時に分離レベルをserializableにする
- ② 端末A: studentに対し、明示的テーブルロック
- ③ **端末B:** トランザクション開始。開始時に分離レベルをserializableにする
- ④ **端末B:** selectでstudent確認
- ⑤ 端末A: id が 3 の name を DDD に変更
- ⑥ 端末A: コミット、端末Bは結果が返ってきたらトランザクション処理終了

【期待出力】

- ② LOCK TABLE
  - ③ BEGIN
  - ④ selectでも待たされる
  - ⑥ 端末Aのコミットと同時に、待たされていた端末Bのselect結果が返る  
結果はDDDになる**前の**値。
-

● Q33 明示的行ロック(共有ロック①)

\* 待ち中にログが表示され、SQLを打ちづらくなったらエンターで改行すると良い。以降も同様。

【問題】緩めの行ロックである共有ロックの動作を確認する

<パターン1:select, updateの動作確認>

- ① 端末A:トランザクション開始
- ② 端末A: studentのid 1 に対して共有ロック取得
- ③ 端末B:トランザクション開始
- ④ 端末B: studentのid 1 に対してselect で情報取得
- ⑤ 端末B: studentのid 2 に対してnameを'XYZ'に変更
- ⑥ 端末B: studentのid 1 の name を'UVW'に変更
- ⑦ 端末A: ロールバック(その後、端末Bもロールバック)

【期待出力】

- ② selectと同じ結果が表示される
- ④ 問題なく表示できる  
(共有ロックされたレコードを別トランザクションから参照できる)
- ⑤ 問題なく更新できる(同じテーブルでも異なるレコードなら更新可)
- ⑥ 待たされる
- ⑦ 端末Aがロールバックした瞬間に端末Bのupdateが成功する

---

● Q34 明示的行ロック(共有ロック②)

【問題】緩めの行ロックである共有ロックの動作を確認する

<パターン2:select for update/shareの動作確認>

- ① 端末A: トランザクション開始
- ② 端末A: studentのid 1 に対して共有ロック取得
- ③ 端末B: トランザクション開始
- ④ 端末B: studentのid 1 に対して排他ロック取得
- ⑤ 端末B: ロックをキャンセルし、ロールバック。再度トランザクション開始
- ⑥ 端末B: studentのid 1 に対して共有ロック取得
- ⑦ 端末A: studentのid 1 のnameを'ACE'に変更
- ⑧ 端末B: ロールバック後、端末Aもロールバック

【期待出力】

- ② selectと同じ結果が出力
- ④ 待たされる
- ⑤ キャンセル時に以下出力  
Cancel request sent  
ERROR: canceling statement due to user request  
CONTEXT: while locking tuple (0,23) in relation "student"
- ⑥ 成功する(selectの結果出力と同じ)
- ⑦ 待たされる(先に共有ロックしても、他トランザクションが共有ロックすると更新等はできなくなる)
- ⑧ 端末Bでロールバック後、端末A側で即座に更新がかかる

### ● Q35 明示的行ロック(排他ロック)

【問題】共有ロックより厳しい排他ロックの動作を確認する

- ① 端末A: トランザクション開始
- ② 端末A: student の id 1 に排他ロック取得
- ③ **端末B:** トランザクション開始
- ④ **端末B:** select で id 1 のレコード取得
- ⑤ **端末B:** student の id 1 に共有ロック取得
- ⑥ **端末B:** 処理キャンセルしてロールバックして再度トランザクション開始。
- ⑦ **端末B:** student の id 1 に排他ロック取得
- ⑧ 端末A でロールバック後、端末B でロールバック

【期待出力】

- ② 通常のselect の結果表示
- ④ 問題なく表示される(排他ロックでもselectは確認できる)
- ⑤ 待たされる(排他ロックがかかっている時は、後発で共有ロックもとれない)
- ⑦ 待たされる(排他ロックがかかっている時は、後発で排他ロックもとれない)

\* 本問題はヒントなし

---

### ● Q36 デッドロック

【問題】

- ① 端末A: トランザクション開始
- ② 端末A: student の id 1 の name を '111' に変更
- ③ **端末B:** トランザクション開始
- ④ **端末B:** student の id 2 の name を 'aaa' に変更
- ⑤ **端末B:** student の id 1 の name を '111' に変更
- ⑥ 端末A: student の id 2 の name を 'aaa' に変更
- ⑦ 端末A と 端末B 両方で student の全行取得
- ⑧ 両者ロールバック

【期待出力】

```
⑥ デッドロック発生 (デッドロックの原因となるupdate文を先に打った方が更新される)
ERROR: deadlock detected
DETAIL: Process 39176 waits for ShareLock on transaction 994; blocked by process 29784.
Process 29784 waits for ShareLock on transaction 995; blocked by process 39176.
HINT: See server log for query details.
CONTEXT: while updating tuple (0,24) in relation "student"
⑦ 端末Aは失敗、Bは成功(デッドロックでエラーが出た方が処理失敗扱い)
```

\* 本問題はヒントなし

---

● Q37 トランザクション分離レベル(ダーティリード未発生・反復不能読み取り 発生)

【問題】

- ① 端末A、Bともにトランザクション開始( read uncommitted に変更)
- ② 端末B:トランザクション分離レベルの確認
- ③ 端末A: student の id 1 の name を 'nonrepeat' に変更
- ④ 端末B: select で student のレコード取得
- ⑤ 端末A: コミット
- ⑥ 端末B: select で student のレコード取得、その後ロールバック

【期待出力】

② transaction\_isolation

-----  
read uncommitted  
(1 行)

- ④ まだ id 1 が name が前の値のままである事  
(未コミットの更新情報は見えない: ダーティリードは発生していない)
- ⑥ id 1 の name が 'nonrepeat' になる事  
(別トランザクションのコミット結果が、トランザクション内で見えてしまう)

---

● Q38 トランザクション分離レベル(反復不能読み取り 改善)

【問題】

- ① 端末A、Bともにトランザクション開始( serializable に変更)
- ② 端末B: トランザクション分離レベルの確認
- ③ 端末A: student の id 1 の name を 'clear' に変更
- ④ 端末B: select で student のレコードを確認
- ⑤ 端末A: コミット
- ⑥ 端末B: select で student のレコード取得、その後ロールバック

【期待出力】

② transaction\_isolation

-----  
serializable  
(1 行)

- ④ id 1 は nonrepeat のまま
- ⑥ id 1 は nonrepeat のまま(反復不能読み取りが発生しない)

\* 本問題はヒントなし

● Q39 トランザクション分離レベル(ファントムリード 発生)

【問題】

- ① 端末A、Bともにトランザクション開始
- ② 端末B:トランザクション分離レベルの確認
- ③ 端末A: studentで id 4 name 'phantom' でレコード追加
- ④ 端末B:selectでstudentのレコード取得
- ⑤ 端末A:コミット
- ⑥ 端末B:selectでstudentのレコード取得、その後ロールバック

【期待出力】

- ② トランザクション開始のデフォルトはread commitである事  
transaction\_isolation
- 

-----  
read committed  
(1 行)

- ④ 追加したレコードはこの時点では見えない事
- ⑥ 追加したレコードが見える事

\* 本問題はヒントなし

---

● Q40 トランザクション分離レベル(ファントムリード 改善)

【問題】

- ① 端末A、Bともにトランザクション開始
- ② 端末A、Bともにrepeatable read に変更
- ③ 端末B:トランザクション分離レベルの確認
- ④ 端末A: studentで id 4 レコードを削除
- ⑤ 端末B: selectでstudentのレコード取得
- ⑥ 端末A:コミット
- ⑦ 端末B: selectでstudentのレコード取得、その後ロールバック

【期待出力】

- ③ transaction\_isolation
- 

repeatable read

- ⑤ deleteされたレコードが見えていること
  - ⑦ deleteされたレコードが見えていること(postgresのrepeatable readはファントムリードが発生しない為、動作改善がされている)
-

● Q41 トランザクション分離レベル(直列異常 改善)

※ これは改善の方が動きがわかりやすい為、先に改善から実施

【問題】

- ① 端末A、Bともにトランザクション開始(serializable)
- ② 端末Aで student の idを合計した値を nameに入れる(idは11とする)処理を実施
- ③ 端末Bで studentの idを合計した値を nameに入れる(idは21とする)処理を実施
- ④ 端末Aをコミット
- ⑤ 端末Bをコミット
- ⑥ 端末Bで studentを確認

【期待出力】

- ⑤ 直列異常が発生するから、commit失敗、というログができる
  - (②と③でstudentテーブルのid合計値をnameに入れた新たなレコードを追加している。
  - ②で新たなレコードが追加された為、③は②で追加されたレコード分のidも含めた集計をしたい(端末A commit → 端末B commit のように直列でトランザクションをした場合はそうなる)が、実際はダーティリードが発生しない為、
  - ③の時に②で追加した値が見えていない。  
その為、直列でトランザクションをかけた際と結果が異なってしまう為、エラーとする)

ERROR: could not serialize access due to read/write dependencies among transactions

DETAIL: Reason code: Canceled on identification as a pivot, during commit attempt.

HINT: The transaction might succeed if retried.

⑥ 端末A側で登録した情報(id 11)のものだけ登録されていること。

---

● Q42 トランザクション分離レベル(直列異常 発生)

【問題】

- ① 端末A、Bともにトランザクション開始  
( repeatable read に変更)
- ② 端末A で student の idを合計した値を  
nameに入れる(id は12 とする)処理を実施
- ③ 端末B で studentの idを合計した値を  
nameに入れる( idは22とする)処理を実施
- ④ 端末Aをコミット
- ⑤ 端末Bをコミット。  
その後、selectでstudentテーブルを全行取得。

【期待出力】

- ⑤ 特にエラーなくコミットできる。  
Id 12も22もname欄は同じ数値となっている。

\* 本問題はヒントなし

### ● Q43 SET 文の影響範囲の理解

【問題】

- ① SET文を使用し、begin 後にトランザクション分離レベルをserializeとして、コミット。  
その後、トランザクション分離レベルを確認。
- ② SET文を使用し、beginせずに、トランザクション分離レベルをserializeに変更。  
その後、トランザクション分離レベルを確認。
- ③ DDL章でも実施したが、スキーマ参照パスの変更をSET文で実施(test\_schemaに変更)した後に  
show文で変わったことを確認する。  
その後、一度psqlの接続を切って、再度つなぎ直した後に、参照スキーマの確認をshow文で行う。

【期待出力】

- ①トランザクション完了後は元の値に戻る  
transaction\_isolation

---

```

read committed
```

- ②トランザクションを開始していないと使えないというログが出る  
WARNING: SET TRANSACTION can only be used in transaction blocks  
SET

また、トランザクション分離レベルの確認では以前のままである(serializableに切り替わらない)  
transaction\_isolation

---

```

read committed
```

- ③ show文を打った後は参照スキーマがtest\_schemaに変わっていること。  
接続し直したは元の値に戻っている事。

※ ②や③の結果から、SET文はpsqlで繋いでいる間(セッション)やトランザクション内等の一部でしか設定を  
切り替えられない

\* 本問題はヒントなし

---

### ● Q44 トランザクション分離レベルの永続的変更(1)

【問題】

- ① test03 DBのスキーマ参照パスを永続的に変更する。  
※ postgresql.confの変更ではなく、SQLで変更する方法で実施すること。  
変更後にshowで確認。  
確認後、psqlで繋ぎ直して再確認。  
[advance] ② test03 のトランザクション分離レベルを永続的にserializableに変更する。  
変更後、psqlでtest03につなぎ直して、再確認。  
変更後、psqlでtest03ではなく、postgres DBにつなぎ直して  
showで確認を行う  
[advance] ③ test03 のトランザクション分離レベルをデフォルトに戻す。  
変更後、psqlでtest03につなぎ直して、再確認。

【期待出力】

- ① 設定変更コマンドの期待出力はヒントに記載  
変更直後のshowは切り替わっていない。  
つなぎ直した際が切り替わる。
  - ② ①と出力は同じ  
psqlで繋ぎ直した後はtransaction isolation levelが変更されている事  
postgres DBにつないだ後のshowは変更前の値である事。
  - ③ 設定変更コマンドの期待出力はヒントに記載。  
つなぎ直した後に元の値(read committed)に戻っていること
-

● Q45 トランザクション分離レベルの永続的変更(2)

【問題】

[advance] ① 前問題とは異なる方法で永続的に  
トランザクション分離レベルをrepeatable readに変える。  
※ postgresql.confではなく、SQLで実施すること。  
変更後、psqlでtest03につなぎ直して、再確認。

【期待出力】

① 設定変更コマンドの期待出力はヒントに記載。  
psql接続後も値は変化しない。

※ 補足あり。問題回答後、参照推奨。

---

■ 後片付け

## 後片付け

```
alter system set default_transaction_isolation =
'repeatable read';
を実行した方は以下コマンドを実施ください。
```

```
alter system reset default_transaction_isolation ;
```

## 4章 DML(基礎)問題

---

### ■ 事前準備

#### 事前準備

以下のSQLをpostgres DB , postgresユーザで繋いで実行する。

```
create database test04 ;
create table dept (dept_id int, name text);
create table users (id int, dept_id int, name text, grade numeric);
create table device (device_id int primary key, user_id int, status boolean, create_at timestamp default current_timestamp);
insert into dept values ('1', 'AX'), ('2', 'BX'), ('3', 'CX');
insert into users values ('1', 'AAA', 3.5), ('2', 'BBB', 2.5), ('3', 'CDef', 3.2), ('4', 'cdEF', 4.5);
insert into device values ('1', 1, true);
```

以下のような表が出来上がったイメージ。  
社内管理システムを想定し、deptは部署名、usersはユーザ、deviceはユーザが使用する端末を指す  
deviceのstatusは使用有無で、trueは使用中、falseは使用停止中とする。

【deptテーブル】

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       | CX   |

【usersテーブル】

| id | dept_id | name | grade |
|----|---------|------|-------|
| 1  | 1       | AAA  | 3.5   |
| 2  | 1       | BBB  | 2.5   |
| 3  | 2       | CDef | 3.2   |
| 4  | 3       | cdEF | 4.5   |

【deviceテーブル】

| device_id | user_id | status | create_at   |
|-----------|---------|--------|-------------|
| 1         | 1       | t      | [timestamp] |

#### ● Q46 シンプルな select(事前準備が正しくできているか確認)

##### 【問題】

事前準備で作成した3つの表(dept, users, device)を全行取得。

※なるべく何も見ずに回答してみましょう

##### 【期待出力】

[dept]

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       | CX   |

[users]

| id | dept_id | name | grade |
|----|---------|------|-------|
| 1  | 1       | AAA  | 3.5   |
| 2  | 1       | BBB  | 2.5   |
| 3  | 2       | CCC  | 3.2   |
| 4  | 3       | DDD  | 4.5   |

[device]

| device_id | user_id | status | create_at                  |
|-----------|---------|--------|----------------------------|
| 1         | 1       | t      | 2022-10-05 08:52:05.757679 |

● Q47 一部のレコード削除

【問題】

deptから、dept\_idが2のレコードを削除。  
ただし削除したレコード情報が返ってくるようにすること。

【期待出力】

| dept_id     |  | name |
|-------------|--|------|
| -----+----- |  |      |
| 2           |  | BX   |

---

● Q48 delete以外によるデータ削除

【問題】

deptを全行削除する。  
その後、selectですべて消えていることを確認する。  
その際にはdelete以外の句を使う事。

【期待出力】

[select結果]

| dept_id     |  | name |
|-------------|--|------|
| -----+----- |  |      |
| (0行)        |  |      |

● Q49 データ投入

【問題】

- ① device に device\_id 2 , user\_id 2 , status null でデータ投入。  
(create\_atは指定省略)  
ただし投入結果が結果の全てが返ってくるようにする事。
- ② dept に dept\_id 1～2 , name AX,BX を1文で追加。  
ただし投入した結果のnameだけが返ってくるようにする事。
- ③ insert ~ select を使用し、deptにdept\_id 3～4 , name AX1 BX2 で  
投入されるようにする。  
ただし投入した結果の全てが返ってくるようにすること。

【期待出力】

| [①] | device_id   user_id   status   create_at      |
|-----|-----------------------------------------------|
|     | -----<br>2   2     2022-10-11 09:39:28.746173 |

| [②] | name              | [③] | dept_id   name              |
|-----|-------------------|-----|-----------------------------|
|     | -----<br>AX<br>BX |     | -----<br>3   AX1<br>4   BX2 |

● Q50 データ更新

【問題】

AX1をAXBU1に、BX1をBXBU2に更新するSQL文を実行。  
ただし、1回のみで実行するものとする。  
また、更新したレコードの情報がすべて返ってくる指定を行う事。

【期待出力】

| dept_id   name     |
|--------------------|
| -----<br>4   BXBU2 |
| 3   AXBU1          |

### ● Q51 csvを使ったデータ運用

#### 【問題】

- ① deviceテーブルの情報をcsvファイルで「/tmp/export.csv」として取得する。
- ② psql接続を切り、/tmp/export.csvを/tmp/import.csvとしてコピーした上で、編集する(OSコマンド)。

[例: 以下を1行ずつ実施]  
 cp /tmp/export.csv /tmp/import.csv  
 vi /tmp/import.csv

編集により、以下の情報を追記する(参考までにviの使い方を次ページに記載。)

```
3,4,f,2022-10-12 08:52:05.757679
4,4,t,2022-10-13 09:39:28.746173
```

- ③ 作成した /tmp/import.csv をdeviceテーブルにインポートする
- ④ deviceテーブルのレコードを全削除したのち、③と同じコマンドを再度実施

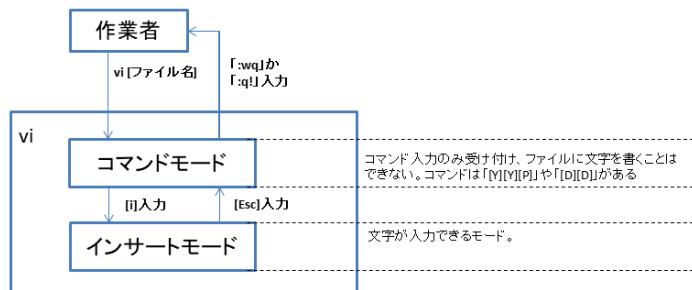
#### 【期待出力】

- ① COPY 2
- ③ ERROR: duplicate key value violates unique constraint "device\_pkey"  
 DETAIL: Key (device\_id)=(1) already exists.  
 CONTEXT: COPY device, line 1
- ④ COPY 4

\* csv のファイルを Linux OS 上で作成する方法は以下

### [参考] viコマンドの使い方

- キーボードを入力しても入力できない場合は「コマンドモード」。  
 (vi [変更ファイル] を実行すると、最初はこれになっているはず)  
 →「i」を押すと、「インサートモード」となり、入力ができる。  
 どちらのモードも方向キー(矢印キー)で移動する。
- コマンドモードでは「コピーしたい行」に方向キーで移動し、「Y」「Y」「P」を押せば、  
 行ごとコピーできる。  
 行を削りたい場合は「D」「D」を押せば消せる。  
 また、vi自体を終わらせるのもこのモードからとなる。  
 「:wq」を押すと、「保存して終了」  
 「:q!」で、「保存せずに終了」。必要な動きをしていたら「:q!」で抜けてしまうのが良い。
- インサートモードを終了してコマンドモードに戻るには  
 「Esc」ボタンを押す。



koppeするデータは以下。(日付は好みで変更可、1日程度ずらしておくと良い)

```
3,4,f,2022-10-12 08:52:05.757679
4,4,t,2022-10-13 09:39:28.746173
```

■ ここまでテーブル情報

[参考] ここまでにできたテーブル情報

- 以下のようなデータが出来上がっているはず。

【deptテーブル】

| dept_id | name  |
|---------|-------|
| 1       | AX    |
| 2       | BX    |
| 3       | AXBU1 |
| 4       | BXBU2 |

【usersテーブル】

| id | dept_id | name | grade |
|----|---------|------|-------|
| 1  | 1       | AAA  | 3.5   |
| 2  | 1       | BBB  | 2.5   |
| 3  | 2       | CDef | 3.2   |
| 4  | 3       | cDEF | 4.5   |

【deviceテーブル】

| device_id | user_id | status | create_at   |
|-----------|---------|--------|-------------|
| 1         | 1       | t      | [timestamp] |
| 2         | 2       |        | [timestamp] |
| 3         | 4       | f      | [timestamp] |
| 4         | 4       | t      | [timestamp] |

● Q52 複数条件の指定方法

【問題】

- ① usersテーブルから[AX]所属(=deptでidが1のもの)で評価が3を超える人を算出  
(join等使わず、シンプルにdept\_idが1のものを指定する。  
以降も同様。)
- ② usersテーブルから[AX]所属か、[BX]所属の人を算出
- ③ usersテーブルでAXBU1以外の人を算出
- ④ usersテーブルから[AX]所属か[BX]所属で、  
かつ3以上の評価の人を算出

【期待出力】

| 【①】 |         |      |       |
|-----|---------|------|-------|
| id  | dept_id | name | grade |
| 1   | 1       | AAA  | 3.5   |

| 【②③】 |         |      |       |
|------|---------|------|-------|
| id   | dept_id | name | grade |
| 1    | 1       | AAA  | 3.5   |
| 2    | 1       | BBB  | 2.5   |
| 3    | 2       | CDef | 3.2   |

| 【④】 |         |      |       |
|-----|---------|------|-------|
| id  | dept_id | name | grade |
| 1   | 1       | AAA  | 3.5   |
| 3   | 2       | CDef | 3.2   |

● Q53 算術関数の使用

【問題】

- ① usersテーブルにて各メンバの評価(grade)を四捨五入して表示。
- ② usersテーブルにて各メンバの評価を四捨五入した場合と  
小数点切り捨てをした場合の差分を表示。

【期待出力】

【①】

| id | dept_id | name | grade | round |
|----|---------|------|-------|-------|
| 1  | 1       | AAA  | 3.5   | 4     |
| 2  | 1       | BBB  | 2.5   | 3     |
| 3  | 2       | CDef | 3.2   | 3     |
| 4  | 3       | cdEF | 4.5   | 5     |

【②】

| id | dept_id | name | grade | ?column? |
|----|---------|------|-------|----------|
| 1  | 1       | AAA  | 3.5   | 1        |
| 2  | 1       | BBB  | 2.5   | 1        |
| 3  | 2       | CDef | 3.2   | 0        |
| 4  | 3       | cdEF | 4.5   | 1        |

● Q54 カラム名の指定

【問題】

- ① usersテーブルにて各メンバの評価を四捨五入した場合と  
小数点切り捨てをした場合の差分を表示。  
その際にカラム名を「check」とすること。
- ② usersのgradeを「評価」というタイトルにしてほしい。

【期待出力】

【①】

| id | dept_id | name | grade | check |
|----|---------|------|-------|-------|
| 1  | 1       | AAA  | 3.5   | 1     |
| 2  | 1       | BBB  | 2.5   | 1     |
| 3  | 2       | CDef | 3.2   | 0     |
| 4  | 3       | cdEF | 4.5   | 1     |

【②】

| id | dept_id | name | 評価  |
|----|---------|------|-----|
| 1  | 1       | AAA  | 3.5 |
| 2  | 1       | BBB  | 2.5 |
| 3  | 2       | CDef | 3.2 |
| 4  | 3       | cdEF | 4.5 |

● Q55 集約関数の使用

【問題】

① usersテーブルの全ユーザの評価の平均点を出力

[advance] ② usersテーブルの全ユーザの評価の平均点を出す。

ただし 少数第2位までとし、  
以下は四捨五入する。

③ usersテーブルの[AX]部署の評価平均点を出力

【期待出力】

|                   |
|-------------------|
| 【①】               |
| avg               |
| -----             |
| 3.425000000000000 |
| (1 行)             |

|       |
|-------|
| 【②】   |
| round |
| ----- |
| 3.43  |

|                   |
|-------------------|
| 【③】               |
| avg               |
| -----             |
| 3.000000000000000 |

● Q56 集約関数とグルーピング

【問題】

usersテーブルの評価を部署ごとに表示する。

その際、部署idも表示すること。

【期待出力】

|                       |
|-----------------------|
| 【①】                   |
| dept_id   avg         |
| -----+-----           |
| 3   4.500000000000000 |
| 2   3.200000000000000 |
| 1   3.000000000000000 |

● Q57 グルーピングと条件式

【問題】

- ① usersテーブルにて、AXBU1を除いて評価を部署ごとに表示する。
- ② usersテーブルで全部署対象に評価平均点を出して、3を超えるもののみ表示する。
- ③ usrsテーブルにて、AXBU1を除いて評価を部署ごとに表示、その結果3を超えるもののみ表示する。

【期待出力】

| 【①】     |                    |
|---------|--------------------|
| dept_id | avg                |
| 2       | 3.2000000000000000 |
| 1       | 3.0000000000000000 |

| 【②】     |                    |
|---------|--------------------|
| dept_id | avg                |
| 3       | 4.5000000000000000 |
| 2       | 3.2000000000000000 |

| 【③】     |                    |
|---------|--------------------|
| dept_id | avg                |
| 2       | 3.2000000000000000 |

● Q58 該当レコード数の確認

【問題】

- ① deviceテーブルにて全デバイスの数を取得。
- ② deviceテーブルにてstatusが入っていないものを除いて数を集計。

【期待出力】

| 【①】   |  |
|-------|--|
| count |  |
| 4     |  |
| (1 行) |  |

| 【②】   |  |
|-------|--|
| count |  |
| 3     |  |
| (1 行) |  |

● Q59 null の取得

【問題】

- ① deviceテーブルにてstatusがnullであるレコードを取得
- ② deviceテーブルにてstatusがnullであるレコード以外を取得

【期待出力】

【①】

| device_id | user_id | status | create_at                  |
|-----------|---------|--------|----------------------------|
| 2         | 2       |        | 2022-10-05 09:39:28.746173 |

【②】

| device_id | user_id | status | create_at                  |
|-----------|---------|--------|----------------------------|
| 1         | 1       | t      | 2022-10-05 08:52:05.757679 |
| 3         | 4       | f      | 2022-10-12 08:52:05.757679 |
| 4         | 4       | t      | 2022-10-13 09:39:28.746173 |

● Q60 演算子の優先度

【問題】

- ① 評価3.5と2.5の成績を足した後に2で累乗して、最後に2で割る事。  
(whereで条件を指定してAAAとBBBの必要情報をとらず、  
そのまま数値計算で良いこととする。from句・where句を  
使用せずに計算する)
- ② ①のSQLで(カッコ)を外して計算結果が変わることを確認。

【期待出力】

【①】

|                    |
|--------------------|
| ?column?           |
| -----              |
| 18.000000000000000 |

【②】

|                   |
|-------------------|
| ?column?          |
| -----             |
| 6.625000000000000 |

● Q61 時刻演算子

【問題】

- ① deviceテーブルの 最大の時刻と最小の時刻を引いて、  
登録時間の差を確認
- ② deviceテーブルの device\_id 1の登録日時から15h前の  
時間を算出

【期待出力】

※ ①も②も登録日時により結果時間が変わるので注意。  
参考として資料作成時のmax(id 4)とmin(id 1)の結果を載せておく。

【①】  
?column?  
-----  
8 days 00:47:22.988494

【②】  
?column?  
-----  
2022-10-04 17:52:05.757679

[参考] 試験環境でのmaxとmin

| max                        |   | min                        |
|----------------------------|---|----------------------------|
| -----                      | + | -----                      |
| 2022-10-13 09:39:28.746173 |   | 2022-10-05 08:52:05.757679 |

● Q62 時刻関数

【問題】

- ① 現在時刻からdeviceテーブルのcreate\_atの最小小時刻をひいて、  
差分を確認する
- ② deviceテーブルのcreate\_atから、年情報だけ取得する。

【期待出力】

※ 出力結果は現時刻により変わるので出力例はあくまで参考程度。

【①】  
?column?  
-----  
20:10:29.489153

【②】  
????  
-----  
2023  
2023  
2023  
2023

### ● Q63 時刻関数の取得タイミング

#### 【問題】

- ① 取得タイミングがトランザクション開始時点、という意味を理解する。
  1. トランザクション開始
  2. 時刻取得タイミングがトランザクション開始時点の関数を実行
  3. その後、少し待って、2と同じものを再度打つ。(ロールバックせずそのまま②へ)
- ② トランザクション開始時点と、SQL開始時点の違いを理解する
  1. 時刻取得タイミングがSQL開始時点の関数を実行
  2. その後、少し待って、1と同じものを実行
  3. ロールバック  
※ 関数実行時点でも結果は同様の為、省略
- ③ SQL開始時点と、関数実行時点の違いについて理解する
  1. 明示的トランザクションをかけずに、以下を実行 (意味は次のページで説明)
 

```
select now(); select statement_timestamp() from (select pg_sleep(10)) a;
```
  2. 明示的トランザクションをかけずに、以下を実行
 

```
select now(); select clock_timestamp() from (select pg_sleep(10)) a;
```

#### 【期待出力】

| 【①-2】<br>now                           | 【②-2】<br>statement_timestamp           | 【③-1】<br>now                           | 【③-2】<br>now                           |
|----------------------------------------|----------------------------------------|----------------------------------------|----------------------------------------|
| -----<br>2022-10-06 10:02:35.246832+09 | -----<br>2022-10-06 10:03:29.7506+09   | -----<br>2022-10-06 10:58:58.383808+09 | -----<br>2022-10-06 10:58:15.863168+09 |
| 【①-3】<br>now                           | 【②-3】<br>statement_timestamp           | 【③-1】<br>statement_timestamp           | 【③-2】<br>clock_timestamp               |
| -----<br>2022-10-06 10:02:35.246832+09 | -----<br>2022-10-06 10:03:47.342998+09 | -----<br>2022-10-06 10:58:58.416459+09 | -----<br>2022-10-06 10:58:25.901124+09 |

トランザクション内だと  
時間置いでも  
結果変わらない

トランザクション内でも  
結果変わる  
(トランザクション開始時点と差)

SQL開始時点だと結果は  
ほとんど変わらない

関数実行時点だと、約10秒  
差が出る(SQLを実行し始めてから  
clock\_timestampが呼び出される  
までに10秒程度かかっている為)

③のSQLは以下からコピペ可。

```
select now() ; select statement_timestamp() from (select pg_sleep(10)) a ;
select now() ; select clock_timestamp() from (select pg_sleep(10)) a ;
```

\* 補足あり。回答後、確認推奨。

● Q64 データ型書式設定関数

【問題】

deviceテーブルのcreate\_at から YYYYMMDD形式で情報を取得

【期待出力】

【①】

to\_char

-----

20221005

20221005

20221012

20221013

● Q65 文字関数

【問題】

- ① usersテーブルのAAAを小文字で表示
- ② usersテーブルの情報を使って、「user名:dept\_id」という形式で表示。
- ③ usersテーブルのdept\_idを3桁の番号(例:001)で表示

【期待出力】

【①】

???

-----

aaa

【②】

user名:dept\_id

-----

AAA:1

BBB:1

CDef:2

cdEF:3

【③】

???

-----

001

001

002

003

● Q66 generate\_series

【問題】

- ① 1～10の連続した数値を取得する。
  - ② 本日から1週間前から、1週間後の2週間の日付を取得する。
- [advance] ③ 192.168.10.1から始まるIPアドレス10個を、2個ずつ  
取得する

【期待出力】

① generate\_series

-----

1

2

3

(省略)

(10 rows)

② generate\_series

※ 1週間前と、後の  
両方が入っている事

-----

2022-09-29

2022-09-30

2022-10-01

(省略)

(14 rows)

③ ?column?

-----

192.168.10.1

192.168.10.3

192.168.10.5

192.168.10.7

192.168.10.9

(5 rows)

● Q67 文字検索(like, ilike)

【問題】

- ① usersテーブルにてnameの前方一致で  
CDef のみヒットするように文字検索を実行  
(大文字・小文字を区別する)
- ② usersテーブルにてnameの 中間一致で  
Cdef、cdEF がヒットするように文字検索を実行  
(大文字・小文字を区別しない)

【期待出力】

【①】

id | dept\_id | name | grade

---+-----+-----+

3 | 2 | CDef | 3.2

【②】

id | dept\_id | name | grade

---+-----+-----+

3 | 2 | CDef | 3.2

4 | 3 | cdEF | 4.5

● Q68 文字検索(similar to)

【問題】

- ① usersテーブルにて、nameを後方一致でAAAと、CDefがヒットする文字検索を実施

【期待出力】

【①】

| id | dept_id | name | grade |
|----|---------|------|-------|
| 1  | 1       | AAA  | 3.5   |
| 3  | 2       | CDef | 3.2   |

● Q69 文字検索(正規表現)

【問題】

- ① usersテーブルにて、正規表現・後方一致でnameがcdEFのみヒットする検索を実行。  
② usersテーブルにて、正規表現・前方一致でnameがAAA以外ヒットする検索を実行。

【期待出力】

【①】

| id | dept_id | name | grade |
|----|---------|------|-------|
| 4  | 3       | cdEF | 4.5   |

【②】

| id | dept_id | name | grade |
|----|---------|------|-------|
| 2  | 1       | BBB  | 2.5   |
| 3  | 2       | CDef | 3.2   |
| 4  | 3       | cdEF | 4.5   |

● Q70 重複排除

【問題】

- ① usersテーブルで所属部署(dept\_id)が重複しない出力を実施。  
(並び替え不要)
- ② usersテーブルで所属部署が重複しない出力を実施。  
(並び替え要。所属部署のみ表示)
- ③ usersテーブルで所属部署が重複しない出力を実施。  
(並び替え要。評価と所属部署を表示)

【期待出力】

| 【①】 | dept_id |
|-----|---------|
|     | -----   |
| 3   |         |
| 2   |         |
| 1   |         |

| 【②】 | dept_id |
|-----|---------|
|     | -----   |
|     | 1       |
|     | 2       |
|     | 3       |

| 【③】 | grade   dept_id |
|-----|-----------------|
|     | -----+-----     |
| 3.5 | 1               |
| 3.2 | 2               |
| 4.5 | 3               |

評価での並び替え結果で表示はしない

● Q71 重複排除とヒット数取得の組み合わせ

【問題】

- ① usersテーブルで所属部署(dept\_id)が重複しないようにして  
ヒット数を取得
- ② deviceテーブルのstatusで重複しないように出力
- ③ deviceテーブルのstatusで重複しないようにヒット数を取得

【期待出力】

| 【①】 | count |
|-----|-------|
|     | ----- |
| 3   |       |

| 【②】   | status |
|-------|--------|
|       | -----  |
|       | f      |
| t     |        |
| (3 行) |        |

| 【③】 | count |
|-----|-------|
|     | ----- |
| 2   |       |

Nullが3パターン目としてあるが、nullは排除され、  
2パターン表示とされる。

● Q72 並び替え

## 【問題】

- ① usersテーブルをnameでアルファベット順に並び替え。
- ② deviceテーブルをuser\_idの数値が大きい順、同じ数値の場合はdevice\_idの若い順に並び替え。

## 【期待出力】

## 【①】

| id | dept_id | name | grade |
|----|---------|------|-------|
| 1  | 1       | AAA  | 3.5   |
| 2  | 1       | BBB  | 2.5   |
| 3  | 2       | CDef | 3.2   |
| 4  | 3       | cdEF | 4.5   |

## 【②】

| device_id | user_id | status | create_at                  |
|-----------|---------|--------|----------------------------|
| 3         | 4       | f      | 2022-10-12 08:52:05.757679 |
| 4         | 4       | t      | 2022-10-13 09:39:28.746173 |
| 2         | 2       |        | 2022-10-05 09:39:28.746173 |
| 1         | 1       | t      | 2022-10-05 08:52:05.757679 |

※ 小文字より大文字優先

● Q73 重複排除並び替えと並び替え指定、どちらが優先されるか

## 【問題】

usersテーブルで、distinct on句を使って所属部署(dept\_id)が重複しない出力を実施。  
(並び替えを実施。3問前の②ベースのSQL文)

ただし、並び替え指定(order by)句で所属idが大きい値から並ぶように指定。

## 【期待出力】

## 【①】

| dept_id |
|---------|
| 3       |
| 2       |
| 1       |

## [参考] 3問前の②ベース(order by指定なし)

| dept_id |
|---------|
| 1       |
| 2       |
| 3       |

※ distinct on より  
order byが優先される

● Q74 取得結果の範囲指定

【問題】

- ① usersテーブルの評価が3を超える人を取得。  
ただし1名のみ取得。(誰でも可)
- ② usersテーブルの評価が3を超える人を取得。  
ただし1名のみ取得。(一番点数高い人)
- ③ deviceテーブルにて登録が新しい順に並べ、  
2件目と3件目のみ取得。

【期待出力】

| 【①】 |         |      |       |
|-----|---------|------|-------|
| id  | dept_id | name | grade |
| 1   | 1       | AAA  | 3.5   |

| 【②】 |         |      |       |
|-----|---------|------|-------|
| id  | dept_id | name | grade |
| 4   | 3       | cdEF | 4.5   |

| 【③】       |         |        |                            |
|-----------|---------|--------|----------------------------|
| device_id | user_id | status | create_at                  |
| 3         | 4       | f      | 2022-10-12 08:52:05.757679 |
| 2         | 2       |        | 2022-10-05 09:39:28.746173 |

● Q75 取得範囲の指定

【問題】

deviceテーブルの device\_id が 2～4 のレコードを取得。

【期待出力】

| device_id | user_id | status | create_at                  |
|-----------|---------|--------|----------------------------|
| 2         | 2       |        | 2022-10-05 09:39:28.746173 |
| 3         | 4       | f      | 2022-10-12 08:52:05.757679 |
| 4         | 4       | t      | 2022-10-13 09:39:28.746173 |

● Q76 まとめ問題

【問題】

- ① 以下の条件を全て満たすSQL文を実行する
  - ・deviceテーブルで、ユーザごとに特定の状態の端末を何台持っているか計算する
  - ・集計条件はstatus情報が入っていない(null)のものとする
  - ・nullデバイスの数を集計した列のタイトルは「nullstatus」とする
  - ・出力はstatusがnullのデバイス数が多いユーザから表示するものとする
- ② 以下の条件を全て満たすSQL文を実行する。
  - ・users テーブルで部署(dept\_id)ごとに所属者(user\_id)の評価(grade)平均点を出す
  - ・ただし、評価が3以下の人には平均の集計対象外
  - ・また、部署idが4の人も集計対象外。
  - ・部署ごとの平均点は少数第一位まで表示。(第二位で四捨五入)
  - ・平均点を出した列のタイトルは「avg」とする。
  - ・集計した結果、3.2以上の部署を表示対象とする
  - ・結果は評価(平均点)が高い順に並べ、上位3部署を表示

【期待出力】

| 【①】     |            |
|---------|------------|
| user_id | nullstatus |
| 2       | 1          |
| 4       | 0          |
| 1       | 0          |

| 【②】     |     |
|---------|-----|
| dept_id | avg |
| 3       | 4.5 |
| 1       | 3.5 |
| 2       | 3.2 |

## 5章 DML(応用)問題

本問題はデータの更新があるのはQ89のみであり、他はほとんどselect文である。

本章の問題は複雑もある為、気になる点から確認する、でも良い。

一部を飛ばして実施する場合、Q89の前と後で検証環境の想定データが少しだけ異なる為、事前準備と、Q89後の「ここまで」をみてデータ状態をあわせてから実施する事。

---

### ■ 事前準備

```
psql -U postgres -d postgres
create database test05 ;
\q
psql -U postgres -d test05
create table dept (dept_id int , name text);
create table users (user_id int , dept_id int , name text , grade numeric);
create table device (device_id int primary key , user_id int , status boolean , create_at timestamp
default current_timestamp);
insert into dept values (1 , 'AX'),(2 , 'BX'),(3 , ''),(4 , 'AX');
insert into users values (1 , 1 , 'AAA' , 3.5),(2 , 1 , 'BBB' , 2.5),(3 , 2 , 'CDef' , 3.2),(4 , 3 , 'cdEF' , 4.5),(5 , 5 ,
'GGG' , 3.8);
insert into device values (1 , 1 , true);insert into device values (2 , 2 , null);insert into device values
(3 , 4 , false);insert into device values (4 , 4 , true); insert into device values (5 , 6 , true);
```

以下のような表が出来上がったイメージ。  
正しく出来上がっているか、selectで確認しておくこと。

【deptテーブル】

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |

【usersテーブル】

| user_id | dept_id | name | grade |
|---------|---------|------|-------|
| 1       | 1       | AAA  | 3.5   |
| 2       | 1       | BBB  | 2.5   |
| 3       | 2       | CDef | 3.2   |
| 4       | 3       | cdEF | 4.5   |
| 5       | 5       | GGG  | 3.8   |

【deviceテーブル】

| device_id | user_id | status | create_at   |
|-----------|---------|--------|-------------|
| 1         | 1       | t      | [timestamp] |
| 2         | 2       |        | [timestamp] |
| 3         | 4       | f      | [timestamp] |
| 4         | 4       | t      | [timestamp] |
| 5         | 6       | t      | [timestamp] |

● Q77 結合1

【問題】

- ① usersテーブルとdeptテーブルを結合し、ユーザと部署名が同時に見られる出力を取得。  
一致したもののみ取得する方法で結合することとする。(結合条件はonとする)
- ② usersテーブルとdeviceテーブルを結合し、ユーザとデバイス情報が同時に見れる出力を取得。  
一致したもののみ取得。(on)

|   | 【usersテーブル】 |         |      |       | 【deptテーブル】 |      | 【deviceテーブル】 |         |               |
|---|-------------|---------|------|-------|------------|------|--------------|---------|---------------|
|   | user_id     | dept_id | name | grade | dept_id    | name | device_id    | user_id | status        |
| ① | 1           | 1       | AAA  | 3.5   | 1          | AX   | 1            | 1       | t [timestamp] |
|   | 2           | 1       | BBB  | 2.5   | 2          | BX   | 2            | 2       | [timestamp]   |
|   | 3           | 2       | CDef | 3.2   | 3          |      | 3            | 4       | f [timestamp] |
|   | 4           | 3       | cdEF | 4.5   | 4          | AX   | 4            | 4       | t [timestamp] |
|   |             | 5       |      | GGG   | 3.8        |      |              | 5       | 6             |

【期待出力】

[①]

| user_id                     | dept_id | name | grade | dept_id | name |
|-----------------------------|---------|------|-------|---------|------|
| 1   1   AAA   3.5   1   AX  |         |      |       |         |      |
| 2   1   BBB   2.5   1   AX  |         |      |       |         |      |
| 3   2   CDef   3.2   2   BX |         |      |       |         |      |
| 4   3   cdEF   4.5   3      |         |      |       |         |      |

[②]

| user_id                                                     | dept_id | name | grade | device_id | user_id | status | create_at |
|-------------------------------------------------------------|---------|------|-------|-----------|---------|--------|-----------|
| 1   1   AAA   3.5   1   1   t   2022-10-05 08:52:05.757679  |         |      |       |           |         |        |           |
| 2   1   BBB   2.5   2   2     2022-10-05 09:39:28.746173    |         |      |       |           |         |        |           |
| 3   2   CDef   3.2   3   4   f   2022-10-12 08:52:05.757679 |         |      |       |           |         |        |           |
| 4   3   cdEF   4.5   4   4   t   2022-10-13 09:39:28.746173 |         |      |       |           |         |        |           |

● Q78 結合 2

【問題】

- ① 結合条件 using でusersとdeptを結合。イコール条件が成立するもののみ表示。
- ② 結合条件 using でusersとdeviceを結合。イコール条件が成立するもののみ表示。
- ③ 結合条件 natural で usersとdeviceを結合。イコール条件が成立するもののみ表示。
- ④ 結合条件 natural で usersとdeptを結合すると何もヒットせず、不都合が生じる。  
まずは上記で0件ヒットとなる事をSQLで確認する事。  
また、これはなぜか。また、どうすれば問題なく使用できるか。

【期待出力】

| [①]     |          |          |       |      |
|---------|----------|----------|-------|------|
| dept_id | user_id  | name     | grade | name |
| 1       | 1   AAA  | 3.5   AX |       |      |
| 1       | 2   BBB  | 2.5   AX |       |      |
| 2       | 3   CDef | 3.2   BX |       |      |
| 3       | 4   cdEF | 4.5      |       |      |

| [②③]    |          |      |       |                            |        |           |
|---------|----------|------|-------|----------------------------|--------|-----------|
| user_id | dept_id  | name | grade | device_id                  | status | create_at |
| 1       | 1   AAA  | 3.5  | 1   t | 2022-10-05 08:52:05.757679 |        |           |
| 2       | 1   BBB  | 2.5  | 2     | 2022-10-05 09:39:28.746173 |        |           |
| 4       | 3   cdEF | 4.5  | 3   f | 2022-10-12 08:52:05.757679 |        |           |
| 4       | 3   cdEF | 4.5  | 4   t | 2022-10-13 09:39:28.746173 |        |           |

| [④]     |      |         |       |
|---------|------|---------|-------|
| dept_id | name | user_id | grade |
| (0行)    |      |         |       |

### ● Q79 結合3

#### 【問題】

- ① usersとdeptでイコール条件に一致しないものも含めて表示できる結合を実行。  
usersにしかない情報が出るような条件で対応すること。結合条件はonとする。  
② ①で、結合条件はusingとする。

#### 【期待出力】

| ①       |         |      |       |         |
|---------|---------|------|-------|---------|
| user_id | dept_id | name | grade | dept_id |
| 1       | 1       | AAA  | 3.5   | 1       |
| 2       | 1       | BBB  | 2.5   | 1       |
| 3       | 2       | CDef | 3.2   | 2       |
| 4       | 3       | cdEF | 4.5   | 3       |
| 5       | 5       | GGG  | 3.8   |         |

#### 【usersテーブル】

| user_id | dept_id | name | grade |
|---------|---------|------|-------|
| 1       | 1       | AAA  | 3.5   |
| 2       | 1       | BBB  | 2.5   |
| 3       | 2       | CDef | 3.2   |
| 4       | 3       | cdEF | 4.5   |
| 5       | 5       | GGG  | 3.8   |

#### 【deptテーブル】

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |

#### 【②】

| ②       |         |      |       |      |
|---------|---------|------|-------|------|
| dept_id | user_id | name | grade | name |
| 1       | 1       | AAA  | 3.5   | AX   |
| 1       | 2       | BBB  | 2.5   | AX   |
| 2       | 3       | CDef | 3.2   | BX   |
| 3       | 4       | cdEF | 4.5   |      |
| 5       | 5       | GGG  | 3.8   |      |

#### 【問題】

- ③ usersとdeviceでイコール条件に一致しないものも含めて表示できる結合を実行。  
usersにしかない情報が出るような条件で対応すること。結合条件はonとする。  
④ ①で、結合条件はusingとする。

#### 【期待出力】

| usersテーブル |         |      |       |
|-----------|---------|------|-------|
| user_id   | dept_id | name | grade |
| 1         | 1       | AAA  | 3.5   |
| 2         | 1       | BBB  | 2.5   |
| 3         | 2       | CDef | 3.2   |
| 4         | 3       | cdEF | 4.5   |
| 5         | 5       | GGG  | 3.8   |

#### 【deviceテーブル】

| device_id | user_id | status | create_at   |
|-----------|---------|--------|-------------|
| 1         | 1       | t      | [timestamp] |
| 2         | 2       |        | [timestamp] |
| 3         | 4       | f      | [timestamp] |
| 4         | 4       | t      | [timestamp] |
| 5         | 6       | t      | [timestamp] |

#### 【③】

| ③       |         |      |       |           |         |                            |
|---------|---------|------|-------|-----------|---------|----------------------------|
| user_id | dept_id | name | grade | device_id | user_id | status                     |
| 1       | 1       | AAA  | 3.5   | 1         | t       | 2022-10-05 08:52:05.757679 |
| 2       | 1       | BBB  | 2.5   | 2         | t       | 2022-10-05 09:39:28.746173 |
| 3       | 2       | CDef | 3.2   |           |         |                            |
| 4       | 3       | cdEF | 4.5   | 3         | f       | 2022-10-12 08:52:05.757679 |
| 4       | 3       | cdEF | 4.5   | 4         | t       | 2022-10-13 09:39:28.746173 |
| 5       | 5       | GGG  | 3.8   |           |         |                            |

#### 【④】

| ④       |         |      |       |           |        |                            |
|---------|---------|------|-------|-----------|--------|----------------------------|
| user_id | dept_id | name | grade | device_id | status | create_at                  |
| 1       | 1       | AAA  | 3.5   | 1         | t      | 2022-10-05 08:52:05.757679 |
| 2       | 1       | BBB  | 2.5   | 2         | t      | 2022-10-05 09:39:28.746173 |
| 3       | 2       | CDef | 3.2   |           |        |                            |
| 4       | 3       | cdEF | 4.5   | 3         | f      | 2022-10-12 08:52:05.757679 |
| 4       | 3       | cdEF | 4.5   | 4         | t      | 2022-10-13 09:39:28.746173 |
| 5       | 5       | GGG  | 3.8   |           |        |                            |

### ● Q80 結合 4

#### 【問題】

- ① usersとdeptでイコール条件に一致しないものも含めて表示できる結合を実行。  
deptにしかない情報が出るような条件で対応すること。結合条件はonとする。
- ② ①で、結合条件はusingとする。

#### 【期待出力】

【usersテーブル】

| user_id | dept_id | name | grade |
|---------|---------|------|-------|
| 1       | 1       | AAA  | 3.5   |
| 2       | 1       | BBB  | 2.5   |
| 3       | 2       | CDef | 3.2   |
| 4       | 3       | cdEF | 4.5   |
| 5       | 5       | GGG  | 3.8   |

【deptテーブル】

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |

【①】

| user_id | dept_id | name | grade | dept_id | name |
|---------|---------|------|-------|---------|------|
| 1       | 1       | AAA  | 3.5   | 1       | AX   |
| 2       | 1       | BBB  | 2.5   | 1       | AX   |
| 3       | 2       | CDef | 3.2   | 2       | BX   |
| 4       | 3       | cdEF | 4.5   | 3       |      |
|         |         |      |       | 4       | AX   |

【②】

| dept_id | user_id | name | grade | name |
|---------|---------|------|-------|------|
| 1       | 1       | AAA  | 3.5   | AX   |
| 1       | 2       | BBB  | 2.5   | AX   |
| 2       | 3       | CDef | 3.2   | BX   |
| 3       | 4       | cdEF | 4.5   |      |
| 4       |         |      |       | AX   |

#### 【問題】

- ③ usersとdeviceでイコール条件に一致しないものも含めて表示できる結合を実行。  
deviceにしかない情報が出るような条件で対応すること。結合条件はonとする。
- ④ ①で、結合条件はusingとする。

#### 【期待出力】

【usersテーブル】

| user_id | dept_id | name | grade |
|---------|---------|------|-------|
| 1       | 1       | AAA  | 3.5   |
| 2       | 1       | BBB  | 2.5   |
| 3       | 2       | CDef | 3.2   |
| 4       | 3       | cdEF | 4.5   |
| 5       | 5       | GGG  | 3.8   |

【deviceテーブル】

| device_id | user_id | status | create_at   |
|-----------|---------|--------|-------------|
| 1         | 1       | t      | [timestamp] |
| 2         | 2       |        | [timestamp] |
| 3         | 4       | f      | [timestamp] |
| 4         | 4       | t      | [timestamp] |
| 5         | 6       | t      | [timestamp] |

【③】

| user_id | dept_id | name | grade | device_id | user_id | status | create_at                  |
|---------|---------|------|-------|-----------|---------|--------|----------------------------|
| 1       | 1       | AAA  | 3.5   | 1         | 1       | t      | 2022-10-05 08:52:05.757679 |
| 2       | 1       | BBB  | 2.5   | 2         | 2       |        | 2022-10-05 09:39:28.746173 |
| 4       | 3       | cdEF | 4.5   | 3         | 4       | f      | 2022-10-12 08:52:05.757679 |
| 4       | 3       | cdEF | 4.5   | 4         | 4       | t      | 2022-10-13 09:39:28.746173 |
|         |         |      |       | 5         | 6       | t      | 2022-10-06 11:12:44.217037 |

【④】

| user_id | dept_id | name | grade | device_id | status | create_at                  |
|---------|---------|------|-------|-----------|--------|----------------------------|
| 1       | 1       | AAA  | 3.5   | 1         | t      | 2022-10-05 08:52:05.757679 |
| 2       | 1       | BBB  | 2.5   | 2         |        | 2022-10-05 09:39:28.746173 |
| 4       | 3       | cdEF | 4.5   | 3         | f      | 2022-10-12 08:52:05.757679 |
| 4       | 3       | cdEF | 4.5   | 4         | t      | 2022-10-13 09:39:28.746173 |
| 6       |         |      |       | 5         | t      | 2022-10-06 11:12:44.217037 |

### ● Q81 結合5

#### 【問題】

① usersとdeptでイコール条件に一致しないものも含めて表示できる結合を実行。

users,deptそれぞれにしかない情報が出るような条件で対応すること。

結合条件はonとする。

② ①で、結合条件はusingとする。

#### 【期待出力】

| 【①】     |         |      |       |         |
|---------|---------|------|-------|---------|
| user_id | dept_id | name | grade | dept_id |
| 1       | 1       | AAA  | 3.5   | 1   AX  |
| 2       | 1       | BBB  | 2.5   | 1   AX  |
| 3       | 2       | CDef | 3.2   | 2   BX  |
| 4       | 3       | cdEF | 4.5   | 3       |
|         |         |      |       | 4   AX  |
| 5       | 5       | GGG  | 3.8   |         |

| 【②】     |         |      |       |      |
|---------|---------|------|-------|------|
| dept_id | user_id | name | grade | name |
| 1       | 1       | AAA  | 3.5   | AX   |
| 1       | 2       | BBB  | 2.5   | AX   |
| 2       | 3       | CDef | 3.2   | BX   |
| 3       | 4       | cdEF | 4.5   |      |
| 4       |         |      |       | AX   |
| 5       | 5       | GGG  | 3.8   |      |

| 【usersテーブル】 |         |      |       | 【deptテーブル】 |      |
|-------------|---------|------|-------|------------|------|
| user_id     | dept_id | name | grade | dept_id    | name |
| 1           | 1       | AAA  | 3.5   | 1          | AX   |
| 2           | 1       | BBB  | 2.5   | 2          | BX   |
| 3           | 2       | CDef | 3.2   | 3          |      |
| 4           | 3       | cdEF | 4.5   | 4          | AX   |
| 5           | 5       | GGG  | 3.8   |            |      |

#### 【問題】

③ usersとdeviceでイコール条件に一致しないものも含めて表示できる結合を実行。

users,deviceそれぞれにしかない情報が出るような条件で対応すること。

結合条件はonとする。

④ ①で、結合条件はusingとする。

#### 【期待出力】

| 【userテーブル】 |         |      |       | 【deviceテーブル】 |         |        |             |
|------------|---------|------|-------|--------------|---------|--------|-------------|
| user_id    | dept_id | name | grade | device_id    | user_id | status | create_at   |
| 1          | 1       | AAA  | 3.5   | 1            | 1       | t      | [timestamp] |
| 2          | 1       | BBB  | 2.5   | 2            | 2       |        | [timestamp] |
| 3          | 2       | CDef | 3.2   |              |         |        |             |
| 4          | 3       | cdEF | 4.5   | 3            | 4       | f      | [timestamp] |
|            |         |      |       | 4            | 4       | t      | [timestamp] |
| 5          | 5       | GGG  | 3.8   | 5            | 6       | t      | [timestamp] |

| 【③】     |         |      |       |           |         |        |                            |
|---------|---------|------|-------|-----------|---------|--------|----------------------------|
| user_id | dept_id | name | grade | device_id | user_id | status | create_at                  |
| 1       | 1       | AAA  | 3.5   | 1         | 1       | t      | 2022-10-05 08:52:05.757679 |
| 2       | 1       | BBB  | 2.5   | 2         | 2       |        | 2022-10-05 09:39:28.746173 |
| 3       | 2       | CDef | 3.2   |           |         |        |                            |
| 4       | 3       | cdEF | 4.5   | 3         | 4       | f      | 2022-10-12 08:52:05.757679 |
| 4       | 3       | cdEF | 4.5   | 4         | 4       | t      | 2022-10-13 09:39:28.746173 |
| 5       | 5       | GGG  | 3.8   | 5         | 6       | t      | 2022-10-06 11:12:44.217037 |

| 【④】     |         |      |       |           |        |                            |  |
|---------|---------|------|-------|-----------|--------|----------------------------|--|
| user_id | dept_id | name | grade | device_id | status | create_at                  |  |
| 1       | 1       | AAA  | 3.5   | 1         | t      | 2022-10-05 08:52:05.757679 |  |
| 2       | 1       | BBB  | 2.5   | 2         |        | 2022-10-05 09:39:28.746173 |  |
| 3       | 2       | CDef | 3.2   |           |        |                            |  |
| 4       | 3       | cdEF | 4.5   | 3         | f      | 2022-10-12 08:52:05.757679 |  |
| 4       | 3       | cdEF | 4.5   | 4         | t      | 2022-10-13 09:39:28.746173 |  |
| 5       | 5       | GGG  | 3.8   | 5         | t      | 2022-10-06 11:12:44.217037 |  |
|         |         |      |       | 5         | t      | 2022-10-06 11:12:44.217037 |  |

### ● Q82 結合 6

【問題】

- ① usersテーブルとdeptテーブルで結合した結果、  
usersテーブルにしか存在しない情報を取得する。(結合条件on)
- ② ①を結合条件usingで試してみる。
- ③ usersテーブルとdeptテーブルで結合した結果、  
deptテーブルにしか存在しない情報を取得する。(結合条件on)

【期待出力】

| ①                                                                                                        |
|----------------------------------------------------------------------------------------------------------|
| user_id   dept_id   name   grade   dept_id   name<br>-----+-----+-----+-----+-----+<br>5   5   GGG   3.8 |

| ②                                                                                        |
|------------------------------------------------------------------------------------------|
| dept_id   user_id   name   grade   name<br>-----+-----+-----+-----+<br>5   5   GGG   3.8 |

| ①                                                                                                     |
|-------------------------------------------------------------------------------------------------------|
| user_id   dept_id   name   grade   dept_id   name<br>-----+-----+-----+-----+-----+<br>        4   AX |

【usersテーブル】

| id | dept_id | name | grade |
|----|---------|------|-------|
| 1  | 1       | AAA  | 3.5   |
| 2  | 1       | BBB  | 2.5   |
| 3  | 2       | CDef | 3.2   |
| 4  | 3       | cdEF | 4.5   |
| 5  | 5       | GGG  | 3.8   |

【deptテーブル】

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |

緑の箇所が結合した結果。  
usersテーブルにしか存在しないと  
わかった情報。  
③で取得する。

左と同様にここ  
がdeptにしか  
存在しない情報。  
③で取得する。

### ● Q83 結合 7

【問題】

- ① usersテーブルとdeptテーブルを結合した結果、それぞれのテーブルにしか  
存在しない情報をのみを表示させる。(結合条件on)
- ② ①をusingを使って取得する

【期待出力】

| ①                                                                                                                          |
|----------------------------------------------------------------------------------------------------------------------------|
| user_id   dept_id   name   grade   dept_id   name<br>-----+-----+-----+-----+-----+<br>        4   AX<br>5   5   GGG   3.8 |

【usersテーブル】

| id | dept_id | name | grade |
|----|---------|------|-------|
| 1  | 1       | AAA  | 3.5   |
| 2  | 1       | BBB  | 2.5   |
| 3  | 2       | CDef | 3.2   |
| 4  | 3       | cdEF | 4.5   |
| 5  | 5       | GGG  | 3.8   |

【deptテーブル】

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |

| ②                                                                                                        |
|----------------------------------------------------------------------------------------------------------|
| dept_id   user_id   name   grade   name<br>-----+-----+-----+-----+<br>4         AX<br>5   5   GGG   3.8 |

### ● Q84 結合 8

#### 【問題】

- ① usersとdeviceで、全組み合わせ取得の結合を実施。  
(cross joinという文言をSQLに入れる事。全取得。)
- ② ①と同じ書き方(cross join)で、結果的にinner joinの結果と、同じになるSQL文を実行
- ③ ①でcross joinという文言を入れないケース。
- ④ ③で結果的にinner joinと同じ結果になるSQL文の実行。

#### 【期待出力】

【①】 【③】

| user_id | dept_id | name | grade | device_id | user_id | status | create_at                  |
|---------|---------|------|-------|-----------|---------|--------|----------------------------|
| 1       | 1       | AAA  | 3.5   | 1         | 1       | t      | 2022-10-05 08:52:05.757679 |
| 2       | 1       | BBB  | 2.5   | 1         | 1       | t      | 2022-10-05 08:52:05.757679 |
| 3       | 2       | CDef | 3.2   | 1         | 1       | t      | 2022-10-05 08:52:05.757679 |
| 4       | 3       | cdEF | 4.5   | 1         | 1       | t      | 2022-10-05 08:52:05.757679 |
| ...     |         |      |       |           |         |        |                            |
| 5       | 5       | GGG  | 3.8   | 5         | 6       | t      | 2022-10-06 11:12:44.217037 |

(25 rows)

【②】 【④】

| user_id | dept_id | name | grade | device_id | user_id | status | create_at                  |
|---------|---------|------|-------|-----------|---------|--------|----------------------------|
| 1       | 1       | AAA  | 3.5   | 1         | 1       | t      | 2022-10-05 08:52:05.757679 |
| 2       | 1       | BBB  | 2.5   | 2         | 2       |        | 2022-10-05 09:39:28.746173 |
| 4       | 3       | cdEF | 4.5   | 3         | 4       | f      | 2022-10-12 08:52:05.757679 |
| 4       | 3       | cdEF | 4.5   | 4         | 4       | t      | 2022-10-13 09:39:28.746173 |

(4 rows)

### ● Q85 3つのテーブル結合

#### 【問題】

[advance] dept、users、deviceを繋げて各情報を一括で見る(inner, onを結合条件とする)

#### 【期待出力】

| 【deptテーブル】 |      | 【usersテーブル】 |         |      |       | 【deviceテーブル】 |         |        |             |
|------------|------|-------------|---------|------|-------|--------------|---------|--------|-------------|
| dept_id    | name | user_id     | dept_id | name | grade | device_id    | user_id | status | create_at   |
| 1          | AX   | 1           | 1       | AAA  | 3.5   | 1            | 1       | t      | [timestamp] |
| 2          | BX   | 2           | 1       | BBB  | 2.5   | 2            | 2       |        | [timestamp] |
| 3          |      | 3           | 2       | CDef | 3.2   |              |         |        |             |
| 4          | AX   | 4           | 3       | cdEF | 4.5   | 3            | 4       | f      | [timestamp] |
|            |      | 5           | 5       | GGG  | 3.8   | 4            | 4       | t      | [timestamp] |
|            |      |             |         |      |       | 5            | 6       | t      | [timestamp] |

【①】

| dept_id | name         | user_id                                      | dept_id | name | grade | device_id | user_id | status | create_at |
|---------|--------------|----------------------------------------------|---------|------|-------|-----------|---------|--------|-----------|
| 1   AX  | 1   1   AAA  | 3.5   1   1   t   2022-10-05 08:52:05.757679 |         |      |       |           |         |        |           |
| 1   AX  | 2   1   BBB  | 2.5   2   2     2022-10-05 09:39:28.746173   |         |      |       |           |         |        |           |
| 3       | 4   3   CDef | 3.2   3   4   f   2022-10-12 08:52:05.757679 |         |      |       |           |         |        |           |
| 3       | 4   3   cdEF | 4.5   4   4   t   2022-10-13 09:39:28.746173 |         |      |       |           |         |        |           |

(4 行)

### ● Q86 3つのテーブル結合時の取得範囲指定 1

**【問題】**

[advance] ①3つのテーブルを結合した上で、deptにしかない情報を取得する。  
(結合条件on, outer joinのみ使って実現する)

[advance] ②①をusingを使って取得する。

**【期待出力】**

【deptテーブル】

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |

【usersテーブル】

| user_id | dept_id | name | grade |
|---------|---------|------|-------|
| 1       | 1       | AAA  | 3.5   |
| 2       | 1       | BBB  | 2.5   |
| 3       | 2       | CDef | 3.2   |
| 4       | 3       | cDEF | 4.5   |
| 5       | 5       | GGG  | 3.8   |

【deviceテーブル】

| device_id | user_id | status | create_at   |
|-----------|---------|--------|-------------|
| 1         | 1       | t      | [timestamp] |
| 2         | 2       |        | [timestamp] |
| 3         | 4       | f      | [timestamp] |
| 4         | 4       | t      | [timestamp] |
| 5         | 6       | t      | [timestamp] |

deptにしかない  
情報  
(これを取得する)

|   |                                                                                              |
|---|----------------------------------------------------------------------------------------------|
| ① | dept_id   name   user_id   dept_id   name   grade   device_id   user_id   status   create_at |
|   | -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+                                 |
| ② | user_id   dept_id   name   name   grade   device_id   status   create_at                     |
|   | -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+                                 |
|   | 4   AX                                                                                       |

\* 補足あり。回答後に参照推奨。なお、本問題はヒントはなし。

### ● Q87 3つのテーブル結合時の取得範囲指定 2

**【問題】**

[advance] ①3つのテーブルを結合した上で、usersにしかない情報を取得する。  
(結合条件on, outer joinのみ使って実現)

[advance] ②3つのテーブルを結合した上で、deviceにしかない情報を取得する。  
(結合条件on, inner joinとouter joinを組み合わせて実現)

**【期待出力】**

【deptテーブル】

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |

【usersテーブル】

| user_id | dept_id | name | grade |
|---------|---------|------|-------|
| 1       | 1       | AAA  | 3.5   |
| 2       | 1       | BBB  | 2.5   |
| 3       | 2       | CDef | 3.2   |
| 4       | 3       | cDEF | 4.5   |
| 5       | 5       | GGG  | 3.8   |

【deviceテーブル】

| device_id | user_id | status | create_at   |
|-----------|---------|--------|-------------|
| 1         | 1       | t      | [timestamp] |
| 2         | 2       |        | [timestamp] |
| 3         | 4       | f      | [timestamp] |
| 4         | 4       | t      | [timestamp] |
| 5         | 6       | t      | [timestamp] |

|   |                                                                                              |
|---|----------------------------------------------------------------------------------------------|
| ① | dept_id   name   user_id   dept_id   name   grade   device_id   user_id   status   create_at |
|   | -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+                                 |
| ② | dept_id   name   user_id   dept_id   name   grade   device_id   user_id   status   create_at |
|   | -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+                                 |
|   | 1   AX     5   GGG   3.8                                                                     |

\* 補足あり。回答後に参照推奨。なお、本問題はヒントはなし。

### ● Q88 3つのテーブル結合時の取得範囲指定 3

## 【問題】

[advance] ① 3つのテーブルを結合した上で、各テーブルにしか存在しない情報を取得する。

## 【期待出力】

| 【deptテーブル】 |      | 【usersテーブル】 |         |      |       | 【deviceテーブル】 |         |        |             |
|------------|------|-------------|---------|------|-------|--------------|---------|--------|-------------|
| dept_id    | name | user_id     | dept_id | name | grade | device_id    | user_id | status | create_at   |
| 1          | AX   | 1           | 1       | AAA  | 3.5   | 1            | 1       | t      | [timestamp] |
| 2          | BX   | 2           | 1       | BBB  | 2.5   | 2            | 2       |        | [timestamp] |
| 3          |      | 3           | 2       | CDef | 3.2   |              |         |        |             |
|            |      | 4           | 3       | cdEF | 4.5   | 3            | 4       | f      | [timestamp] |
|            |      |             |         |      |       | 4            | 4       | t      | [timestamp] |
|            |      | 5           | 5       | GGG  | 3.8   | 5            | 6       | t      | [timestamp] |

| dept_id | name | user_id | dept_id | name | grade | device_id | user_id | status | create_at                  |
|---------|------|---------|---------|------|-------|-----------|---------|--------|----------------------------|
| 4       | AX   | 5       | 5       | GGG  | 3.8   |           |         |        | 2022-10-06 04:29:38.261622 |

\* 補足あり。回答後に参照推奨。なお、本問題はヒントはなし。

### ● Q89 select 結果で update に近い処理

## 【問題】

[advance] ① usersテーブルのuser\_id 3のnameをdeptテーブルにある、自身の所属している部署名に更新する

(更新対象(users.dept\_id)と更新元データ(dept.dept\_id)がイコール条件である場合結合を実施すること)

[advance] ② usersテーブルのuser\_id 5のname(ユーザ名)を deptテーブルの dept\_id 4 のname(部署名)に更新。

(更新対象(users.dept\_id)と更新元データ(dept.dept\_id)がイコール条件でない場合)

## 【期待出力】

| ①[参考] 更新後のusersテーブル |         |      |       | ②[参考] 更新後のusersテーブル |         |      |       | [参考] deptとusersの結合結果(inner) |         |      |       |
|---------------------|---------|------|-------|---------------------|---------|------|-------|-----------------------------|---------|------|-------|
| user_id             | dept_id | name | grade | user_id             | dept_id | name | grade | user_id                     | dept_id | name | grade |
| 1                   | 1       | AAA  | 3.5   | 1                   | 1       | AAA  | 3.5   | 1                           | AX      |      |       |
| 2                   | 1       | BBB  | 2.5   | 2                   | 1       | BBB  | 2.5   | 2                           | AX      |      |       |
| 3                   | 2       | BX   | 3.2   | 3                   | 2       | BX   | 3.2   | 3                           | BX      |      |       |
| 4                   | 3       | cdEF | 4.5   | 4                   | 3       | cdEF | 4.5   | 4                           |         |      |       |
| 5                   | 5       | GGG  | 3.8   | 5                   | 5       | AX   | 3.8   |                             |         |      |       |

user\_id 3はdeptとの結合でイコールとなるものがあるが、user\_id 5はイコールとなるものがない

■ ここまでテーブル状況

## [参考] ここまでテーブル状況

【deptテーブル】

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |

【deviceテーブル】

| device_id | user_id | status | create_at   |
|-----------|---------|--------|-------------|
| 1         | 1       | t      | [timestamp] |
| 2         | 2       |        | [timestamp] |
| 3         | 4       | f      | [timestamp] |
| 4         | 4       | t      | [timestamp] |
| 5         | 6       | t      | [timestamp] |

【usersテーブル】

| user_id | dept_id | name | grade |
|---------|---------|------|-------|
| 1       | 1       | AAA  | 3.5   |
| 2       | 1       | BBB  | 2.5   |
| 3       | 2       | BX   | 3.2   |
| 4       | 3       | cdEF | 4.5   |
| 5       | 5       | AX   | 3.8   |

● Q90 副問合せの基本

【問題】

- ① usersテーブルにて評価が3.5以下のユーザが何人いるかグループごとに集計し、該当ユーザが2人以上いる部署のidを取得する  
(select ~ from (select~) を使用する)
- ② ①と同じだが、部署idでなく、部署名がわかるように取得する  
結合が必要な場合はイコール条件が成立するもののみ表示とする  
(結合条件は任意で良い。回答はonを使った場合で記載する)

【期待出力】

【①】  
dept\_id | count  
-----+-----  
1 | 2

【②】  
dept\_id | count | dept\_id | name  
-----+-----+-----+-----  
1 | 2 | 1 | AX

● Q91 3つの副問合せ

【問題】

deviceテーブルにて端末を1つしか持っておらず、かつそういったユーザらが各部署に何人いるか集計し、概要ユーザが2名以上いる部署名を表示する。  
(select ~ from ( select ~ )を使用する)

結合が必要な場合はイコール条件が成立するもののみ表示とする。  
(結合条件は任意で良い。回答はonを使った場合で記載する)

【期待出力】

| dept_id | count | dept_id | name |
|---------|-------|---------|------|
| 1       | 2     | 1       | AX   |

\* 補足あり。回答後、確認推奨。

---

● Q92 副問合せ同士のjoin

【問題】

usersテーブルから全ユーザを対象に評価が3以上のユーザを取得  
(この結果をaとする)、  
deviceテーブルから全端末を対象にユーザごとに何端末持っているか集計  
(この結果をbとする)、  
aとbの結果を結合し、3以上のユーザが何端末持っているか確認する。

結合はイコール条件が成立しないものも出力する。  
(aにしかない情報、bにしかない情報を両方出力する)  
(結合条件は任意で良い。回答はonを使った場合で記載)

【期待出力】

| user_id | dept_id | name | grade | user_id | count |
|---------|---------|------|-------|---------|-------|
| 1       | 1       | AAA  | 3.5   | 1       | 1     |
| 3       | 2       | BX   | 3.2   |         |       |
| 4       | 3       | cdEF | 4.5   | 4       | 2     |
| 5       | 5       | AX   | 3.8   |         |       |
|         |         |      |       | 2       | 1     |
|         |         |      |       | 6       | 1     |

● Q93 副問合せを条件に指定

【問題】

gradeが最も低い人の所属部署の名前を、結合せずに取得する。  
(where句以下の条件で副問合せを使用して取得する)

なお、同じ評価で複数行返ってこないように必ず1件しか返ってこないようにすること。

【期待出力】

| dept_id   name |
|----------------|
| -----+-----    |
| 1   AX         |

(1 row)

● Q94 from より前の副問合せ

【問題】

- ① deptテーブルのid 3の部署名は本当はCXだが、諸事情によりDB書き込みができないとする。  
そんな中、上記に所属するcdEF氏に関するレポートが必要となった。  
select文・副問合せを活用し、「CX, cdEF, cdEFが持つデバイス1台」  
(dept\_name, name, device\_id)を表示したい。
- ② ユーザー覧を氏名をマスク(隠す)した状態で出力したい。  
user\_idは正しい値を出し、nameは一律「mask」としたい。(副問合せは使用しない)
- ③ ユーザー覧を氏名をマスク(隠す)した状態で出力したい。  
user\_idは正しい値を出し、nameはdeptテーブルのidが最も大きい部署の名前でmaskする。  
(副問合せを使用する)

【期待出力】

| ① dept_name   name   device_id | ② user_id   name | ② user_id   name |
|--------------------------------|------------------|------------------|
| -----+-----                    | -----+-----      | -----+-----      |
| CX   cdEF   3                  | 1   mask         | 1   AX           |
|                                | 2   mask         | 2   AX           |
|                                | 4   mask         | 4   AX           |
|                                | 3   mask         | 3   AX           |
|                                | 5   mask         | 5   AX           |

※ 副問合せを使うからname列が同じになるわけではなく  
副問合せでなくても同じ動きとなる事がわかる(標準動作)

### ● Q95 相関副問合せ

**【問題】**

- ① 相関副問合せを使用して、usersテーブルにて、dept\_idごとに最も高い値を取得する。
- ② 相関副問合せを使用して、usersテーブルにて、ユーザー一覧と、deptごとの最高得点を横に並べて表示する。
- ③ 相関副問合せを使用して、usersテーブルにてユーザー一覧と、そのユーザが持つデバイス数を取得する。

**【期待出力】**

| ①       |         |      |       |
|---------|---------|------|-------|
| user_id | dept_id | name | grade |
| 1       | 1       | AAA  | 3.5   |
| 4       | 3       | cdEF | 4.5   |
| 3       | 2       | BX   | 3.2   |
| 5       | 5       | AX   | 3.8   |

| ②       |      |       |     |
|---------|------|-------|-----|
| user_id | name | grade | max |
| 1       | AAA  | 3.5   | 3.5 |
| 2       | BBB  | 2.5   | 3.5 |
| 4       | cdEF | 4.5   | 4.5 |
| 3       | BX   | 3.2   | 3.2 |
| 5       | AX   | 3.8   | 3.8 |

| ③       |      |       |
|---------|------|-------|
| user_id | name | count |
| 1       | AAA  | 1     |
| 2       | BBB  | 1     |
| 4       | cdEF | 2     |
| 3       | BX   | 0     |
| 5       | AX   | 0     |

### ● Q96 シンプルな in の使い方

**【問題】**

usersテーブルから、nameがAX or BXのレコードを取得する。  
(inを使う事)

**【期待出力】**

| user_id | dept_id | name | grade |
|---------|---------|------|-------|
| 3       | 2       | BX   | 3.2   |
| 5       | 5       | AX   | 3.8   |

● Q97 複数のレコードを条件とした副問合せ(in)

【問題】

- ① usersテーブルにて評価が3.5以下のユーザが所属する  
部署名だけdeptテーブルから抜き出す(inを使う)
- ② ①から部署idが2以上のものを取得する。
- ③ ①の条件で合致しない部署名を取得する。

【期待出力】

| ① dept_id   name |
|------------------|
| -----+-----      |
| 1   AX           |
| 2   BX           |

| ② dept_id   name |
|------------------|
| -----+-----      |
| 2   BX           |

| ③ dept_id   name |
|------------------|
| -----+-----      |
| 3                |
| 4   AX           |

※ inによる副問合せを使うと  
where以降の条件で複数のものを指定でき、  
かつ簡単に追加条件を増やせることが  
わかる

● Q98 複数のレコードを条件とした副問合せ(some/any)

【問題】

- ① usersテーブルにて評価が3.5以下のユーザが所属する  
部署名だけdeptテーブルから抜き出す  
(someを使う事。)
- ② ①と同じだが、anyを使う事。  
[advance] ③ usersテーブルにて評価が3.5以下のユーザが所属する  
部署のidより上のidの部署情報をdeptから取得する。

【期待出力】

| 【①】【②】 dept_id   name |
|-----------------------|
| -----+-----           |
| 1   AX                |
| 2   BX                |

| 【③】 dept_id   name |
|--------------------|
| -----+-----        |
| 1   AX             |
| 2   BX             |

※ 補足あり。回答後、確認推奨。

● Q99 some/any と対となる動作をする指定

【問題】

[advance] usersテーブルにて評価が3.5以下のユーザが所属する  
部署のidより上のidの部署情報をdeptから取得する。  
(some/anyを使わない、max関数を使わない)

※ 本問題は前問題のsome/anyの補足を確認した後に実施する事推奨。

some/anyは「条件のどれかに一致する」もの取得する形だったが、  
「全てに一致した」もの取得するものもある。

【期待出力】

| dept_id |  | name |
|---------|--|------|
| 1       |  | AX   |
| 2       |  | BX   |

※ 補足あり。回答後、確認推奨。

---

● Q100 シンプルな exists による副問合せ

【問題】

- ① usersにuser\_idが100のユーザがいるか確認する。
- ② usersにuser\_idが1のユーザがいればdeptの一覧を取得する。
- ③ usersにuser\_idが1のユーザがない場合、deptの一覧を取得する。  
(いる場合はdeptを取得しない)

【期待出力】

|                 |
|-----------------|
| ① fなので<br>存在しない |
| exists          |

② user\_id 1がいるので、  
deptを取得する

| dept_id |  | name |
|---------|--|------|
| 1       |  | AX   |
| 2       |  | BX   |
| 3       |  |      |
| 4       |  | AX   |

③ user\_id 1がいるので  
deptを取得しない

| dept_id |  | name |
|---------|--|------|
|         |  |      |

● Q101 exists と相関副問合せの組み合わせ

**【問題】**

usersテーブルにて評価が3.5以下のユーザが所属する  
部署名だけdeptテーブルから抜き出す。(existsを使う)

**【期待出力】**

| dept_id   name |
|----------------|
| -----+-----    |
| 1   AX         |
| 2   BX         |

● Q102 with 句の使用

**【問題】**

- ①usersテーブルにて評価が3.5以下のユーザを取得し、  
そのユーザがいる部署名だけ表示する。(withを使い、Inやexistsと同じ表示出力とすること)
  - ②①と同じだが、出力を絞らず可能な限り出力すること。  
(同じ部署に該当ユーザが複数いる場合、表示する)
  - ③usersテーブルから全ユーザを対象に評価が3以上のユーザを取得(この結果をaとする)、  
deviceテーブルから全端末を対象にユーザごとに何端末持っているか集計  
(この結果をbとする)、  
aとbの結果を結合し、3以上のユーザが何端末持っているか確認する。  
結合はイコール条件が成立しないものも出力する。  
(aにしかない情報、bにしかない情報を両方出力する)  
(結合条件は任意で良い。回答はonを使った場合で記載)
- ※「副問合せ同士のjoin」と同じ出力をwithで実施

**【期待出力】**

| [①]<br>dept_id   name           | [②]<br>user_id   dept_id   name   grade   dept_id   name                                                               | [③]<br>user_id   dept_id   name   grade   user_id   count                                                                                                                            |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -----+-----<br>1   AX<br>2   BX | -----+-----+-----+-----+-----<br>2   1   BBB   2.5   1   AX<br>1   1   AAA   3.5   1   AX<br>3   2   BX   3.2   2   BX | -----+-----+-----+-----+-----+-----<br>1   1   AAA   3.5   1   1<br>3   2   BX   3.2   1  <br>4   3   cdEF   4.5   4   2<br>5   5   AX   3.8   1  <br>        2   1<br>        6   1 |

● Q103 2つのクエリ取得結果の結合

【問題】

① deptテーブルの総取得結果と、usersのdept\_id, name の総取得をまとめて確認したい。

1つの結果表に、上記2つをまとめて表示。

(重複は許容するものとする。dept\_idで数値が若い順に並び替える事)

② ①と同じだが、重複排除とする。

【期待出力】

【①】  
dept\_id | name

| dept_id | name |
|---------|------|
| 1       | AAA  |
| 1       | AX   |
| 1       | BBB  |
| 2       | BX   |
| 2       | BX   |
| 3       |      |
| 3       | cdEF |
| 4       | AX   |
| 5       | AX   |

【②】  
dept\_id | name

| dept_id | name   |
|---------|--------|
| 1       | AAA    |
| 1       | AX     |
| 1       | BBB    |
| 2       | BX     |
| 3       |        |
| 3       | cdEF   |
| 4       | AX     |
| 5       | AX(8行) |

● Q104 重複行の取得

【問題】

deptの情報と、usersのdept\_idとnameが重複するかチェックしたい。

(重複行があれば取得したい)

【期待出力】

dept\_id | name

| dept_id | name |
|---------|------|
| 2       | BX   |

(1行)

● Q105 2つのselectの差分を取得

【問題】

usersのdept\_id、nameとdeptを比較し、usersテーブルのdept\_id、nameにdeptと全く同じ登録がされているものを除いて結果を表示する。

【期待出力】

| ①       |      |
|---------|------|
| dept_id | name |
| 1       | AAA  |
| 1       | BBB  |
| 3       | cdEF |
| 5       | AX   |
| (4行)    |      |

● Q106 重複排除した行の取得

【問題】

[advenve] deviceテーブルにてuser\_idに重複なし指定をし、デバイスを持っているユーザー一覧を取得した。ところが今度は複数台デバイスを持っているユーザが気になり始めた。重複排除の対象で消された行だけを取得したい。

【期待出力】

| user_id |
|---------|
| -----   |
| 4       |
| (1行)    |

● Q107 再帰的問い合わせ

【問題】

- ① 試験用のシーケンス(with\_seq)を作成(特に細かい指定は不要)
- ② 10回連続で上記シーケンスの値を取得する。(1SQL文で)  
取得したい情報は loop\_count(ループ回数),  
seq\_value(シーケンスの値),loop\_x10(ループ数 × 10) の  
3つとする

【期待出力】

| [②]        |           |          |
|------------|-----------|----------|
| loop_count | seq_value | loop_x10 |
| 1          | 1         | 10       |
| 2          | 2         | 20       |
| 3          | 3         | 30       |
| 4          | 4         | 40       |
| 5          | 5         | 50       |
| 6          | 6         | 60       |
| 7          | 7         | 70       |
| 8          | 8         | 80       |
| 9          | 9         | 90       |
| 10         | 10        | 100      |

● Q108 クエリ実施の詳細確認

【問題】

- ① usersの user\_id 4 のレコードを取得するselect文の計画を確認する。(実行はしない)
- ② ①と同じ条件で、実行もするパターンで確認。

【期待出力】

[①]  
QUERY PLAN

---

```
Seq Scan on users (cost=0.00..20.13 rows=4 width=72)
 Filter: (user_id = 4)
(2 行)
```

[②]  
QUERY PLAN

---

```
Seq Scan on users (cost=0.00..20.13 rows=4 width=72) (actual time=0.018..0.019 rows=1 loops=1)
 Filter: (user_id = 4)
 Rows Removed by Filter: 4
Planning Time: 0.046 ms
Execution Time: 0.028 ms
(5 行)
```

### ● Q109 Window 関数 1

## 【問題】

- ① usersテーブルで部署ごとに評価の平均を取得
  - ② ①の結果のランク付けを実施。全体で順位をつける。  
同着を許容し、次順位は同着+1の順位とする。  
なお、avgは小数第二位で切り上げて表示とする。

【期待出力】

## ● Q110 Window 関数 2

## 【問題】

usersテーブルで部署ごとに評価の平均を取得し、その結果で、1位と各部署との点数差分を表示する。

## 【期待出力】

| user_id | dept_id | name | grade | round | 差分                  |
|---------|---------|------|-------|-------|---------------------|
| 1       | 1       | AAA  | 3.5   | 3.0   | 0.0000000000000000  |
| 2       | 1       | BBB  | 2.5   | 3.0   | 0.0000000000000000  |
| 3       | 2       | BX   | 3.2   | 3.2   | -0.2000000000000000 |
| 5       | 5       | AX   | 3.8   | 3.8   | -0.8000000000000000 |
| 4       | 3       | cdEF | 4.5   | 4.5   | -1.5000000000000000 |

### ● Q111 UPSERT1

## 【問題】

- ① deptテーブルを確認。
- ② dept\_id 99のレコードがdeptテーブルにない場合は以下情報を登録したいが、ない場合はそのままとしたい。  
( 99, 'SP' )
- ③ エラー解消する対処を実施する。
- ④ 再度②と同じ処理を実施。
- ⑤ deptテーブルを確認。
- ⑥ dept\_id 99のレコードがdeptテーブルにない場合は以下情報を登録したいが、ない場合はそのままとしたい。  
( 99, 'SP2' )
- ⑦ deptテーブルを確認。

## 【期待出力】

- ①⑦ 以下の青枠参照
- ② ERROR: there is no unique or exclusion constraint matching the ON CONFLICT specification
- ③ ALTER TABLE
- ④ INSERT 0 1
- ⑤ INSERT 0 0

The diagram illustrates the state of the dept table across three stages:

- ①の結果:** dept\_id | name  
-----+---  
1 | AX  
2 | BX  
3 |  
4 | AX
- ⑤の結果:** dept\_id | name  
-----+---  
1 | AX  
2 | BX  
3 |  
4 | AX  
99 | SP
- ⑦の結果:** dept\_id | name  
-----+---  
1 | AX  
2 | BX  
3 |  
4 | AX  
99 | SP

\* 補足あり。回答後に参照推奨。

---

### ● Q112 UPSERT2

## 【問題】

- ① dept\_id 98のレコードがdeptテーブルにない場合は以下情報を登録したいが、ない場合はdept\_id 98のnameを「重複」に更新したい。  
( 98, 'secret' )
- ② deptテーブルを確認。
- ③ dept\_id 98のレコードがdeptテーブルにない場合は以下情報を登録したいが、ない場合はdept\_id 98のnameを「重複」に更新したい。  
( 98, 'secret2' )
- ④ deptテーブルを確認。

## 【期待出力】

- ① INSERT 0 1
- ②④ 右枠参照
- ③ INSERT 0 1

The diagram illustrates the state of the dept table across two stages:

- ②の結果:** dept\_id | name  
-----+---  
1 | AX  
2 | BX  
3 |  
4 | AX  
99 | SP  
98 | secret
- ④の結果:** dept\_id | name  
-----+---  
1 | AX  
2 | BX  
3 |  
4 | AX  
99 | SP  
98 | 重複

### ● Q113 UPSERT3

#### 【問題】

① dept\_id 98,99に対し、それぞれ以下のように設定したい。

(98,'秘密')(99,'特殊')

上記がなければそのまま登録したいし、重複していれば  
指定したもので更新したい。

② deptテーブルを確認。

③ dept\_id 98 と99を削除する。

#### 【期待出力】

① INSERT 0 2

② 右枠参照

| ②の結果    |      |
|---------|------|
| dept_id | name |
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |
| 98      | 秘密   |
| 99      | 特殊   |

### ● Q114まとめ

#### 【問題】

端末の保持数が一般的(1つ)ではないユーザを取得したいが、  
ユーザーを管理しているusersテーブル、及び部署を管理するdeptテーブルが  
不明な原因により汚れている。

以上の事から、少し工夫をして上記情報を取得したい。  
具体的には以下のように情報を取得したい。

①: ②と③のどちらかにヒットするuser\_idを取得  
(重複がある場合は排除する事)

②: deviceテーブルでユーザごとに端末所持数を集計し、  
所持数が0か2のuser\_idのみ取得  
※ 所持数条件はorを使わずに指定

③: ユーザの所属するグループ名がnullでなく、  
かつユーザ名が[AX][BX]ではない(部署名ではない)ユーザのuser\_idを取得  
(deptテーブルとusersテーブルの汚れ対策)  
※ 結合を使う場合は条件一致するものだけを取得する方法で実施すること

※ なお、①～③を取得する際に3つのテーブルをjoinする方法は避ける事

#### 【期待出力】

| user_id |
|---------|
| 4       |
| 1       |
| 2       |

## 6章 高度な操作問題

### その1:外部制約の詳細や条件文等

---

#### ■ 事前準備

新たなDB(test06a)を使う。

```
psql -U postgres -d postgres
create database test06a;
\q

psql -U postgres -d test06a
create table student (id int primary key default 999 ,name text ,grade int ,class int ,c_group int ,eval int);

insert into student (id ,name ,grade ,class ,c_group ,eval) values (1, 'AAA', 1, 1, 1, 70),(2, 'BBB', 2, 1, 1, 80),
(3, 'CCC', 1, 2, 1, 50);
```

上記にて完成するテーブルは以下。

[student]

| <b>Id(Pkey, default 999)</b> | <b>name</b> | <b>grade</b> | <b>class</b> | <b>c_group</b> | <b>eval</b> |
|------------------------------|-------------|--------------|--------------|----------------|-------------|
| 1                            | AAA         | 1            | 1            | 1              | 70          |
| 2                            | BBB         | 2            | 1            | 1              | 80          |
| 3                            | CCC         | 1            | 2            | 1              | 50          |

---

#### ● Q115 自ら定義する型

##### 【問題】

- ① 郵便番号を登録する為の「郵便番号」型(post\_code)を作成する。  
なお郵便番号は「XXX-YYYY」形式固定とする。  
(XとYは数値)
- ② [id(int), name(text), p\_code(post\_code)]列を持つusersテーブルを作成する。
- ③ usersテーブルに「1, AAA, 111-2222」を登録する。
- ④ 指定と異なるフォーマットが入った時の動作を確認する。  
usersテーブルに「2, BBB, 111-33333」を登録する。
- ⑤ usersテーブルのidが1のpost\_codeを「111-33333」に更新しようとする。

##### 【期待出力】

- ① CREATE DOMAIN
- ② CREATE TABLE
- ③ INSERT 0 1
- ④⑤ ERROR: value for domain post\_code violates check  
constraint "post\_code\_check"  
※ 指定したフォーマットと合わない為、エラーとなる事。

### ● Q116 プリペアドステートメントを対話モードで試す

#### 【問題】

- ① 前問題で作ったusersテーブルの情報一覧を取得する  
プリペアドステートメント(p\_select)を用意する。
- ② p\_selectを実施し、usersテーブルの情報一覧を取得する。
- ③ usersテーブルに情報を登録するプリペアドステートメント(p\_insert)を用意する。(id, name, post\_codeそれぞれ値を指定できるようにすること)
- ④ p\_insertから[2,BBB,111-3333]を登録する。
- ⑤ p\_selectで④で入れた情報が入っているか確認する。
- ⑥ p\_insert, p\_selectを削除する。

#### 【期待出力】

- ①③ PREPARE  
 ②⑤ 右の青枠参照。  
 ④ PREPARE  
 ③ INSERT 0 1  
 ⑥ DEALLOCATE × 2

| ② | id   name   p_code | ⑤                  |
|---|--------------------|--------------------|
|   | -----+-----+-----  | id   name   p_code |
|   | 1   AAA   111-2222 | -----+-----+-----  |
|   |                    | 1   AAA   111-2222 |
|   |                    | 2   BBB   111-3333 |

### ● Q117 外部キー制約のアクション 1

#### 【問題】

- ①以下の構成のテーブルを作成する。その際、employeeテーブルのdept\_nameをdeptテーブルのnameを参照するように外部キー制約を付与する。その際のアクションはupdate/deleteともにcascadeとする

| [dept]        | [employee]                               |
|---------------|------------------------------------------|
| id(int)<br>参照 | id(int)<br>dept_name(text)<br>name(text) |

[advance] ② employeeテーブルの登録に必要な対応を施し、再度①の条件でemployテーブルの登録を行う。

- ③deptに以下を登録する  
(1, "AX"), (2, "BX"), (3, "CX")  
その後、employeeに以下を登録する  
(1, "AX", "AAA"), (2, "BX", "BBB"), (3, "CX", "CCC")
- ④ employeeのdept\_nameをtest!に変更しようとする(外部制約が動いているか確認)
- ⑤ 現在のemployeeテーブルの状態を確認の上、deptテーブルのAXをDXに変更する。  
その後、employeeテーブルも切り替わっているか確認を行つ。
- ⑥ deptテーブルのCXを削除する。  
その後、employeeテーブルもCX所属ユーザ(CCC)が消えているか確認する

#### 【期待出力】

- ①deptテーブルは成功。  
 employeeテーブルはエラー。  
 ERROR: there is no unique constraint matching given keys for referenced table "dept"  
 ②③処理成功  
 ④ ERROR: insert or update on table "employee" violates foreign key constraint "employee\_dept\_name\_fkey"  
 DETAIL: Key (dept\_name)=(test) is not present in table "dept".

| ⑤(変更前)                                                                                     | ⑤(変更後)                                                                                     | ⑥(削除後)                                                                                        |
|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| id   dept_name   name<br>-----+-----+-----<br>1   AX   AAA<br>2   BX   BBB<br>3   CX   CCC | id   dept_name   name<br>-----+-----+-----<br>2   BX   BBB<br>3   CX   CCC<br>1   DX   AAA | id   dept_name   name<br>-----+-----+-----<br>2   BX   BBB<br>1   DX   AAA<br>※ id 3 cccがないこと |

### ● Q118 外部キー制約のアクション2

#### 【問題】

- ① employeeテーブルに前問題-②と同様の必要な対応を施した上で、  
studentテーブルのidはemployeeテーブルのidを参照する設定を入れる。  
その際、employee側のidが変更された場合nullとなる、  
deleteされた場合デフォルト値が入るように設定する。



- ② 必要な対応を行った上で再度studentテーブルに外部制約を登録。  
③ employeeテーブル(参照先)のid 1を id 4に変更。  
④ ③のエラーを言解消する対応を実施。  
⑤ studentの現在の状況を確認した上で、employeeテーブルのid 1を 4に変更。  
その後、再度studentテーブルを確認する  
⑥ employeeテーブルの id 2 のレコードを削除する。  
⑦ ⑥のエラーを解消するための必要な対応を行う  
⑧ studentの現在の状況を確認した上で、employeeテーブルの id 2 のレコードを削除する。  
その後、再度studentテーブルを確認する

#### 【期待出力】

① employeeテーブルの処理は成功。studentテーブルの設定は失敗。(以下エラーログ)  
ERROR: insert or update on table "student" violates foreign key constraint "student\_id\_fkey"  
DETAIL: Key (id)=(3) is not present in table "employee".

② 処理成功

③ エラーとなる。

~~~

ERROR: null value in column "id" violates not-null constraint  
DETAIL: Failing row contains (null, AAA, 1, 1, 1, 70).  
CONTEXT: SQL statement "UPDATE ONLY "public"."student" SET "id" = NULL WHERE \$1  
OPERATOR(pg\_catalog.=)"id""

~~~

④⑦ 処理成功

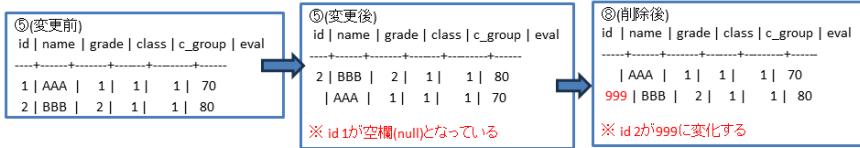
⑤⑧ 以下図参照

⑥ エラーとなる

~~~

ERROR: insert or update on table "student" violates foreign key constraint "student\_id\_fkey"  
DETAIL: Key (id)=(999) is not present in table "employee".

~~~



● Q119 制約解除

**【問題】**

- 【advance】**①deptテーブルを削除する  
 ②studentテーブルがemployeeテーブルを  
 参照する外部キーを削除する。

**【期待出力】**

- ①② 処理成功

● Q120 複合外部キー制約

**【問題】**

- ① personテーブル[ id(int), name(text) ] を新たに作り、employeeのid, nameを参照する  
 複合外部キー制約を試したい。  
 既にemployeeテーブルのidにはunique制約がついているはずである。  
 追加でemployeeテーブルのnameにunique制約を付けたうえで、  
 personテーブルを作成する。(失敗する)  
 ② 必要な対処を行い、personテーブルを作成する。  
 ③ 参照先のemployeeテーブルの情報一覧を取得する。  
 ④ 複合外部キー制約を試す。personテーブルに(4, AAA)を入れる。(成功)  
 ⑤ 複合外部キー制約を試す。personテーブルに(999, BBB)を入れる。(失敗)  
 ⑥ 複合外部キー制約を試す。personテーブルに(999, AAA)を入れる。(失敗)  
 ⑦ personテーブルを削除する。

**【期待出力】**

① ALTER TABLE

ERROR: there is no unique constraint matching given keys for referenced table "employee"

②④⑦ 処理成功

⑤⑥ 2列とも完全一致でないと成功しない。(以下エラー文)  
 ~~~~

| ③   | id | dept_name | name |
|-----|----|-----------|------|
| 4   | DX |           | AAA  |
| 999 | BX |           | ZZZ  |

ERROR: insert or update on table "person" violates foreign key constraint "person\_id\_name\_fkey"  
 DETAIL: Key (id, name)=(登録しようしたid, 登録しようとしたname) is not present in table "employee".

\* 補足あり。回答後、参照推奨。

### ● Q121 条件に応じた出力

## 【問題】

- ① studentテーブルに以下を追加登録する。  
(3, 'CCC', 1, 2, 1, 50), (4, 'DDD', 2, 2, 1, 50), (5, 'EEE', 1, 2, 1, 90),  
(6, 'FFF', 2, 1, 2, 60)

② studentテーブルにて、grade + class + c\_groupごとに評価(eval)の  
平均値を出し、80点以上なら○、60~79点なら△、59点以下は×  
とする判定(judge)列を出力する。

### 【期待出力】

## ②の結果

| grade | class | c_group | groupavg          | jedge |
|-------|-------|---------|-------------------|-------|
| 2     | 1     | 2       | 60.00000000000000 | △     |
| 2     | 1     | 1       | 80.00000000000000 | ○     |
| 1     | 1     | 1       | 70.00000000000000 | △     |
| 2     | 2     | 1       | 50.00000000000000 | ×     |
| 1     | 2     | 1       | 70.00000000000000 | △     |

### ● Q122 複数パターンによる集約処理の出力

【問題】※ 以下①～③は全て1文で実施する事

- ① studentテーブルの grade , class , c\_group それぞれ独立に評価(eval)の平均値を出力する。
  - ② studentテーブルの grade , class , c\_group の組み合わせごとに評価平均値を出力する。  
(ただし左側のほうが優先度が高く、間がない組み合わせ  
(例: gradeとc\_groupの組み合わせ)は出力不要。)
  - ③ studentテーブルの grade , class , c\_group の全組み合わせの評価平均値を出力する。

### 【期待出力】

## ①の結果

grade | class | c\_group | avg

| 2 |  |   | 63.33333333333333     |
|---|--|---|-----------------------|
| 1 |  |   | 70.00000000000000     |
|   |  | 2 | 63.33333333333333     |
|   |  | 1 | 70.00000000000000     |
|   |  |   | 2   60.00000000000000 |
|   |  |   | 1   68.00000000000000 |

## ②の結果 ※一部抜粋

| grade | class | c_group | avg               |
|-------|-------|---------|-------------------|
| 1     | 2     | 1       | 70.00000000000000 |
| 2     | 1     |         | 70.00000000000000 |
| 2     |       |         | 63.33333333333333 |

### ③の結果 ※一部抜粋

```
⑤の結果 ふた印が付く
grade | class | c_group | avg
-----+-----+-----
 | 1 | 1 | 75.000000000000000000000000
1 | | 1 | 70.000000000000000000000000
```

## その2:パーティションとテーブル空間

### ■ 事前準備

以下SQLをpostgres DB , postgresユーザで繋いで実行する。

```
psql -U postgres -d postgres
create database test06b;
\$q
```

またLinux(OS)側で、以下ディレクトリを作成しておく。

```
mkdir ~/12/data01_ts1
mkdir ~/12/data01_ts2
```

その後、作成したtest06bにアクセスする。

```
psql -U postgres -d postgres
```

### ● Q123 リストパーティションとテーブル空間

#### 【問題】

- ① data01\_ts1をデータの配置場所とするts1 テーブル空間を作成する。  
同じようにdata01\_ts2をデータの配置場所とするts2も作成する。
- ② id(int), dept(text), name(text), memo(timestamp)列を有する  
person\_listテーブルを作成する。  
このテーブルは宣言的パーティショニング(リストパーティション)の  
親テーブルとし、deptの値で対象のテーブルを決める。
- ③ deptが「AX」「BX」の時にそれぞれ入る子テーブルを  
2つ(person\_ax, person\_bx)作成する。  
person\_axにはts1を、person\_bxにはts2を設定する。

#### 【期待出力】

- ① CREATE TABLESPACE × 2
- ② CREATE TABLE
- ③ CREATE TABLE × 2

### ● Q124 リストパーティションとテーブル空間の動作確認

#### 【問題】

- ① person\_listテーブルに対し、以下の情報を登録する  
「1, AX, AAA, now()」「2, BX, BBB, now()」「3, AX, CCC, now()」
- ② person\_ax, person\_bx, person\_listテーブルをそれぞれレコード全取得する。
- ③ テーブル空間の一覧を取得する。
- ④ 作成したテーブル空間とデータの配置場所(パス)を取得する。
- ⑤ DBとの接続を切断し、データの配置場所に物理データファイルがあるか確認する。

#### 【期待出力】

① INSERT 0 3

⑤ /var/lib/postgresql/12/data01\_ts1配下にディレクトリができていること

|                                                                                                                                                                                                            |                                                                                                                                                                                                                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ② person_ax<br>id   dept   name   memo<br>-----+-----+-----+-----<br>1   AX   AAA   2024-07-20 14:29:29.248169<br>3   AX   CCC   2024-07-20 14:29:29.248169                                                | ② person_bx<br>id   dept   name   memo<br>-----+-----+-----+-----<br>2   BX   BBB   2024-07-20 14:29:29.248169                                                                                                   |
| ② person_list<br>id   dept   name   memo<br>-----+-----+-----+-----<br>1   AX   AAA   2024-07-20 14:29:29.248169<br>3   AX   CCC   2024-07-20 14:29:29.248169<br>2   BX   BBB   2024-07-20 14:29:29.248169 | ③ oid   spcname   pg_tablespace_location<br>-----+-----+-----<br>1663   pg_default  <br>1664   pg_global  <br>16392   ts1   /var/lib/postgresql/12/data01_ts1<br>16393   ts2   /var/lib/postgresql/12/data01_ts2 |

\* 補足あり。回答後、確認推奨。

### ● Q125 リストパーティションの子テーブルを跨いだ処理

#### 【問題】

- ① test06bに再接続し、person\_list(親テーブル)に対し、deptに対しidx1というindexを付与する  
(データの実体は子テーブルだが、親テーブルにindexを付与できるか確認する)
- ② person\_listを指定し、idが3のユーザのdeptをBXに変更する
- ③ person\_axとperson\_bxをそれぞれレコード一覧を確認し、  
元々AXにいたid3のユーザ情報がテーブル移動したことを確認する。
- ④ person\_listを指定し、全ユーザの名前をSamenameに変更する。  
(親テーブルから、子テーブルを跨いだupdateを実施する)
- ⑤ person\_listテーブルの全レコードを確認する

#### 【期待出力】

① CREATE INDEX

② UPDATE 1

④ UPDATE 3

|                                                                                                                                                                                                               |                                                                                                                                                             |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ③ person_ax<br>id   dept   name   memo<br>-----+-----+-----+-----<br>1   AX   AAA   2024-07-20 14:29:29.248169                                                                                                | ③ person_bx<br>id   dept   name   memo<br>-----+-----+-----+-----<br>2   BX   BBB   2024-07-20 14:29:29.248169<br>3   BX   CCC   2024-07-20 14:29:29.248169 |
| ⑤<br>id   dept   name   memo<br>-----+-----+-----+-----<br>1   AX   SameName   2024-07-20 14:29:29.248169<br>2   BX   SameName   2024-07-20 14:29:29.248169<br>3   BX   SameName   2024-07-20 14:29:29.248169 |                                                                                                                                                             |

### ● Q126 リストパーティションと主キーについて

#### 【問題】

- ① id(int), dept(text), name(text), memo(timestamp)列を有するperson\_list2テーブルを作成する。  
 deptをリストパーティションのパーティションキーとすること。  
 またidを主キーと設定すること。(失敗)  
 なぜ失敗したかも考える事。
- ② idは主キーとしたい。①の結果も踏まえ、改善案を考え、実行する事。

#### 【期待出力】

①

```
2024-07-20 14:49:20.432 UTC [333982] ERROR: unique constraint on partitioned table must
include all partitioning columns
```

```
2024-07-20 14:49:20.432 UTC [333982] DETAIL: PRIMARY KEY constraint on table
"person_list2" lacks column "dept" which is part of the partition key.
```

```
2024-07-20 14:49:20.432 UTC [333982] STATEMENT: create table person_list2 (id int primary
key , dept text, name text , memo timestamp) partition by list (dept) ;
ERROR: unique constraint on partitioned table must include all partitioning columns
DETAIL: PRIMARY KEY constraint on table "person_list2" lacks column "dept" which is part of
the partition key.
```

② CREATE TABLE

### ● Q127 リストパーティションの範囲外の値について

#### 【問題】

- ① AXとBX用の子テーブルを作った。ではそれ以外の部署の情報を入れようするとどうなるのか。  
 「4, CX, DDD, now()」で登録してみる。  
 [advance] ② ①の対策をとる。直接「dept: CX」のテーブルは作成しない事。  
 AX, BX以外(CXでもDXでもなんでも可)が所属できる対応を入れる。
- ③ ①と同じ処理を行う  
 ④ person\_listの全レコードを取得する。また②対処の全レコード確認を行う

#### 【期待出力】

①

```
ERROR: no partition of relation "person_list" found for row
DETAIL: Partition key of the failing row contains (dept) = (CX).
```

② ヒント欄に記載

③ INSERT 0 1

| ④person_list |      |          |                            |
|--------------|------|----------|----------------------------|
| id           | dept | name     | memo                       |
| 1            | AX   | SameName | 2024-07-20 14:29:29.248169 |
| 2            | BX   | SameName | 2024-07-20 14:29:29.248169 |
| 3            | BX   | SameName | 2024-07-20 14:29:29.248169 |
| 4            | CX   | DDD      | 2024-07-20 14:54:48.087079 |

| ④ ②での対応 |      |      |                            |
|---------|------|------|----------------------------|
| id      | dept | name | memo                       |
| 4       | CX   | DDD  | 2024-07-20 14:54:48.087079 |

### ● Q128 パーティションテーブルのデータ削除について

【問題】

- ① person\_listにてidが3,4のレコードを同時に削除する。(子テーブルをまたがった削除を行う)  
その後、person\_listを確認する
- ② person\_listテーブルを削除する
- ③ person\_ax(子テーブル)の全レコードを取得しようとする
- ④ テーブル空間のts1とts2を削除する
- ⑤ テーブル空間の一覧を確認する
- ⑥ DBの接続を解除して、物理データ格納ディレクトリを確認し、  
ファイルがなくなっていることを確認する。確認後test06b DBに再接続する。

【期待出力】

① DELETE 2。全レコード取得結果は右参照。

② DROP TABLE

③ ERROR: relation "person\_ax" does not exist

※ 親テーブルを削除すると、自動で子テーブルも削除される

④ DROP TABLESPACE × 2

⑤ 右参照

⑥ /var/lib/postgresql/12/data01\_ts1/ 配下に何もない事

| ① | id | dept | name     | memo                       |
|---|----|------|----------|----------------------------|
|   | 1  | AX   | SameName | 2024-07-20 14:29:29.248169 |
|   | 2  | BX   | SameName | 2024-07-20 14:29:29.248169 |

| ⑤ | oid  | spcname    | pg_tablespace_location |
|---|------|------------|------------------------|
|   | 1663 | pg_default |                        |
|   | 1664 | pg_global  |                        |

### ● Q129 レンジパーティション

【問題】

- ① id(int), dept(text), name(text), memo(timestamp)列を有するperson\_rangeテーブルを作成する。  
memoをレンジパーティションのパーティションキーとすること。
- ② 2024/1/1～2024/12/31の範囲を格納する子テーブル(person\_2024),  
2023/1/1～2023/12/31の範囲を格納する子テーブル(person\_2023),  
それ以外の範囲を格納する子テーブル(person\_default)をそれぞれ作成する
- ③ 「1, AX, AAA, 2024年timestamp」「2, BX, BBB, 2024から1年前のtimestamp」  
「3, AX, CCC, now()から2年前のtimestamp」を登録する。(now()を使ってtimestamp指定する事)
- ④ 全子テーブルと、親テーブルの情報を取得する。
- ⑤ person\_rangeテーブルを削除する。

【期待出力】

- ① CREATE TABLE  
② CREATE TABLE × 3  
③ INSERT 0 3  
④ DROP TABLE

| ④person_range | id | dept | name | memo                       |
|---------------|----|------|------|----------------------------|
|               | 2  | BX   | BBB  | 2023-07-20 15:18:10.628689 |
|               | 1  | AX   | AAA  | 2024-07-20 15:18:10.628689 |
|               | 3  | AX   | CCC  | 2022-07-20 15:18:10.628689 |

| ④person_2024 | id | dept | name | memo                       |
|--------------|----|------|------|----------------------------|
|              | 1  | AX   | AAA  | 2024-07-20 15:18:10.628689 |

| ④person_2023 | id | dept | name | memo                       |
|--------------|----|------|------|----------------------------|
|              | 2  | BX   | BBB  | 2023-07-20 15:18:10.628689 |

| ④person_default | id | dept | name | memo                       |
|-----------------|----|------|------|----------------------------|
|                 | 3  | AX   | CCC  | 2022-07-20 15:18:10.628689 |

### ● Q130 ハッシュパーティション

## 【問題】

- ① id(int), dept(text), name(text), memo(timestamp)列を有するperson\_hashテーブルを作成する。  
idをハッシュパーティションのパーティションキーとすること。
- ② 3テーブルで分割する事を想定し、まずは余りが0のケース(person\_0)、1のケース(person\_1)の子テーブルを作成する。
- ③ ②以外のパターンを想定するため、デフォルトテーブルを作る
- ④ ②と同じ方法で余り2のケース(person\_2)を想定した子テーブルを作成する
- ⑤ 「1, AX, AAA, now()」「2, BX, BBB, now()」「3, AX, CCC, now()」を挿入する
- ⑥ 全子テーブルと、親テーブルで情報取得する
- ⑦ person\_hashテーブルを削除する

## 【期待出力】

```

① CREATE TABLE
② CREATE TABLE × 2
③
2024-07-20 15:26:07.861 UTC [336441] ERROR: a hash-partitioned table may not have a default partition
2024-07-20 15:26:07.861 UTC [336441] STATEMENT: create table person_default partition of person_hash default;
ERROR: a hash-partitioned table may not have a default partition

```

- ④ CREATE TABLE
- ⑤ INSERT 0 3
- ⑦ DROP TABLE

| ⑥ person_0                                                                      |
|---------------------------------------------------------------------------------|
| ⑥ person_1                                                                      |
| ⑥ person_2                                                                      |
| id   dept   name   memo<br>+-----+<br>2   BX   BBB   2024-07-20 15:29:38.341222 |
| id   dept   name   memo<br>+-----+<br>3   AX   CCC   2024-07-20 15:29:38.341222 |
| id   dept   name   memo<br>+-----+<br>1   AX   AAA   2024-07-20 15:29:38.341222 |

| ⑥ person_hash                                                                                                                                                             |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id   dept   name   memo<br>+-----+<br>2   BX   BBB   2024-07-20 15:29:38.341222<br>3   AX   CCC   2024-07-20 15:29:38.341222<br>1   AX   AAA   2024-07-20 15:29:38.341222 |

### その3:contribとdblink

---

#### ■ 事前準備(DB クラスタ追加作成)

本演習では2つのDBが必要となる為、リモート接続を想定した別DBを用意する。

以下手順を実施する事。

本件で実施していることの説明は補足に記載している。

気になる方は参照してほしい。

以下コマンドを実施し、DBクラスタを複数作る

```
/usr/lib/postgresql/12/bin/initdb -D /var/lib/postgresql/12/data02
vi /var/lib/postgresql/12/disk02/postgresql.conf
```

~~~

```
port = 5433
```

~~~

```
/usr/lib/postgresql/12/bin/pg_ctl -D /var/lib/postgresql/12/data02 start
```

```
psql -p 5433
```

```
¥q
```

※ 補足あり。手順実施後、参照推奨。(■を★にかえてタイトルとあわせて検索推奨)

---

### ● Q131 dblink による DB 作成

#### 【問題】

- ① ポート5432(デフォルト)のDBに接続し、DB (test06c)を作成する。
- ② dblinkを使用できるようにする。
- ③ dblinkを使用し、デフォルトDBに接続したまま、[ポート5433側\(リモートDB\)](#)にもtest06c DBを作成する。
- ④ 現在の接続を切断し、[ポート5433](#)のtest06cに接続できるか確認する。
- ⑤ 現在の接続を切断し、ポート5432のtest06cに接続できるか確認する。

#### 【期待出力】

- ① CREATE DATABASE
  - ② CREATE EXTENSION
  - ③
- 
- ```
dblink_exec
```

```
-----  
CREATE DATABASE
```

- ④⑤ test06c=# に切り替わる事(問題なく接続できる事)

● Q132 dblink に寄るテーブル作成

【問題】

- ① 接続中のデフォルトDBにて id(int), dept(text), name(text), memo(text)の列を持つ person_1テーブルを作成する
- ② デフォルトDBに接続したまま。[ポート5433](#)側に①と同じ列を持つperson_2を作成する(失敗)
- ③ ②の失敗原因を改善する
- ④ ②と同じSQLを実施

【期待出力】

- ① CREATE TABLE
 - ②
-
- ```
2024-10-08 18:19:56.786 UTC [1214093] ERROR: function dblink_exec(unknown, unknown) does not exist at character 8
2024-10-08 18:19:56.786 UTC [1214093] HINT: No function matches the given name and argument types. You might need
to add explicit type casts.
2024-10-08 18:19:56.786 UTC [1214093] STATEMENT: select dblink_exec('dbname=test06c user=postgres host=127.0.0.1
port=5433', 'create table person_2 (id int , dept text, name text , memo text)');
ERROR: function dblink_exec(unknown, unknown) does not exist
LINE 1: select dblink_exec('dbname=test06c user=postgres host=127.0....
^
HINT: No function matches the given name and argument types. You might need to add explicit type casts.
```
- ③ ヒント参照
  - ⑤
- 
- ```
dblink_exec
```
-
- ```
CREATE TABLE
```

● Q133 dblink による挿入とレコード取得

## 【問題】

- ① デフォルトDB側に「1,AX,AAA,01」「2,BX,BBB,01」「3,AX,CCC,01」を登録する
- ② デフォルトDBに接続したまま、リモートDB(ポート5433)のperson\_2に  
「4,CX,DDD,02」「5,CX,EEE,02」を登録する
- ③ デフォルトDBに接続したまま、person\_1とperson\_2の結果をまとめて  
取得する

[advance] ④ id順に並び替えたうえで③の結果を取得する

## 【期待出力】

① INSERT 0 3

②

dblink\_exec

-----

INSERT 0 2

| (3) |    |      |      |      |
|-----|----|------|------|------|
|     | id | dept | name | memo |
|     | 5  | CX   | EEE  | 02   |
|     | 4  | CX   | DDD  | 02   |
|     | 1  | AX   | AAA  | 01   |
|     | 2  | BX   | BBB  | 01   |
|     | 3  | AX   | CCC  | 01   |

| (4) |    |      |      |      |
|-----|----|------|------|------|
|     | id | dept | name | memo |
|     | 1  | AX   | AAA  | 01   |
|     | 2  | BX   | BBB  | 01   |
|     | 3  | AX   | CCC  | 01   |
|     | 4  | CX   | DDD  | 02   |
|     | 5  | CX   | EEE  | 02   |

● Q134 dblink を使った update, delete

## 【問題】

- ① デフォルトDBに接続した状態で、person\_2側にあるid 5のdeptを「DX」に変える
- ② デフォルトDBに接続した状態で、person\_2側にあるid 4のレコードを削除する
- ③ デフォルトDBに接続した状態で、person\_1とperson\_2の情報をまとめて取得する  
(ソートして出す事)

## 【期待出力】

①

dblink\_exec

-----

UPDATE 1

②

dblink\_exec

-----

DELETE 1

| (3) id 4がなく、5のdeptがDXになっている |    |      |      |      |
|-----------------------------|----|------|------|------|
|                             | id | dept | name | memo |
|                             | 1  | AX   | AAA  | 01   |
|                             | 2  | BX   | BBB  | 01   |
|                             | 3  | AX   | CCC  | 01   |
|                             | 5  | DX   | EEE  | 02   |

\* 本問題はヒントなし。

### ● Q135 2相コミットメントを使う為の準備

#### 【問題】

- ① DBの接続を解除し、デフォルトDB(data01,5432),  
[リモートDB\(data02,5433\)](#)に2相コミットメントを使う為の  
 準備を実施する。  
 準備ができたらデフォルトDB側のtest06cに接続する。

#### 【期待出力】

- ① ヒント欄参照

### ● Q136 2相コミットメント(処理成功)

#### 【問題】

- ① 2相コミットメントを実施する。  
 まずはデフォルトDB側で、person\_1に対し「6,BX,FFF,01」を登録し、「local」トランザクションとして  
 準備完了とする  
 ② デフォルトDB側から、[person\\_2](#)に対し「7,CX,HHH,02」を登録し、「remote」トランザクションとして  
 準備完了とする  
 ③ デフォルトDB側と、[リモートDB\(5433\)](#)側をそれぞれ2相コミットメント用トランザクションがいるか  
 確認する(デフォルトDB側から確認実施)  
 ④ デフォルトDB側から、各DBにcommit処理を実行する  
 ⑤ ③と同じ事をし、2相コミットメント用トランザクションがなくなったことを確認する  
 ⑥ デフォルトDB側から、person\_1と[person\\_2](#)を合わせた結果を取得する(ソート済)

#### 【期待出力】

```

①
BEGIN
INSERT 0 1
PREPARE TRANSACTION

②
dblink_exec

PREPARE TRANSACTION

④ COMMIT PREPARED
dblink_exec

COMMIT PREPARED

```

| ③ デフォルトDB(5432)側 |       |                               |          |          |
|------------------|-------|-------------------------------|----------|----------|
| transaction      | gid   | prepared                      | owner    | database |
| 499              | local | 2024-07-21 05:53:40.424008+00 | postgres | test06c  |

| ④ リモートDB(5433)側 |        |                            |          |           |
|-----------------|--------|----------------------------|----------|-----------|
| transaction1    | gid1   | prepared1                  | ownerid1 | database1 |
| 492             | remote | 2024-07-21 05:53:46.771507 | postgres | test06c   |

| ⑤ 5432も5433も以下となる |     |          |       |          |
|-------------------|-----|----------|-------|----------|
| transaction       | gid | prepared | owner | database |
| (0 rows)          |     |          |       |          |

| ⑥ 5432も5433も以下となる |      |      |      |  |
|-------------------|------|------|------|--|
| id                | dept | name | memo |  |
| 1                 | AX   | AAA  | 01   |  |
| 2                 | BX   | BBB  | 01   |  |
| 3                 | AX   | CCC  | 01   |  |
| 5                 | DX   | EEE  | 02   |  |
| 6                 | BX   | FFF  | 01   |  |
| 7                 | CX   | HHH  | 02   |  |

### ● Q137 2相コミットメント(処理失敗)

## 【問題】

- ① 2相コミットメントを実施する。  
まずはデフォルトDB側で、`person_1`に対し「8,AX,III,01」を登録し、「local」トランザクションとして準備完了とする
- ② デフォルトDB側から、`person_2`に対し「9,CX,III,02」を登録し、「remote」トランザクションとして準備完了とする
- ③ デフォルトDB側で5435側をロールバックする
- ④ [リモートDB\(5433\)側](#)の2相コミットメント用トランザクションが残っているか確認する
- ⑤ [リモートDB\(5433\)側](#)もロールバックする
- ⑥ デフォルトDB、[リモートDB\(5433\)](#)に2相コミットメント用トランザクションがなくなったか確認する  
デフォルト側から、`person_1`と`person_2`を合わせた結果を取得する(ノート済)

## 【期待出力】

```

①
BEGIN
INSERT 0 1
PREPARE TRANSACTION
②
dblink_exec

PREPARE TRANSACTION
③ ROLLBACK PREPARED
⑤
dblink

("ROLLBACK PREPARED")

```

| ④                                                | transaction1 | gid1     | prepared1                  | ownerid1 | database1 |
|--------------------------------------------------|--------------|----------|----------------------------|----------|-----------|
|                                                  | 493          | remote   | 2024-07-21 06:05:07.162733 | postgres | test06c   |
| <b>⑥ トランザクション確認(5432, 5433ともに以下となる)</b>          |              |          |                            |          |           |
| transaction                                      | gid          | prepared | owner                      | database |           |
| (0 rows)                                         |              |          |                            |          |           |
| <b>⑥ person_1と2のまとめた結果<br/>(id 8, 9が増えていない事)</b> |              |          |                            |          |           |
| id                                               | dept         | name     | memo                       |          |           |
| 1                                                | AX           | AAA      | 01                         |          |           |
| 2                                                | BX           | BBB      | 01                         |          |           |
| 3                                                | AX           | CCC      | 01                         |          |           |
| 5                                                | DX           | EEE      | 02                         |          |           |
| 6                                                | BX           | FFF      | 01                         |          |           |
| 7                                                | CX           | HGG      | 02                         |          |           |

## ■ 後片付け

問題を解き終えたら以下を実行しておく事。

```
\q
```

```
/usr/lib/postgresql/12/bin/pg_ctl -D /var/lib/postgresql/12/data02 stop
```

## その4:PL/pgSQLと自動処理

---

### ■ 事前準備

#### 事前準備1

以下のSQLをpostgres DB, postgresユーザで繋いで実行する。

```
create database test06d;
create table dept (dept_id int, name text, psn_cnt int);
create table person (id int, dept text, name text, memo text);
create table tbl_history (ope_time timestamp, memo text);

insert into dept(dept_id, name) values (1, 'AX'), (2, 'BX');
```

以下のような表が出来上がったイメージ。  
社内管理システムを想定し、deptは部署名、personは従業員、tbl\_historyはテーブル操作の履歴をとるテーブルとする。

【deptテーブル】

| dept_id | name | psn_cnt |
|---------|------|---------|
| 1       | AX   |         |
| 2       | BX   |         |

【personテーブル】

| id | dept | name | memo |
|----|------|------|------|
|    |      |      |      |
|    |      |      |      |
|    |      |      |      |
|    |      |      |      |

【tbl\_historyテーブル】

| ope_time | memo |
|----------|------|
|          |      |

#### ● Q138 シンプルなトリガーの用意

##### 【問題】

- ① 呼ばれたらhistoryテーブル(not tbl\_history。意図的に存在しないテーブルとする)のope\_timeに現在時刻、memoに"simple1"と入れる  
ファンクション(simple\_func)を作成する。
- ② personテーブルにinsert処理がされた後に、処理された行ごとに  
simple\_funcを呼び出すトリガー(simple\_func\_tg)を用意する
- ③ ①で作成したsimple\_funcの情報を確認するコマンドを実行する
- ④ ②で作成したsimple\_func\_tgを確認するコマンドを実行する

##### 【期待出力】

- ① CREATE FUNCTION
- ② CREATE TRIGGER

|                                                                                                                                                                           |                                                                                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| ③ ( \$x 実行時の結果 )<br>proname   simple_func<br>prosrc  <br>  begin +<br>  insert into history( ope_time, memo ) values ( now(), "simple1"); +<br>  return null; +<br>  end; | ④ ( \$x 実行時の結果 )<br>tgname   simple_func_tg<br>tgrelid   person<br>proname   simple_func |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|

### ● Q139 トリガーの動作確認

## 【問題】

- ① personテーブルに「1, 'AX', 'AAA', 'memo」を挿入する
- ② personテーブルの全レコード情報を取得
- ③ ①が動くように正しい設定を行う
- ④ 再度①と同じ処理を実施する
- ⑤ ①で登録したデータのmemoを「memo2」に更新する
- ⑥ personテーブルとtbl\_historyの全レコード情報を取得。
- ⑦ のinsert文と、トリガー・ファンクションが効いていることを確認する。

## 【期待出力】

①

```
ERROR: relation "history" does not exist
LINE 1: insert into history(ope_time, memo) values (now(), "simpl...
QUERY: insert into history(ope_time, memo) values (now(), "simple1")
CONTEXT: PL/pgSQL function simple_func() line 3 at SQL statement
```

② 何も登録されていない。

(ファンクション処理で失敗するとその直前処理もなかった事にされる)

③ ヒント参照。(なるべくヒントを見ずに対応してみましょう)

④ INSERT 0 1

⑤ UPDATE 1

| ⑥ personテーブル            | ⑥ tbl_historyテーブル                    |
|-------------------------|--------------------------------------|
| id   dept   name   memo | ope_time   memo                      |
| 1   AX   AAA   memo2    | 2024-07-12 11:40:11.454329   simple1 |

### ● Q140 IF文を含んだファンクション

## 【問題】

- ① 前問題でUPDATEがhistoryに登録されないことが分かった。

UPDATEとDELETEもINSERTと同じように登録されるように変更したい。必要な事項を行ってほしい。

なお、historyのメモには「処理名(※)、テーブル名、古いデータ、新しいデータ」が登録されるようになること。

※ 処理名は「INSERT/UPDATE/DELETE」のどれか。古いデータ、新しいデータは処理で

その概念があるものだけ良い。またpersonテーブルの全てのカラムに処理がされるものと想定する。

- ② ①の動確を行なう。personテーブルにinsert, update, deleteを試す。

登録データは「2, BX, BBB, memo」とし、updateはid 2の人に対しmemoを「memo2」とする。

deleteもid 2を消す。その後、tbl\_historyテーブルの全レコードを確認して、期待通り動いているか確認する。

- ③ personテーブルに「3, AX, CCC」(memoなし)で登録する。

その後、tbl\_historyテーブルの全レコードを確認を確認する。

## 【期待出力】

- ① ヒント欄参照。なお、記載が長くなるものについては一度全てを作ろうとせず、一部を作て期待通り動くか動確をする、と言う事を繰り返して作っていくと良い。

- ② INSERT 0 1 , UPDATE 1 , DELETE 1 , 情報所得結果は以下

- ③ INSERT 0 1 , 情報所得結果は以下。memo欄が空である事。

| ②                          | ope_time   memo                            |
|----------------------------|--------------------------------------------|
| 2024-07-12 11:40:11.454329 | simple1                                    |
| 2024-07-15 02:17:45.431987 | INSERT,person,2,BX,BBB,memo                |
| 2024-07-15 02:17:50.752227 | UPDATE,person,2,BX,BBB,memo2,2,BX,BBB,memo |
| 2024-07-15 02:18:25.616389 | DELETE,person,2,BX,BBB,memo2               |
| ③                          | ope_time   memo                            |
| 2024-07-12 11:40:11.454329 | simple1                                    |
| 2024-07-15 02:17:45.431987 | INSERT,person,2,BX,BBB,memo                |
| 2024-07-15 02:17:50.752227 | UPDATE,person,2,BX,BBB,memo2,2,BX,BBB,memo |
| 2024-07-15 02:18:25.616389 | DELETE,person,2,BX,BBB,memo2               |
| 2024-07-15 02:28:58.28816  |                                            |

### ● Q141 キーワードに該当する情報がない場合の処理

#### 【問題】

前問題-③でtbl\_historyテーブルのmemoが空欄になった。

この部分の処理を深堀する。

前問題-③では要素を1カラムにまとめて結合して入れていたが、今回はカラムをわけて登録する例で試す。

- ① id int, name text のカラムを持つ、tbl\_history2テーブルを作成する。
- ② tbl\_history2(id, name)に対し、新しいidと新しいnameが登録される関数(simple\_func2)を作成する。  
(特にif文による条件分岐は不要)
- ③ personテーブルにinsert処理がされたらsimple\_func2を呼び出すトリガー(simple\_func\_tg2)を作成する。
- ④ personテーブルに対し、idとmemoを「4, simple\_func2」とした情報を登録する。(nameは指定しない)  
その後、tbl\_historyとtbl\_history2を確認する。

#### 【期待出力】

- ① CREATE TABLE
- ② CREATE FUNCTION
- ③ CREATE TRIGGER
- ④ INSERT 0 1 , 情報所得結果は以下。  
(処理に失敗せず、memo、あるいはname欄が空である事)

| ④ tbl_history              |                                                     |
|----------------------------|-----------------------------------------------------|
| ope_time                   | memo                                                |
| 2024-07-12 11:40:11.454329 | simple1                                             |
| 2024-07-15 02:17:45.431987 | INSERT, person, 2, BX, BBB, memo                    |
| 2024-07-15 02:17:50.752227 | UPDATE, person, 2, BX, BBB, memo2, 2, BX, BBB, memo |
| 2024-07-15 02:18:25.616389 | DELETE, person, 2, BX, BBB, memo2                   |
| 2024-07-15 02:28:58.28816  |                                                     |
| 2024-07-15 03:06:12.484725 |                                                     |

| ④ tbl_history2 |      |
|----------------|------|
| id             | name |
| 4              |      |

\* 補足あり。回答後、確認推奨。

### ■ ここまでテーブル状態

## ここまで状況

ここまで完了までのテーブル構成は以下となっている

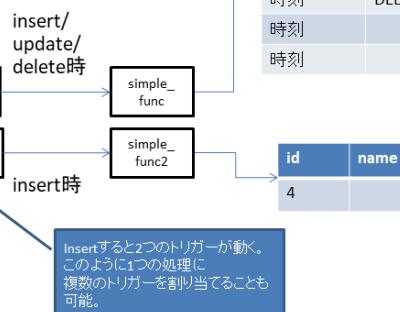
#### 【dept】

| dept_id | name | psn_cnt |
|---------|------|---------|
| 1       | AX   |         |
| 2       | BX   |         |

| ope_time | memo     |
|----------|----------|
| 時刻       | simple1  |
| 時刻       | INSERT ~ |
| 時刻       | UPDATE ~ |
| 時刻       | DELETE ~ |
| 時刻       |          |

#### 【person】

| id | dept | name | memo         |
|----|------|------|--------------|
| 1  | AX   | AAA  | memo2        |
| 3  | AX   | CCC  |              |
| 4  |      |      | Simple_func2 |



### ● Q142 関数の行ごと処理とSQL文ごと処理の違い

#### 【問題】

- ① 処理が実行されるとtbl\_history(ope\_time,memo)に「現時刻, do SQL now」を挿入する関数(do\_sql\_func)を定義する。
- ② personテーブルの更新時にSQL文ごとに実行時にされるトリガー(upt\_sql\_tg)を用意する。  
呼び出す関数は①とする。
- ③ personテーブルのAXを全て「AX2」とする。  
その後、tbl\_historyを確認し、2つのトリガー(simple\_func\_tg(行ごと)と、upt\_sql\_tg(SQL文ごと))が動いた事を確認する。  
[advance] なお、行ごと処理の2つ目のmemo欄が空欄である理由も考えてみましょう。

#### 【期待出力】

- ① CREATE FUNCTION
- ② CREATE TRIGGER
- ③ UPDATE 2 , tbl\_historyの結果は以下。

③ 該当箇所のみ抜粋。  
updateで2レコードが更新された為、行ごと実行は2行である。(赤箇所)  
SQL文ごとの実施は何レコード更新されようが、1回だけ実行の為、1行のみ。(緑箇所)

| ope_time                   |  | memo                                         |
|----------------------------|--|----------------------------------------------|
| 2024-07-15 09:12:54.738476 |  | UPDATE,person,1,AX2,AAA,memo2,1,AX,AAA,memo2 |
| 2024-07-15 09:12:54.738476 |  | 2024-07-15 09:12:54.738476   do SQL now      |

### ● Q143 登録したい情報空欄化 一次改善

#### 【問題】

前問題-③の空欄問題の原因を取り除く事で正しく動くか確認を行う。

- ① personテーブルのidが3のレコードのmemoに「add\_memo」という値を入れる。
- ② personテーブルにて、deptがAX2のレコードのdeptをAX3に更新する。  
(2レコード更新)
- ③ tbl\_historyを確認し、memoが空欄となる事が解消されたか確認を行う

#### 【期待出力】

- ① UPDATE 1
- ② UPDATE 2

③ 該当箇所のみ抜粋。  
②での更新は前問題-③と同様、idが1と3のレコードの更新だが、  
今回は行ごと更新履歴の2つ目も値が入っている(赤字箇所)

| ope_time                  |  | memo                                                                            |
|---------------------------|--|---------------------------------------------------------------------------------|
| 2024-07-15 09:25:26.59265 |  | UPDATE,person,1,AX3,AAA,memo2,1,AX2,AAA,memo2                                   |
| 2024-07-15 09:25:26.59265 |  | 2024-07-15 09:25:26.59265   UPDATE,person,3,AX3,CCC,add_memo,3,AX2,CCC,add_memo |
| 2024-07-15 09:25:26.59265 |  | 2024-07-15 09:25:26.59265   do SQL now                                          |

### ● Q144 登録したい情報空欄化 恒久改善

【問題】

[advice] 2問前の問題-③の空欄問題について、memoが空でも問題なく表示させるように変更する。

- ① 2問目の問題-③と同じ状態を作る。Id 3のレコードのmemoをnullに変更する。
- ② ①実施によりトリガーが動くはず。tbl\_historyをみて事象再現しているか確認する。
- ③ personテーブルのmemoが空でもtbl\_historyのmemoが正しく表示されるように改善する。  
更新時には更新対象のmemoが空である場合(old)と、更新時にnullとする場合(new)の両方を考慮する事。  
※ 変更するのはupdate時のmemoが空の場合のみで良い。  
(insertやdeleteの対応、memo以外の対応は不要)
- ④ personテーブルのdeptがAX3のものを、AX4に変更する。(2レコード更新)  
またtbl\_historyの全レコードを取得し、③の対処が効いているか確認する。

【期待出力】

- ① UPDATE 1
- ③ ヒント参照。
- ④ UPDATE 2。tbl\_historyの結果は以下。

| ② 該当箇所のみ抜粋。                                                                    |            |
|--------------------------------------------------------------------------------|------------|
| 今回はupdate1の為、行ごと実行の履歴も1レコードのみ。そのレコードはmemoがnullとなる為、空となっている。(赤字箇所)              |            |
| ope_time                                                                       | memo       |
| <br>2024-07-15 09:34:59.472005                                                 |            |
| 2024-07-15 09:34:59.472005                                                     | do SQL now |
| ④ 該当箇所のみ抜粋。                                                                    |            |
| 今回はupdateするレコードのうち、1レコードはmemoがnullだが正しく値が入っている(赤字箇所)                           |            |
| ope_time                                                                       | memo       |
| <br>2024-07-15 09:41:56.718958   UPDATE,person,1,AX4,AAA,memo2,1,AX3,AAA,memo2 |            |
| 2024-07-15 09:41:56.718958   UPDATE,person,3,AX4,CCC,,3,AX3,CCC,               |            |
| 2024-07-15 09:41:56.718958   do SQL now                                        |            |

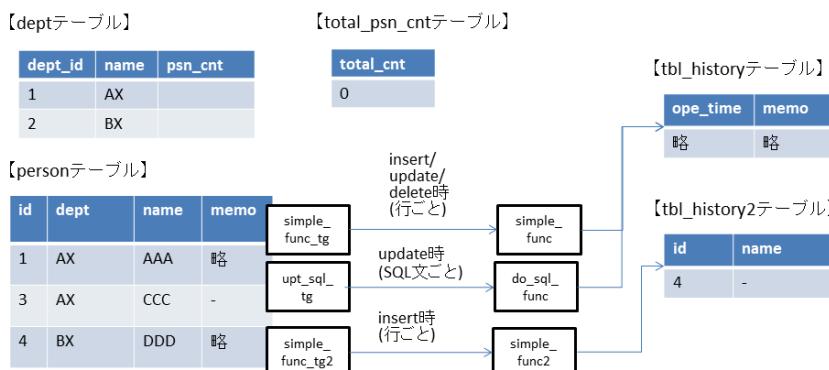
### ■ 追加準備・ここまででのテーブル状態

#### 事前準備2、ここまで完了後のテーブル状態

次問以降に向けて一度環境を再整理する、  
以下を実行すること。

```
update person set dept = 'BX', name = 'DDD' where id = 4 ;
update person set dept = 'AX' where dept = 'AX4' ;
create table total_psn_cnt(total_cnt int);
insert into total_psn_cnt values (0);
```

上記変更後のテーブル状態は以下のようないmage。



### ● Q145 特定カラムが処理された場合のトリガー

【問題】

[advance] ① do\_sql\_funcを更新する。

personテーブルのdept列が更新された場合のみ呼び出されることを考慮し、  
更新後の部署の人数をdeptテーブルのpsn\_cntに登録する。

[advance] ② personテーブルのdept(部署名)が更新された場合のみdo\_sql\_funcが  
呼び出されるようにする

なお、tbl\_history2への更新トリガー(upt\_sql\_tg)は削除するが、トリガーネームは  
そのままとする

※ 行ごと/oracle文ごとかは、より向いていると思われる方を設定する

③ ①②での設定が期待通りか動作確認をする。

personテーブルのdeptがAXのレコードを、BXに変更する。

その後、deptテーブルの全レコードを取得する。

④ idが1のレコードのみ、deptを「AX2」とする。その後、deptテーブルの全レコードを取得する。

【期待出力】

① CREATE FUNCTION

② DROP TRIGGER

CREATE TRIGGER

③ UPDATE 2, deptテーブルの結果は以下

④ UPDATE 1, deptテーブルの結果は以下。

③ 正しく人数を集計できている  
dept\_id | name | psn\_cnt

|   |    |   |
|---|----|---|
| 1 | AX | 0 |
| 2 | BX | 3 |

④ person側で、deptテーブルに存在しない部署名を登録したこと。  
AX2分のデータが欠損している  
dept\_id | name | psn\_cnt

|   |    |   |
|---|----|---|
| 1 | AX | 0 |
| 2 | BX | 2 |

### ● Q146 SQL文ごと実施関数の集計処理

【問題】

① total\_psn\_cntテーブルのtotal\_cntにpersonテーブルの人数を入れる

関数(sum\_psn\_func)を作成する。

また、personテーブルがinsert/delete/truncateしたときに上記関数を呼び出す、  
SQL文ごと実施トリガー(sum\_psn\_tg)を設定する。

② ①の動確を行う。personテーブル(id, dept, name)に「5,BX,EEE」を登録する。

その後、total\_psn\_cntテーブルを確認する。

③ ①の動確を行う。personテーブルからidが5のレコードを削除する。

その後、total\_psn\_cntテーブルを確認する。

【期待出力】

① CREATE FUNCTION

CREATE TRIGGER

② INSERT 0 1, total\_psn\_cntテーブルの結果は以下

③ DELETE 1, total\_psn\_cntテーブルの結果は以下

②  
total\_cnt  
-----  
4

④  
total\_cnt  
-----  
3

## ● Q147 テーブル間のズレの防止

【問題】

[advance] 前問題でテーブル間の情報に不整合が生じた。  
このような事がないようにする為に、新たなトリガー・ファンクションを設定する。

- ① まずは不整合を直す。deptテーブルに登録が漏れている「dept\_id, name」が「3, AX2」の情報を登録する。
- ② deptのnameを新しい値に更新したら、personの該当部署に所属している人の部署名(dept)も更新する。  
ファンクション(upt\_dept\_name)を設定する。  
また上記を実行するトリガー(upt\_dept\_name\_tg)も設定する。
- ③ ②で設定した値が期待通り動くか確認する。  
deptテーブルのnameがAX2のレコードを、AX3とする。  
その後、personテーブルの確認をする。
- ④ personテーブルのdept(部署名)を変更する際に、deptテーブルのname(ないもの)を指定した場合はエラーとするファンクション(upt\_person\_dept)を作成し、それを実行するトリガー(upt\_dept\_name\_tg)を作成する。  
処理失敗時のエラー文は「tried to change dept column, but does not exist dept table」とする。
- ⑤ ④の動確を行う。deptテーブルに存在しないAX4を、personテーブルのAX3レコードで更新しようとする。(失敗する)
- ⑥ personテーブルのid3のレコード(dept: BX)にて、deptテーブルに存在するAX3に更新する(成功する)  
その後、personテーブル、deptテーブルのレコード確認を行う。

【期待出力】

```
① INSERT 0 1
②④ CREATE FUNCTION
 CREATE TRIGGER
③ UPDATE 1, personテーブルの結果は以下
⑤ ERROR: tried to change dept column, but does not exist dept table
CONTEXT: PL/pgSQL function upt_person_dept() line 4 at RAISE
⑥ UPDATE 1, dept, personテーブルの結果は以下。
```

| ③ 正しく人数を集計している              | ④ personテーブル                | ⑤ deptテーブル               |
|-----------------------------|-----------------------------|--------------------------|
| id   dept   name   memo     | id   dept   name   memo     | dept_id   name   psn_cnt |
| -----                       | -----                       | -----                    |
| 3   BX   CCC                | 1   AX3   AAA   memo2       | 1   AX   0               |
| 1   AX3   AAA   memo2       | 3   AX3   CCC               | 2   BX   1               |
| 4   BX   DDD   simple_func2 | 4   BX   DDD   simple_func2 | 3   AX3   2              |

\* 補足あり。回答後、確認推奨。

## ● Q148 やや複雑な入力値チェック

【問題】

deptテーブルのpsn\_cnt(部署ごとの人数)をやや特殊な使い方をしたい。  
基本は0以上の値が入るはずだが、-1となった場合は論理削除扱いとしたい。  
この運用とする理由は、部署名を変更する際に過去に使った部署名を使いたくない為。  
ただし論理削除する時にその部署に人がいると困る。  
よって、psn\_cntが0の時のみ、-1を登録できることとし、1以上の数値の場合は-1を設定できない事とする。  
なお、psn\_cntは本来の意図(部署ごと人数集計)でも使う為、正の数値は問題なく入る事、  
また-2以下は入らない事とする。

- ① 上記を満たす、入力値チェック関数(psn\_cnt\_sql\_check\_func)を作成する。  
psn\_cntが0以外の時に-1に変更しようとした際には「tried to put -1 on psn\_cnt, but now it's not 0」というエラーを出す。  
また、-2以下を登録しようとした際には「tried to put less than -1」というエラーを出す。

あわせて上記関数を起動するトリガー(psn\_cnt\_check\_tg)を作成する。  
 ② ①の動確をする。BXは現在psn\_cntが1のはずである。この状態でBXのpsn\_cntを-1を設定しようとする。(失敗する)  
 ③ ①の動確をする。AXは現在psn\_cntが0のはずである。この状態で-2を入れようとする。(失敗する)  
 ④ ①の動確をする。AXは現在psn\_cntが0のはずである。この状態で-1を入れようする。  
 またAX3は現在、psn\_cntが2のはずである。この状態で10を入れようとする。  
 上記実施後、deptテーブルを確認する。

【期待出力】

```
① CREATE FUNCTION
 CREATE TRIGGER
② ERROR: tried to put -1 on psn_cnt, but now it's not 0
CONTEXT: PL/pgSQL function psn_cnt_sql_check_func() line 4 at RAISE
③ ERROR: tried to put less than -1
CONTEXT: PL/pgSQL function psn_cnt_sql_check_func() line 8 at RAISE
④ UPDATE 1 × 2. Deptテーブルの結果は右の黒枠参照。
```

| ④                        |
|--------------------------|
| dept_id   name   psn_cnt |
| -----                    |
| 2   BX   1               |
| 1   AX   -1              |
| 3   AX3   10             |

\* 補足あり。回答後、確認推奨。

## ■ 追加準備

### 事前準備3

次問に向けて環境を再整理する。以下を実行すること。

```
create table contact(person_id int, email text);
```

```
insert into contact values (3, 'ccc@example.com');
```

```
create view simple_person as select id , memo from person ;
```

```
create view person_contact as select p.id, p.name , c.email from person
join contact c on p.id = c.person_id ;
```

上記変更後のテーブル状態は以下のようないmage。

【personテーブル】

| <b>id</b> | <b>dept</b> | <b>name</b> | <b>memo</b> |
|-----------|-------------|-------------|-------------|
| 1         | AX          | AAA         | 略           |
| 3         | AX          | CCC         | -           |
| 4         | BX          | DDD         | 略           |

【contactテーブル】

| <b>person_id</b> | <b>email</b>    |
|------------------|-----------------|
| 3                | ccc@example.com |

【simple\_personビュー】

| <b>id</b> | <b>memo</b> |
|-----------|-------------|
| 1         | 略           |
| 3         | -           |
| 4         | 略           |

【person\_contactビュー】

| <b>id</b> | <b>name</b> | <b>email</b>    |
|-----------|-------------|-----------------|
| 3         | CCC         | ccc@example.com |

### ● Q149 トリガーファンクションによる処理代替

【問題】

① person\_contactビューに対し、「5,EEE,eee@example.com」を登録しようとする(失敗)

② ①を実行できるようにするための関数(ins\_psn\_con)を作成する。

③ ②でいう5,EEEはpersonテーブルのid、nameに、5,eee@example.comはcontactのperson\_idとemailにデータ登録するようにする。

[advance] ④ ②を実行するトリガー(ins\_psn\_con\_tg)を設定する。行ごと実行とし、insertされた時に代替処理が起動するような設定とする。

トリガー対象は誤ってpersonテーブルに設定する。(失敗)

[advance] ⑤ ④について正しい設定をする。

⑥ ①のinsert処理を行う。

その後、person\_contactビュー、personテーブル、contactテーブルの情報を取得する。

【期待出力】

① ERROR: cannot insert into view "person\_contact"

DETAIL: Views that do not select from a single table or view are not automatically updatable.

HINT: To enable inserting into the view, provide an INSTEAD OF INSERT trigger or an unconditional ON INSERT DO INSTEAD rule.

② CREATE FUNCTION

③ ERROR: "person" is a table

DETAIL: Tables cannot have INSTEAD OF triggers.

④ CREATE TRIGGER

⑤ INSERT 0 1。各テーブル/ビューの情報は以下。

| ⑤ person_contact |             |                 |  |
|------------------|-------------|-----------------|--|
| <b>id</b>        | <b>name</b> | <b>email</b>    |  |
| 3                | CCC         | ccc@example.com |  |
| 5                | EEE         | eee@example.com |  |

| ⑤ person  |             |             |              |
|-----------|-------------|-------------|--------------|
| <b>id</b> | <b>dept</b> | <b>name</b> | <b>memo</b>  |
| 1         | AX3         | AAA         | memo2        |
| 3         | AX3         | CCC         |              |
| 5         |             | EEE         |              |
| 4         | BX          | DDD         | simple_func2 |

| ⑤ contact        |                 |
|------------------|-----------------|
| <b>person_id</b> | <b>email</b>    |
| 3                | ccc@example.com |
| 5                | eee@example.com |

## ● Q150 ルールによる代替処理

## 【問題】

- ① simple\_personビューに対し、「6,memo」を登録しようとする。
- ② simple\_personビューに対するinsert/update/deleteはされたくない。これらを実施された場合、自動でselectされるルール(ins/upd/del\_simple\_psn\_rule)を設定する
- ③ total\_psn\_cntテーブルに対するinsertも、select(切り替えるようなルール(ins\_total\_psn\_rule)を設定する
- ④ 作ったins\_total\_psn\_ruleの詳細を確認する
- [advance] ⑤ simple\_personビューに対して設定したルールの詳細を確認する
- ⑥ ②③の動確をする。simple\_personに「7,memo」を登録しようとする。またtotal\_psn\_cntに「10」を登録しようとする。

## 【期待出力】

- ① INSERT O 1  
②③ CREATE RULE

| ④ schemaname | tablename     | rulename            | definition                                                                                                                                                              |
|--------------|---------------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public       | total_psn_cnt | ins_total_psn_rule  | CREATE RULE ins_total_psn_rule AS +<br>  ON INSERT TO public.total_psn_cnt DO INSTEAD SELECT total_psn_cnt.total_cnt+<br>    FROM total_psn_cnt;                        |
| ⑤            | schemaname    | tablename           | rulename definition                                                                                                                                                     |
| public       | simple_person | del_simple_psn_rule | CREATE RULE del_simple_psn_rule AS +<br>  ON DELETE TO public.simple_person DO INSTEAD SELECT simple_person.id,+<br>    simple_person.memo +<br>    FROM simple_person; |
| public       | simple_person | ins_simple_psn_rule | CREATE RULE ins_simple_psn_rule AS +<br>  ON INSERT TO public.simple_person DO INSTEAD SELECT simple_person.id,+<br>    simple_person.memo +<br>    FROM simple_person; |
| public       | simple_person | upd_simple_psn_rule | CREATE RULE upd_simple_psn_rule AS +<br>  ON UPDATE TO public.simple_person DO INSTEAD SELECT simple_person.id,+<br>    simple_person.memo +<br>    FROM simple_person; |

|                 |              |
|-----------------|--------------|
| ⑥ simple_person |              |
| id              | memo         |
| 1               | memo2        |
| 3               | 5            |
| 6               | memo         |
| 4               | simple_func2 |

|            |
|------------|
| INSERT O 0 |
|------------|

|                 |
|-----------------|
| ⑦ total_psn_cnt |
| total_cnt       |
| 5               |

|            |
|------------|
| INSERT O 0 |
|------------|

## ● Q151 ルールによる追加処理について

## 【問題】

- ① total\_psn\_cntは1レコードのみで運用する。deleteされた時はinsertも実施し、必ず1レコードとなるようにするルール(del\_total\_psn\_rule)を作る。
- ② total\_psn\_cntに対しレコード削除を実施し、その後、当テーブルの情報を取得する。(想定通り動かない)
- ③ ②で期待通り動かなかった原因と想定される対応を行う。  
その後、total\_psn\_cntにinsertにてを入れる。更にその後、レコード削除を行い、その後に該当テーブルの情報を取得する。(期待通り動かない)
- ④ ③でも上手く動かない理由は①で設定したルールによるものと考えた。  
具体的にはルール起動の要因となるSQL(delete)と、alsoで指定した追加SQL(insert)の起動タイミングによるものと考えた。  
つまり起因SQL(delete)してから追加SQL(insert)がされるのではなく、追加SQL(insert)されてから起因SQL(delete)が動いていると想像した。
- よって正しい動きのルールを作る為に、起因SQLをinsert、追加SQLをdeleteとするルール(ins\_total\_psn\_rule)を作成する。(追加SQL(delete)が動いてから起因SQL(insert)が動く事を期待する)
- なお、上記を作る前に期待通り動かない①のルール(del\_total\_psn\_rule)は削除し、total\_psn\_cntも現在0レコードである為、1レコードとする為、100という数値を登録する。

[advance] ⑤ total\_psn\_cntに追加レコードを登録し、その後当テーブルの情報を取得する。(期待通り動かない)

期待通り動かない理由はなぜか。

- ⑥ ins\_total\_psn\_ruleを削除し、total\_psn\_cntに1レコード追加する。  
その後、total\_psn\_cntに更新が入った場合、tbl\_history(にも)情報(現在時刻,"old\_cnt: 値, new\_cnt: 値")をいれるルール(upd\_total\_psn\_rule)を作成する。

⑦ ⑥の動確をする。total\_psn\_cntのレコードを100に更新する。その後、total\_psn\_cntとtbl\_historyの情報を確認する。

## 【期待出力】

- ① CREATE RULE  
② DROP RULE, INSERT O 1, DELETE 2,  
データ取得時の結果は右参照  
④ DROP RULE, INSERT O 1, CREATE RULE  
⑤ INSERT O 1,データ取得時の結果は右参照  
⑥ DROP RULE, INSERT O 1, CREATE RULE  
⑦ UPDATE 1

|             |
|-------------|
| ② total_cnt |
| -----       |
| 5           |
| total_cnt   |
| (0 rows)    |

|             |
|-------------|
| ③ total_cnt |
| -----       |
| (0 rows)    |

|             |
|-------------|
| ⑤ total_cnt |
| -----       |
| 100         |

|                 |
|-----------------|
| ⑦ total_psn_cnt |
| -----           |

|                            |                       |
|----------------------------|-----------------------|
| ⑦ ope_time                 | memo                  |
| 2024-07-16 18:37:32.080149 | old_cnt:5 new_cnt:100 |

\* 補足あり。回答後、確認推奨。

### ● Q152 トリガー、ルール確認に便利なメタコマンド

#### 【問題】

- [advance] ① personテーブルに付与されたトリガーリストを確認する  
 [advance] ② simple\_personビューに付与されたルールの一覧を確認する

#### 【期待出力】

| ② View "public.simple_person"                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |         |           |          |          |             |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|-----------|----------|----------|-------------|
| Column                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Type    | Collation | Nullable | Default  | Description |
| id                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | integer |           |          | plain    |             |
| memo                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | text    |           |          | extended |             |
| View definition:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |         |           |          |          |             |
| SELECT person.id,<br>person.memo<br>FROM person;                                                                                                                                                                                                                                                                                                                                                                                                                                                           |         |           |          |          |             |
| Rules:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |         |           |          |          |             |
| del_simple_psn_rule AS<br>ON DELETE TO simple_person DO INSTEAD SELECT simple_person.id,<br>simple_person.memo<br>FROM simple_person                                                                                                                                                                                                                                                                                                                                                                       |         |           |          |          |             |
| ins_simple_psn_rule AS<br>ON INSERT TO simple_person DO INSTEAD SELECT simple_person.id,<br>simple_person.memo<br>FROM simple_person                                                                                                                                                                                                                                                                                                                                                                       |         |           |          |          |             |
| upt_simple_psn_rule AS<br>ON UPDATE TO simple_person DO INSTEAD SELECT simple_person.id,<br>simple_person.memo<br>FROM simple_person                                                                                                                                                                                                                                                                                                                                                                       |         |           |          |          |             |
| Triggers:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |         |           |          |          |             |
| simple_func_tg AFTER INSERT OR DELETE OR UPDATE ON person FOR EACH ROW EXECUTE FUNCTION simple_func()<br>simple_func_tg2 AFTER INSERT ON person FOR EACH ROW EXECUTE FUNCTION simple_func2()<br>sum_psn_tg AFTER INSERT OR DELETE OR TRUNCATE ON person FOR EACH STATEMENT EXECUTE FUNCTION<br>sum_psn_func()<br>upt_dept_name_tg BEFORE UPDATE OF dept ON person FOR EACH ROW EXECUTE FUNCTION upt_person_dept()<br>upt_sql_tg AFTER UPDATE OF dept ON person FOR EACH ROW EXECUTE FUNCTION do_sql_func() |         |           |          |          |             |

| ① Table "public.person" |         |           |          |
|-------------------------|---------|-----------|----------|
| Column                  | Type    | Collation | Nullable |
| id                      | integer |           |          |
| dept                    | text    |           |          |
| name                    | text    |           |          |
| memo                    | text    |           |          |

|                                                                                                               |
|---------------------------------------------------------------------------------------------------------------|
| Triggers:                                                                                                     |
| simple_func_tg AFTER INSERT OR DELETE OR UPDATE ON person FOR EACH ROW EXECUTE FUNCTION simple_func()         |
| simple_func_tg2 AFTER INSERT ON person FOR EACH ROW EXECUTE FUNCTION simple_func2()                           |
| sum_psn_tg AFTER INSERT OR DELETE OR TRUNCATE ON person FOR EACH STATEMENT EXECUTE FUNCTION<br>sum_psn_func() |
| upt_dept_name_tg BEFORE UPDATE OF dept ON person FOR EACH ROW EXECUTE FUNCTION upt_person_dept()              |
| upt_sql_tg AFTER UPDATE OF dept ON person FOR EACH ROW EXECUTE FUNCTION do_sql_func()                         |

### ■ 追加準備

#### 事前準備4

次間に向けて環境を再整理する。以下を実行すること。

```
delete from person where id > 4;
update person set memo = 2018 where id = 1;
update person set memo = 2019 where id = 3;
update person set memo = 2020 where id = 4;
```

```
create table salary (person_id int , sal int);
insert into salary(psn_id) values (1), (3), (4);
```

上記変更後のテーブル状態は以下のようないmage。

【personテーブル】

| id | dept | name | memo |
|----|------|------|------|
| 1  | AX   | AAA  | 2018 |
| 3  | AX   | CCC  | 2019 |
| 4  | BX   | DDD  | 2020 |

【salaryテーブル】

| psn_id | sal  |
|--------|------|
| 1      | null |
| 3      | null |
| 4      | null |

### ● Q153 ストアドファンクション1

#### 【問題】

[advance] ① 以下の条件で、与えられたperson\_idに沿って、  
 その人の給与を計算する関数(get\_sal)を作成する。  
 条件は以下である。  
     • personのmemoに入社年が記載されている。  
     1年ごとにベース賃金にプラスで1000円加算される。  
     (勤続年数が2年ならベース賃金に2000円上乗せ)  
     • 基本給は20万とする  
 ② ①の関数を使い、personのidが1の人の給与を算出する。

#### 【期待出力】

① CREATE FUNCTION

②

get\_sal

-----

206000

※現在2024年。2024 - 2018で6年。1年ごとに1000円上乗せ、  
 基本給20万なので20万6000円となる。

年が進んだ場合、それに応じて金額が変動するので注意。  
 適切な年数にあてはめて計算が合っている確認する事。

### ● Q154 ストアドファンクション2

#### 【問題】

[advance] ① 前問題で作った関数を活用し、personに登録されている  
 全ユーザの給与を一括で計算する  
 関数(get\_all\_sal)を作る。

② ①の動作確認を行う。

#### 【期待出力】

① CREATE FUNCTION

②

get\_all\_sal

-----

(1,206000)

(3,205000)

(4,204000)

### ● Q155 ストアドファンクション3

#### 【問題】

- ① 2問前に作った関数を使い、全ユーザの給与をsalaryテーブルのsalに更新する関数(set\_all\_sal)を作成する。  
なお、処理が問題なく完了したら「update complete!!」と出力されるようにする。
- ② ①の関数の動確を行う。
- ③ salaryテーブルを確認し、全ユーザの給与が反映されているか確認する。

#### 【期待出力】

① CREATE FUNCTION

②

set\_all\_sal

-----

update complete!!

③

psn\_id | sal

-----+-----

1 | 206000

3 | 205000

4 | 204000

※ id 1, 3, 4で1年ずつpersonのmemoに入れた数値は異なる。よって1000円ずつズレる。

### ● Q156 ストアドプロシージャ

#### 【問題】

- [advance] ①これまでの問題でpersonテーブルとsalaryテーブルは常に同じidのレコードがないと困る。  
(全ユーザの給与を算出しようと/oracle)も、salaryテーブルに該当ユーザのidのレコードがないと算出が漏れる  
よってpersonテーブルに登録した際(これはsalaryテーブルにもpersonと同じidのレコードを  
登録する(salは計算不要)プロシージャ(add\_person)を作る。  
ただし人が複数人増える可能性もある為、同時に複数人登録できる事に対応していること。  
また、deptテーブルにある部署以外の人を登録しようとすると、  
その人は登録されないようにする(rollback)こと。  
(dept)にある部署のユーザのみ登録される(commit)ようにする)
- ② ①で作ったプロシージャの情報を取得する
- [advance] ③ ①で作ったプロシージャの動確を行う。Id [5, 6, 7], dept [ BX, CX, BX ], name[ EEE ,FFF, HHH ]を  
①プロシージャで登録する。
- ④ personテーブルの一覧を取得する。
- ⑤ ①のプロシージャを削除する。

#### 【期待出力】

① CREATE PROCEDURE

③ CALL

⑤ DROP PROCEDURE

④

|                         |
|-------------------------|
| id   dept   name   memo |
| 1   AX3   AAA   2018    |
| 3   AX3   CCC   2019    |
| 4   BX   DDD   2020     |
| 5   BX   EEE            |
| 7   BX   HHH            |

※存在しないCXを入れようとした  
FFFは登録されていない

②一部抜粋(¥xでの表示)

```
-[RECORD 1]-----+
proname | add_person
prosrc |
| begin
| for i in 1..array_length(p_id,1) loop
| if exists (select * from dept where name = p_dept[i]) then
| insert into person(id , dept , name) values (p_id[i], p_dept[i], p_name[i]);
| insert into salary(psn_id) values (p_id[i]);
| commit ;
| else
| rollback ;
| end if;
| end loop;
| end ;
```

## 7章 +α問題

### その 1:PL/pgSQL や自動処理

#### ■ 事前準備

#### 事前準備

以下のSQLをpostgres DB , postgresユーザで繋いで実行する。

```
create database test07a;
create table dept (dept_id int, name text, psn_cnt int);
create table person (id int , dept text , name text , memo text);
create table tbl_history (ope_time timestamp, memo text);

insert into dept(dept_id,name) values (1, 'AX'),(2, 'BX');
```

以下のような表が出来上がったイメージ。  
社内管理システムを想定し、deptは部署名、personは従業員、tbl\_historyはテーブル操作の履歴をとるテーブルとする。

【deptテーブル】

| dept_id | name | psn_cnt |
|---------|------|---------|
| 1       | AX   |         |
| 2       | BX   |         |

【personテーブル】

| id | dept | name | memo |
|----|------|------|------|
|    |      |      |      |
|    |      |      |      |
|    |      |      |      |
|    |      |      |      |

【tbl\_historyテーブル】

| ope_time | memo |
|----------|------|
|          |      |

#### ● Q157 ルールの処理実行順序 1

##### 【問題】

- ① insert処理を起動条件とするルール(also)の実行順序を確認したい。  
tbl\_historyにinsertされた場合に「now(), also!!」と入るようなルール(ins\_test\_rule)を作成する。
- ② ①の動確をする。tbl\_historyテーブルに「now(), SQL!!」を入れようとする。(失敗)  
なぜ失敗したかも併せて考える。
- ③ ①のルールを削除する

##### 【期待出力】

- ① INSERT 0 1
- ② ERROR: infinite recursion detected in rules for relation "tbl\_history"
- ③ DROP RULE

### ● Q158 ルールの処理実行順序 2

## 【問題】

- ① 前問題のように無限ループにならないように確認が必要。  
よって別テーブルにinsertする形にし、登録時に入れられる時間情報をみて、ルール起動要因のSQLと、alsoによる追加SQLのどちらが先に動いてたかチェックしたい。  
deptに情報を入れたら、tbl\_historyにclock\_timestamp(), alsoを入れるルール(ins\_test\_rule)を作成する。
- ② ①の動確をする。  
deptに「6, clock\_timestamp(), 1」を入れる。  
ただし本SQL実行に5秒かかるようにsleepをかける。  
実行後、deptとtbl\_historyをチェックする
- ③ ①のルールを削除する。

## 【期待出力】

- ① CREATE RULE
- ② 時間をチェックし入力SQLが先に動いて、alsoでの追加SQLが後に実行された事がわかる
- ③ DROP RULE

| ② dept  |                               |         | ② tbl_history              |        |  |
|---------|-------------------------------|---------|----------------------------|--------|--|
| dept_id | name                          | psn_cnt | ope_time                   | memo   |  |
| 6       | 2024-07-20 01:58:55.231287+00 | 1       | 2024-07-20 01:59:00.236515 | also!! |  |

ルールはdeptにデータ登録しようとした②の処理で起動する。  
その際には先に入力SQLを登録し、その後にalsoで指定したSQL(tbl\_history)に処理を入れるはず。  
その際、入力SQLでは5秒程度かかるように仕掛けをいれている。  
よって、この2つの「入力時間」に5秒程度の差があればよい。

### ● Q159 ルールの処理実行順序 3

## 【問題】

- ① 前問題同様に、ルールの起動順序を確認する。今回はdeleteを起因とするルールの確認をする。  
deptテーブルに対しdeleteすると、tbl\_historyテーブルに「clock\_timestamp, also!!」と入れる形のルール(del\_test\_rule)を作成する。
- ② ①の動確をする。  
deptにてidが6の部署を削除する。  
delete処理を起因とするルールの場合は、先にalsoで指定したSQLが起動し、その後にdeleteで該当レコードが消されるはず。  
レコードが削除される時間が記録に残せない為、ルール処理含めて処理完了後、即座にinsert処理でtbl\_historyに「clock\_timestamp, del complete!!」と入れる事で確認を行う。  
上記実施後、tbl\_historyの中身をみて、先にSQLで指定したinsertが実行され、その後、deleteされた事がわかるようになる。  
なお、delete処理も5秒程度時間を要する仕組みを入れておくこと。
- ③ ①のルールを削除する。

## 【期待出力】

- ① CREATE RULE
- ② DELETE 1  
INSERT O 1  
tbl\_historyの結果は右参照。
- ③ DROP RULE

先にinsert処理(alsoによる追加SQL)が実行されたとわかる。  
それを実施後、5秒かかるdeleteを実施、その後即座にinsertを入れているが、  
大体5秒のズレがある。(間に5秒待ちのdelete処理が挟まっていることがわかる)

## その2:更に色々試してみたい人向け

---

### ■ 事前準備

#### 事前準備

新たなDB(test\_ad)を使う。  
基本はDML(基礎)で使ったテーブルと同様の構成だが、以下の専用のテーブルを新たに作成する。

```
create database test07b;
psql -U postgres -d test07b
create table dept (dept_id int, name text);
create table users (id int, dept_id int, name text, grade numeric);
create table device (device_id int primary key, user_id int, status boolean, create_at timestamp default current_timestamp);
insert into dept values (1, 'AX'), (2, 'BX'), (3, 'CX'), (4, 'AX');
insert into users values (1, 1, 'AAA', 3.5), (2, 1, 'BBB', 2.5), (3, 2, 'CDef', 3.2), (4, 3, 'cdEF', 4.5);
insert into device values (1, 1, true);

create table student (name text, english int, math int, science int);
create table history (person text, action text, timestamp timestamp);
create table grade (student_id int, grade int);
insert into grade values (1, 80), (2, 63), (3, -1), (4, -1), (5, 96);
```

専用テーブルは以下のような表が出来上がったイメージ。  
正しく出来上がっているか、selectで確認しておくこと

【studentテーブル】

| name | english | math | Science |
|------|---------|------|---------|
| AAA  | 45      | 15   | 51      |

【gradeテーブル】

| student_id | grade |
|------------|-------|
| 1          | 80    |
| 2          | 63    |
| 3          | -1    |
| 4          | -1    |
| 5          | 96    |

【historyテーブル】

| person | action | timestamp                  |
|--------|--------|----------------------------|
| AAA    | call   | 2021-10-06 21:19:37.732217 |

-1は追試が必要で  
追試結果待ちというステータス

\* DBごと変えるので注意

---

### ● Q160 generate\_seriesを使った試験データ投入

#### 【問題】

- ①データとして一部データが重複しない値が望ましい場合(例:ユーザ名)を想定した。  
試験データ投入を想定した演習を実施する。  
studentテーブルに対し、なるべくランダムな情報を投入する。  
条件は以下とする。
  - ・nameはusersテーブルに入っている名前[番号]で作成する(生徒名を記載)
  - ・english～scienceは整数で数値を0～100の範囲でランダム(各教科の試験の成績を記載)
  - ・40生徒分のデータ作成を行う

- ②データとして同じような値が繰り返すものが望ましい場合(例:通話履歴などの履歴系)を想定し、  
historyテーブルにデータ投入を行う。  
以下の手順でダミーデータを作成する。
  - ・ユーザAとBのデータとB投入(Aはperson:A, action:call, timestamp:現在から365日前、  
Bはperson:B, action:replay, timestampはAと同じ)
  - ・generate\_seriesを使用し、データを作成(name+actionはA+Bの繰り返し、  
timestampは364日前から1日ごと)。
  - ※ AとBは同時刻を許容する
  - ・50履歴分作成する

【期待出力】

```
① INSERT 0 40
② INSERT 0 2
INSERT 0 50
```

【参考】①のselect文の結果

| name  | english | math | science |
|-------|---------|------|---------|
| AAA1  | 45      | 15   | 51      |
| AAA2  | 85      | 27   | 95      |
| AAA3  | 93      | 97   | 37      |
| AAA4  | 5       | 91   | 36      |
| (以下略) |         |      |         |

【参考】②のgenerate\_series前のselect文結果

| person | action | timestamp                  |
|--------|--------|----------------------------|
| A      | call   | 2021-10-06 21:19:37.732217 |
| B      | replay | 2021-10-06 21:19:37.732217 |

【参考】②のselect文結果例

| person | action | timestamp                  | create_time                  |
|--------|--------|----------------------------|------------------------------|
| A      | call   | 2021-10-06 20:25:15.192762 | 2021-10-07 21:07:37.49178+09 |
| B      | replay | 2021-10-07 20:25:29.640117 | 2021-10-07 21:07:37.49178+09 |
| A      | call   | 2021-10-06 20:25:15.192762 | 2021-10-08 21:07:37.49178+09 |
| B      | replay | 2021-10-07 20:25:29.640117 | 2021-10-08 21:07:37.49178+09 |
| A      | call   | 2021-10-06 20:25:15.192762 | 2021-10-09 21:07:37.49178+09 |
| B      | replay | 2021-10-07 20:25:29.640117 | 2021-10-09 21:07:37.49178+09 |

\* 補足あり。回答後、確認推奨。

- Q161 重複を削ったうえで、nullもカウントするには？

【問題】

- [advance] deptテーブルに対して、name行を一意に絞った上で、  
null行も含めてカウントする。  
(dept\_idを集計するやり方は除く)

【前提】

- deptテーブルは以下である事

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |

【期待出力】

|       |
|-------|
| count |
| ----- |
| 3     |
| (1 行) |

- 
- Q162 特定情報を持つ情報を除いた計算

【問題】

- grade表にて -1を除いて平均点を出力する。  
ただしwhereは指定しない事。

【期待出力】

|                    |
|--------------------|
| avg                |
| -----              |
| 79.666666666666667 |

[参考] 全体で計算した場合  
avg

|                    |
|--------------------|
| 47.400000000000000 |
| -----              |

\* 補足あり。回答後、確認推奨。

---

● Q163 レコードごとの最大値・最小値

**【問題】**

- Studentテーブルにてレコードごとにenglish, math , scienceで最も高い値、低い値を表示。

**【前提条件】**

問題「generate\_seriesを使った試験データ投入」を完了していること。  
(先に本問題を進めたい場合は上記の答えを実行後、  
本問題を実施ください)

**【期待出力】**(english～scienceはランダムな値を入れている為、  
結果の数値は変化する)

[参考]

| name  | english | math | science |
|-------|---------|------|---------|
| AAA1  | 30      | 63   | 36      |
| BBB1  | 43      | 88   | 90      |
| cdEF1 | 13      | 94   | 60      |

(以下略)

| name  | english | math | Science |
|-------|---------|------|---------|
| AAA1  | 34      | 63   | 36      |
| BBB1  | 43      | 88   | 90      |
| edEF1 | 13      | 94   | 60      |

縦はmaxやminでとれるが  
横は...?

● Q164 現在情報の取得

**【問題】**

- ① 現在のつないでいるDBの情報を取得。
- ② 現在のスキーマ情報を取得
- ③ postgresqlのバージョンを取得
- ④ 現在のロールを取得

**【期待出力】**

① current\_database

-----  
test\_ad

② current\_schema

-----  
public

※ postgres12 on Ubuntu 20.04 の場合。

③ version

-----  
PostgreSQL 12.20 (Ubuntu 12.20-0ubuntu0.20.04.1) on  
x86\_64-pc-linux-gnu, compiled by gcc (Ubuntu 9.4.0-  
1ubuntu1~20.04.2) 9.4.0, 64-bit

④ current\_role

-----  
postgres

### ● Q165 様々な index

DDL章では通常indexとマルチカラムindexを作ったが、他にもindexの作り方はある。

#### 【問題】

- ① usersテーブルのidに一意index(uni\_idx)を付与。
- ② usersテーブルのgradeにて評価が3以上のものに対しindex(part\_idx)付与。
- ③ usersテーブルのnameで大文字化したデータに対するindex(func\_idx)付与。
- ④ メタコマンドでテーブル情報を確認し、indexが付与されているか確認する。

#### 【期待出力】

- ①②③ CREATE INDEX

| ④ テーブル"public.users" |         |      |           |       |  |
|----------------------|---------|------|-----------|-------|--|
| 列                    | 型       | 照合順序 | Null 値を許容 | デフォルト |  |
| id                   | integer |      |           |       |  |
| dept_id              | integer |      |           |       |  |
| name                 | text    |      |           |       |  |
| grade                | numeric |      |           |       |  |

インデックス:

```
"uni_idx" UNIQUE, btree (id)
"func_idx" btree (upper(name))
"part_idx" btree (grade) WHERE grade >= 3::numeric
"users_idx" btree (user_id)
```

### ● Q166 role の DB 権限

DCL章ではテーブルに対する権限を付与したが、シーケンスやDBに対する付与も可能。以下で、DBに関する付与を試してみる。

#### 【問題】

- ① postgresユーザでuser01を作成(ログインとパスワード付与)
  - ② test02 DBにuser01で繋ぎ直す
  - ③ スキーマ(test\_schema02)作成を実行しようとする
  - ④ postgresqlで繋ぎ直す
  - ⑤ user01にtest02 DBに「作成」権限を付与
  - ⑥ user01で繋ぎ直す
  - ⑦ スキーマ作成(test\_schema02)を再実施
- ※ 完了後はpostgresユーザでtest02にアクセスし直しておいてください

#### 【期待出力】

- ③ ERROR: permission denied for database test02  
 ⑦ CREATE SCHEMA

● Q167 role の管理+α

**【問題】**

- ① create roleを使わずにログイン権限のあるユーザ(user03)を作成
- ② user03にパスワードaaaを付与
- ③ user03に、user01と同じ権限を付与。  
(grant createは使わない事)  
その後、user03でtest02 DBに繋ぐ。
- ④ schema\_test03をuser03にて作成する  
その後、postgresユーザでtest\_ad DBに繋ぐ。

**【期待出力】**

- ① CREATE ROLE
- ② ALTER ROLE
- ③ GRANT ROLE
- ④ CREATE SCHEMA

● Q168 ロングトランザクション排除(強制停止)

**【問題】**

- ① 端末A: トランザクション開始
- ② 端末B: 実行クエリー一覧取得し、pidを把握  
(見づらければ\zを実行)
- ③ 端末B: pidを指定してプロセス終了
- ④ 端末A: selectで情報取得
- ⑤ 端末A: ロールバック実行

**【期待出力】**

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                            |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ② \zを実行した場合の表示<br><pre>-[ RECORD 4 ]-----+ datid        65770 datname      test_ad pid          16848 usesysid     10 username     postgres application_name   psql client_addr   ::1 client_hostname   client_port    54052 backend_start   2022-10-06 10:25:18.439263+09 xact_start    2022-10-06 10:26:18.442493+09 query_start   2022-10-06 10:26:18.442493+09 state_change   2022-10-06 10:26:18.442533+09 wait_event_type   Client wait_event     ClientRead state          idle in transaction backend_xid    backend_xmin   query          begin ; backend_type   client backend</pre> | ③-[ RECORD 1 ]-----+         pg_terminate_backend   t<br><br>④ サーバとの接続が想定外にクローズされました。         おそらく要求の処理前または処理中にサーバが異常終了         したことを意味しています。         サーバへの接続が失われました。リセットしています: 成功。<br><br>⑤ WARNING: there is no transaction in progress         ROLLBACK |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

● Q169 ロングクエリの停止

【問題】

- ① 端末A: pg\_sleepで100秒待機クエリを実行
- ② 端末B: 実行クエリ一覧取得し、pidを把握  
(見づらければ\!xを実行)
- ③ 端末B: pidを指定してプロセス終了

【期待出力】

|                                                   |       |
|---------------------------------------------------|-------|
| ②-[ RECORD 4 ]-----                               | ----- |
| datid   65770                                     |       |
| dataname   test_ad                                |       |
| <b>pid</b>   52100                                |       |
| usesysid   10                                     |       |
| username   postgres                               |       |
| application_name   psql                           |       |
| client_addr   ::1                                 |       |
| client_hostname                                   |       |
| client_port   54323                               |       |
| backend_start   2022-10-06 10:26:49.062486+09     |       |
| xact_start   2022-10-06 10:42:39.41861+09         |       |
| <b>query_start</b>   2022-10-06 10:42:39.41861+09 |       |
| state_change   2022-10-06 10:42:39.418613+09      |       |
| wait_event_type   Timeout                         |       |
| wait_event   PgSleep                              |       |
| state   active                                    |       |
| backend_xid                                       |       |
| backend_xmin   1182                               |       |
| <b>query</b>   select pg_sleep(100);              |       |
| backend_type   client backend                     |       |

③ 端末B

-----

pg\_cancel\_backend | t

-----

④ 端末A

-----

ERROR: canceling statement due to user request

-----

● Q170 サイクルシーケンスの動確

DDL章ではシーケンスの番号使い切りの例を試したが、数値を使い切った。

場合に初期値に戻らせることもできる。  
→ 上記ケースの動作を確認する。

【問題】

- ① cycleオプション付きの新シーケンス(test\_seq)作成。  
条件は10から開始で、10ずつインクリメント。30で終わりとする。
- ② 上限まで使い切った後の動作確認を確認。

【期待出力】

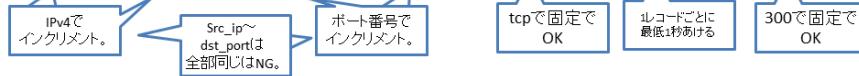
- ① CREATE SEQUENCE
- ② 3回目で30となる。4回目で1が返り、5回目で11となる。  
cycle後はstartの10は無視され、1からとなる。  
(minvalueを指定していればその数値からとなる)

### ● Q171 試験データ作成

#### 【問題】

以下の条件の検証用CSVを作成。以下の検証データがCSVでできればどんな方法でもOK。  
以下を25行作る。(直接insertやcopy不可。最終的に2500万行としたい為、SQLで実現する)

| src_ip      | dst_ip     | src_port | dst_port | proto | entry_time          | latency |
|-------------|------------|----------|----------|-------|---------------------|---------|
| 192.168.0.1 | 172.16.0.1 | 50000    | 60000    | tcp   | 2022-01-01 00:00:00 | 300     |
| 192.168.0.1 | 172.16.0.2 | 50000    | 60001    | tcp   | 2022-01-01 00:00:01 | 300     |
| ...         | ....       | ...      | ...      | ...   | ...                 | ...     |
| 192.168.0.2 |            | 50001    |          |       |                     |         |



#### 【期待出力】(例)

| id | src_ip      | dst_ip     | src_port | dst_port | proto | entry_time          | latency |
|----|-------------|------------|----------|----------|-------|---------------------|---------|
| 1  | 192.168.0.1 | 172.16.0.1 | 50000    | 60000    | TCP   | 2022-01-01 00:00:00 | 300     |
| 2  | 192.168.0.1 | 172.16.0.2 | 50000    | 60001    | TCP   | 2022-01-01 00:00:01 | 300     |
| 3  | 192.168.0.1 | 172.16.0.3 | 50000    | 60002    | TCP   | 2022-01-01 00:00:02 | 300     |
| 4  | 192.168.0.1 | 172.16.0.4 | 50000    | 60003    | TCP   | 2022-01-01 00:00:03 | 300     |

## 各種問題のヒント

### (1).各種問題のヒント

各問題のヒントを以下に記載する。

#### 【DDL】

---

##### ● Q1 DB 作成

- ① create database 文法を使用する
  - ② DB 一覧を確認するメタコマンドは教科書編 1-3 節 「メタコマンドについて」に記載あり
- 

##### ● Q2 table 作成(public)

- ① psql から入り直す必要がある
  - ② テーブル作成には create table。制約がある点、注意。
  - ③ 別名指定には constraint 句を使用。また name には 2 つ制約をつける点、注意。
  - ④ メタコマンドでテーブル情報確認
- 

##### ● Q3 table 情報更新

- ①②③ 教科書編 2-3 節 (3)-c 「テーブルの変更処理」参照
  - ④ 教科書編 2-3 節 (3)-a 「制約の別名と変更」参照。2 種あるが、どちらかしかできない。
  - ⑤ 教科書編 2-3 節 (3)-a 「制約の別名と変更」参照。2 種あるが、どちらかしかできない。教科書に記載の「制約の記述」とは、\d テーブル名 で出てくる情報のどれか。
  - ⑥ alter table テーブル1 add foreign key (テーブル1 のカラム) references テーブル2(テーブル2 のカラム) ;
  - ⑦ alter table テーブル名 add check ( カラム名 like ‘‘\_’’ ) ;
- 

##### ● Q4 シーケンス追加とテーブル情報変更

- ① 教科書編 2-3 節 (3)-b 「シーケンスとは」参照
  - ② alter table [テーブル名] alter column [カラム名] set default [つける値] ;
  - ③ 2 問目④と同じ
- 

##### ● Q5 シーケンス関連動作の確認

- ① 教科書編 2-3 節 (3)-b 「シーケンスの設定詳細」参照。
- ②③⑥ 教科書編 2-3 節 (3)-b 「シーケンスとは」参照。
- ④ ②と同様の SQL。

- ⑤ ③を複数回実行するだけ。
  - ⑦ ⑥から設定する値を変える。
  - ⑧ 教科書編 2-3 節 (6) 「各種削除」参照。
- 

● Q6 index 付与

- ①② 教科書編 2-3 節 (3)-c 「インデックスとは」参照。
  - ③ カラム指定に複数のカラム情報を入れる。
  - ④ 教科書編 2-3 節 (6) 「各種削除」参照。
- 

● Q7 制約チェック

- ①～⑦ 教科書編 5-2 節 (1) 「データの挿入 1」参照。
  - ⑦～⑧ 教科書編 5-2 節 (3) 「データの削除」参照。条件式の指定も必要。条件式は 5-3 節 (1) 「条件の指定と演算子」参照。
- 

● Q8 index メンテナンス

- ①② 教科書編 2-3 節 (3)-c 「Index の管理」参照。
- 

●Q9 ビューとマテリアライズドビューの作成

- ②③ 教科書編 2-3 節 (4) 参照。select 以下は教科書編 5-2 節 (4) 「データの取得」参照。
  - ④ 教科書編 5-2 節 (4) 「データの取得」参照。
- 

●Q10 ビューとマテリアライズドビューの動作確認: 参照元の変更とその影響

- ③ 教科書編 5-2 節 (2) 「データの更新」参照。
  - ⑤ 教科書編 2-3 節 (4) 「マテリアライズドビューについて」 参照。
- 

●Q11 ビューとマテリアライズドビューの動作確認: ビューに対する変更処理  
特になし

---

●Q12 ビュー・マビューの削除

- ① 教科書編 2-3 節 (6) 「各種削除」参照。
- 

● Q13 table 作成(新スキーマ)

- ① 教科書編 2-3 節 (1) 「各種作成」参照。
- ②⑤ 教科書編 2-3 節 (3) 「テーブル作成の詳細」参照。

- ③ select でもテーブル作成時と同様にスキーマ指定が可能。
  - ④ SET 文で設定変更。参照スキーマは search\_path。SET search\_path TO [スキーマ名] ;
  - ⑦ ⑤と全く同じで OK
  - ⑧⑩12 メタコマンドでテーブルとスキーマ一覧を使用する。
  - ⑨ ④と同じで指定するスキーマを増やすだけ。
- 11 教科書編 2-3 節 (3)-c 「テーブルの変更処理」参照。ただし、⑩の通り、表示されないのでスキーマ指定が必要。  
13～16 教科書編 2-3 節 (5) 参照。
- 

#### ● Q14 スキーマ削除

- ①～③ 教科書編 2-3 節 (6) 「各種削除」参照。
- 

#### ● Q15 スキーマ削除(まとめて削除)

- ① カスケード削除オプションを指定するとテーブルごと削除可能。カスケードオプションの指定方法は前問題のログを見るとヒントとなる。
- 

#### ● Q16 drop でエラーとさせない

- ② 教科書編 2-3 節 (6) 「if exists 句について」参照。
- 

#### ● Q17 DB 名変更と DB 削除

- ①③ 教科書編 2-3 節 (2) 参照
  - ② psql でつなぎ直し
  - ④ 教科書編 2-3 節 (6) 「各種削除」参照。
- 

### 【DCL】

---

#### ● Q18 ロール作成

- ①～② 教科書編 3-3 節 (1) 「ロールの操作」 参照。
  - ③④ つなぎ直しには psql を使用。
  - ⑤ 教科書編 2-3 節 (1) 「各種作成」 参照。
- 

#### ● Q19 権限付与

- ①②④ 情報の取得は教科書編 5-2 節 (4) 「データの取得」 参照。
  - ③ 権限付与は教科書編 3-3 節 (2) 「テーブルに対する権限付与」 参照
-

## ●Q20 ロールにロール付与

③ 権限付与は教科書編 3-3 節 (2) 「ロールに(別)ロールを参加させる」参照

---

## ● Q21 権限削除

① 教科書編 3-3 節 (3) 参照。

③④ 教科書編 5-2 節 (2) 「データの更新」参照。

---

## ● Q22 所有者変更

①③⑤⑥ 教科書編 2-3 節 (1) 「各種変更」参照

② 教科書編 3-3 節 (4) 参照。

④ 期待出力は「ALTER ROLE」。

---

## ● Q23 ロール削除

①②④ 教科書編 3-3 節 (1) 「ロールの操作」参照。

③ 期待出力は「DROP DATABASE」「REVOKE」。ログをよく読む事。

---

## ● Q24 ビューと組み合わせたロールの活用

② ビュー定義の書き方は教科書編 2-3 節 (4) 「ビューについて」参照。

③④ 「CREATE ROLE」と「GRANT」が期待出力

---

## ●Q25 ロールに参加したロールの解除

② 教科書編 3-3 節 (3) 参照

③ まとめてロールを削除するには対象を「,」で区切って記載するだけ。

---

## 【TCL】

## ● Q26 トランザクション(正常)

②⑤ 教科書編 4-2 節 「トランザクション操作(1)」参照。

③ 教科書編 5-2 節 (2) 「データの更新」参照。

④⑥ 教科書編 5-2 節 (4) 「データの取得」参照。

---

## ● Q27 トランザクション(失敗)

特になし

---

● Q28 トランザクション(savepoint 復帰)

③ 教科書編 4-2 節 「トランザクション操作(1)」参照。

⑤⑦ 教科書編 4-2 節 「トランザクション操作(2)」参照。

---

● Q29 何もない所でトランザクションコマンドを打つと？

①③ 教科書編 4-2 節 「明示的トランザクションと暗黙的トランザクション」参照。

---

● Q30 トランザクション ID 確認

特になし

---

● Q31 トランザクション(ロック待ち)

特になし

---

● Q32 明示的テーブルロック

①③ 教科書編 4-3 節 「明示的ロックについて①」参照。

② 教科書編 4-4 節 「トランザクション分離レベルの変更方法」参照。

---

● Q33 明示的行ロック(共有ロック①)

② 教科書編 4-3 節 「明示的ロックについて②」参照。

---

● Q34 明示的行ロック(共有ロック②)

④ 教科書編 4-3 節 「明示的ロックについて②」参照。

⑤ 処理キャンセルは Ctrl + C で実行。

---

● Q35 明示的行ロック(排他ロック)

特になし

---

● Q36 デッドロック

特になし

---

● Q37 トランザクション分離レベル(ダーティリード未発生・反復不能読み取り 発生)

①② 教科書編 4-4 節 「トランザクション分離レベルの変更方法」参照。

---

● Q38 トランザクション分離レベル(反復不能読み取り 改善)

特になし

---

● Q39 トランザクション分離レベル(ファンтомリード 発生)

特になし

---

● Q40 トランザクション分離レベル(ファンтомリード 改善)

② 教科書編 4-4 節 「トランザクション分離レベルの変更方法」参照。

---

● Q41 トランザクション分離レベル(直列異常 改善)

②③ 教科書編 5-2 節 (1) 「データの挿入 1」 及び 5-3 (2) 「集約関数」 参照。カラムの値の合計は sum 関数を使用する。特定列の合計値を算出してその結果を挿入するには insert ~ select 文を使用する。

---

● Q42 トランザクション分離レベル(直列異常 発生)

特になし

---

● Q43 SET 文の影響範囲の理解

特になし

---

● Q44 トランザクション分離レベルの永続的変更(1)

① 期待出力は「 ALTER DATABASE 」。教科書編 2-3 節 (1) 「各種変更」 参照。

② set で使用していた transaction isolation level は使用できない。default\_transaction\_isolation を指定する必要がある。

③ 設定するのは set。設定リセットは reset を使う。(set でもできるが、default を指定すると良い。)

---

● Q45 トランザクション分離レベルの永続的変更(2)

① 期待出力は「 ALTER SYSTEM 」。

---

【DML(基礎)】

---

● Q46 シンプルな select(事前準備が正しくできているか確認 )

①②③ 教科書編 5-2 節 (4) 「データの取得」 参照。

---

● Q47 一部のレコード削除

教科書編 5-2 節 (3) 「データの削除」 及び 5-2 節 (2) 「更新箇所のチェック」 参照。

---

● Q48 delete 以外によるデータ削除

教科書編 5-2 節 (3) 「データの削除」 参照。

---

● Q49 データ投入

①② 教科書編 5-2 節 (1) 「データの挿入 1」 参照。

③ 文字列の結合は教科書編 5-3 節 (2) 「文字列関数」 参照。

---

● Q50 データ更新

AX・BX にある「X」という文字を「XBU」に置き換えると、AX1 は AXBU1、 BX2 は BXBU2 になる。

文字の置換には関数を使用する。

教科書編 5-3 節 (2) 「文字列関数」 参照。

---

● Q51 csv を使ったデータ運用

①③ 教科書編 5-2 節 (1) 「データの挿入 2」 参照。

---

● Q52 複数条件の指定方法

①②③ 教科書編 5-3 節 (1) 「条件の指定と演算子」 参照。

④ (カッコ)の使いどころがポイント

---

● Q53 算術関数の使用

① 教科書編 5-3 節 (2) 「算術関数とキャスト変換について」 参照。

② 関数の結果から関数の結果を引く事ができる

---

● Q54 カラム名の指定

①② 教科書編 5-3 節 (4) 「distinct と count の組み合わせと結果出力について」 参照。

---

● Q55 集約関数の使用

① 教科書編 5-3 節 (3) 「集約関数」 参照。

② round 関数と trunc 関数の同時使用をする。

③ where で条件を指定したうえで平均をとる

---

**● Q56 集約関数とグルーピング**

① 教科書編 5-3 節 (5) 「グルーピング」参照。

---

**● Q57 グルーピングと条件式**

①② where か having のどちらかで dept\_id = 3 以外を指定する。教科書編 5-3 節 (5) 「where と having の違い」参照。

---

**● Q58 該当レコード数の確認**

①② 教科書編 5-3 節 (3) 「ヒット数の取得」参照。

---

**● Q59 null の取得**

①② 教科書編 5-3 節 (3) 「Null レコードの取得について」参照。

---

**● Q60 演算子の優先度**

①② 教科書編 5-3 節 (1) 「その他演算子」参照。

---

**● Q61 時刻演算子**

①② 教科書編 5-3 節 (1) 「その他演算子」参照。

---

**● Q62 時刻関数**

①② 教科書編 5-3 節 (2) 「日付/時刻関数」参照。

---

**● Q63 時刻関数の取得タイミング**

①②③ 教科書編 5-3 節 (2) 「日付/時刻関数」参照。③の SQL の意味は補足に記載。

---

**● Q64 データ型書式設定関数**

教科書編 5-3 節 (2) 「データ型書式設定関数」参照。

---

**● Q65 文字関数**

①② 教科書編 5-3 節 (2) 「文字列関数」参照。

③ Ipad は文字列型を扱う。dept\_id は int の為、キャストが必要になる。

---

● Q66 generate\_series

- ①② 教科書編 5-3 節 (2) 「その他関数」参照。  
③ IP アドレスは inet 型を使う。192.168.10.1 をそのまま指定してもただの文字列となるため、キャスト変換が必要となる。
- 

● Q67 文字検索(like, ilike)

- ①② 教科書編 5-3 節 (3) 「文字検索(like, ilike)」参照。
- 

● Q68 文字検索(similar to)

- ① 教科書編 5-3 節 (3) 「文字検索(similar to)」参照。
- 

● Q69 文字検索(正規表現)

- ① 正規表現に一致、大文字小文字を区別する方法で検索可能。教科書編 5-3 節 (3) 「文字検索(正規表現)」参照。  
② 正規表現に不一致、大文字小文字区別しない方法で検索可能。
- 

● Q70 重複排除

- ①②③ 教科書編 5-3 節 (4) 「重複排除」参照。
- 

● Q71 重複排除とヒット数取得の組み合わせ

- ①③ 教科書編 5-3 節 (4) 「distinct と count の組み合わせと結果出力について」参照。
- 

● Q72 並び替え

- ①② 教科書編 5-3 節 (6) 「並び替えと取得数について」参照。
- 

● Q73 重複排除並び替えと並び替え指定、どちらが優先されるか

3問前の②を見返すと良い。

---

● Q74 取得結果の範囲指定

- ①②③ 教科書編 5-3 節 (6) 「並び替えと取得数について」参照。
- 

● Q75 取得範囲の指定

教科書編 5-3 節 (6) 「範囲の取得」参照。

---

● Q76 まとめ問題

コツはいきなり全部を作ろうとするのではなく、一部分をちょっとずつ作っていく事。

① status が null のもの = null のものも含めたデバイス数 - null のものを除いたデバイス数

② 特になし

---

【DML(応用)】

---

● Q77 結合 1

①② 教科書編 6-2 節「内部結合とは」参照。

---

● Q78 結合 2

① ~ ④ 教科書編 6-2 節「結合条件とは」参照。

---

● Q79 結合 3

① ~ ④ 教科書編 6-2 節「結合条件とは」「外部結合とは」参照。

---

● Q80 結合 4

① ~ ④ 教科書編 6-2 節「結合条件とは」「外部結合とは」参照。

---

● Q81 結合 5

① ~ ④ 教科書編 6-2 節「結合条件とは」「外部結合とは」参照。

---

● Q82 結合 6

① ~ ③ 教科書編 6-2 節「内部結合と外部結合の関係性について②」参照。

---

● Q83 結合 7

①② 教科書編 6-2 節「内部結合と外部結合の関係性について③」参照。

---

● Q84 結合 8

①② 教科書編 6-2 節「交差結合とは」参照。

---

● Q85 3つのテーブル結合

書式は `select * from [tbl名1] inner join [tbl名2] on [条件] inner join [tbl名3] on [条件];`

---

● Q86 3つのテーブル結合時の取得範囲指定 1

特になし。これまで問題で解いてきたことを組み合わせれば回答できるはず。

---

● Q87 3つのテーブル結合

特になし。これまで問題で解いてきたことを組み合わせれば回答できるはず。

---

● Q88 3つのテーブル結合

特になし。これまで問題で解いてきたことを組み合わせれば回答できるはず。

---

● Q89 select 結果で update に近い処理

①② 書式は `update [更新対象テーブル] set [カラム名1] = [参照先テーブル].[カラム名2] from [参照先テーブル] where [条件式];`

( insert は insert ~ select があるが、update は上記構文で select 結果で更新が可能)

---

● Q90 副問合せの基本

①② 教科書編 6-4 節 (1) 「副問合せ」参照。

---

● Q91 3つの副問合せ

副問合せは入れ子にできる。`select ~ from (select ~ from ( select ~ ) )` というイメージ。

---

● Q92 副問合せ同士の join

`select ~ from ( xxxx ) a full outer join ( yyyy ) b on [条件式]` で記載する。

---

● Q93 副問合せを条件に指定

副問合せで users テーブルから、grade(評価)が最も低い人の所属する部署 id を取得して、外側の問い合わせに渡してあげると良い。

教科書編 6-4 節 (1) 「副問合せの別の書き方」参照。

---

● Q94 from より前の副問合せ

- ① 教科書編 6-4 節 (1) 「from より前に記載する副問合せ①」参照。  
 ②③ 教科書編 6-4 節 (1) 「from より前に記載する副問合せ②」参照。
- 

● Q95 相関副問合せ

- ① 教科書編 6-4 節 (2) 「相関副問合せについて①」参照。  
 ② 教科書編 6-4 節 (2) 「相関副問合せについて②」参照。  
 ③ 教科書編 6-4 節 (2) 「相関副問合せについて②」「同③」参照。
- 

● Q96 シンプルな in の使い方

教科書編 6-4 節 (3) 「in のシンプルな使い方」参照。

---

● Q97 複数のレコードを条件とした副問合せ(in)

- ①③ 教科書編 6-4 節 (3) 「in を使った副問合せ」参照。
- 

● Q98 複数のレコードを条件とした副問合せ(some/any)

- ①② 教科書編 6-4 節 (3) 「some/any を使った副問合せ」参照。  
 ③ some/any の動作を理解する事が必要。難しければ補足を確認する事。
- 

● Q99 some/any と対となる動作をする指定

「postgres 副問合せ式」で検索。

---

● Q100 シンプルな exists による副問合せ

- ①～③ 教科書編 6-4 節 (3) 「exists について」参照。
- 

● Q101 exists と相関副問合せの組み合わせ

教科書編 6-4 節 (3) 「exists と相関副問合せを使った動作イメージ」参照。

---

● Q102 with 句の使用

教科書編 6-5 節 「with を使った問い合わせについて」参照。

---

● Q103 2つのクエリ取得結果の結合

- ①② 教科書編 6-3 節 「2つの select の結果をまとめる」参照。

● Q104 重複行の取得

教科書編 6-3 節「2つの select の共通部分を取得」参照。

---

● Q105 2つの select の差分を取得

教科書編 6-3 節「2つの select の差を取得」参照。

---

● Q106 重複排除した行の取得

`distinct` をしないで `device` から `user_id` を取得した結果と、`distinct` して `device` から `user_id` を取得した結果の差分をとる。( `except` )

ただし、`except` は2つのテーブルの差分をとるが、同じ値(重複)があった場合はデフォルトで排除してしまう。

⇒ 排除しないオプションが必要。postgresの公式ドキュメントで調べてみると良い。

---

● Q107 再帰的問い合わせ

① 教科書編 2-3 節 (3)-b 「シーケンスとは」参照。

② 教科書編 6-5 節「再帰的問い合わせ 2」参照。

---

● Q108 クエリ実施の詳細確認

①② 教科書編 6-6 節 (1) 「SQLの動作チェック」参照。

---

● Q109 Window 関数 1

① 教科書編 6-6 節 (2) 「Window 関数とは」参照。

② 教科書編 6-6 節 (2) 「Window 関数を使った順位付け」参照。

---

● Q110 Window 関数 2

教科書編 6-6 節 (2) 「Window 関数を使った比較」参照。

---

● Q111 UPSERT1

②④⑥ 教科書編 6-6 節 (3) 「特定条件の時だけ挿入 or 更新」参照。

③ エラーログにヒントあり。

---

● Q112 UPSERT2

①③ 教科書編 6-6 節 (3) 「特定条件の時だけ挿入 or 更新」参照。

---

### ● Q113 UPSERT3

① 教科書編 6-6 節 (3) 「複数 upsert を使用した場合」参照。

---

### ● Q114まとめ

`union` を使用する。(②+③)

②と③の出力は青枠参照。

ヒント: ②の結果

`user_id`

-----  
4  
2

ヒント: ③の結果

`user_id`

-----  
1  
2

### 【7章-①: 外部制約の詳細や条件文等について】

---

### ● Q115 自ら定義する型

①② 教科書編 7-2 節 (2) 参照。

③④ 教科書編 5-2 節 (1) 「データの挿入 1」参照。

⑤ 教科書編 5-2 節 (2) 「データの更新」参照。

---

### ● Q116 プリペアドステートメントを対話モードで試す

教科書編 7-2 節 (3) 参照。

②で上手くいかない場合、引数を何でも良いので指定する事を試してみると良い。

---

### ● Q117 外部キー制約のアクション 1

① 教科書編 7-2 節 (4) 「外部キー制約のアクションについて(2)」参照。

② ①の期待出力のエラーログを見ること。

---

### ● Q118 外部キー制約のアクション 2

② ①の期待出力のエラーログを見ること。

③ エラーログに `not null` という記載がある。`not null` に関連している。参照先の `employee` の値を変えると参照元はどうなるかをあわせて確認。

⑥ 参照先のデータを消すと参照元は `default` の値を入れる設定となっている。参照元に情報を登録する際には何が必要か改めて確認する。

● Q119 制約解除

- ① うまくいかなかった場合、ログを確認する
- 

● Q120 複合外部キー制約

- ① 教科書編 7-2 節 (4) 「外部キー制約の別の指定方法」参照。
- 

● Q121 条件に応じた出力

- ② 副問い合わせと case 文を使用する。副問い合わせは教科書編 6-4 節 (1) 「副問い合わせ」、case 文は教科書編 7-2 節 (5) 「条件判定」「case 文の具体例」参照。
- 

● Q122 複数パターンによる集約処理の出力

- ①②③ group by XXX (列名) という書式を使う。教科書編 7-2 節 (5) 「まとめて様々なパターンを集計」参照。
- 

【7章-②:パーティションとテーブル空間】

---

● Q123 リストパーティションとテーブル空間

- ① 教科書編 7-2 節 (1) 「テーブル空間」参照。  
② 教科書編 7-2 節 (1) 「宣言的パーティショニング」参照。  
③ 教科書編 7-2 節 (1) 「テーブル空間」「宣言的パーティショニング」参照。2つの合わせ技。
- 

● Q124 リストパーティションとテーブル空間の動作確認

- ① 教科書編 5-2 節 (1) 「データの挿入 1」参照。  
② 教科書編 5-2 節 (4) 「データの取得」参照。  
③④ 教科書編 7-2 節 (1) 「テーブル空間の設定確認」参照。
- 

● Q125 リストパーティションの子テーブルを跨いだ処理

- ① 教科書編 2-3 節 (3)-c 「インデックスとは」参照。  
②④ 教科書編 5-2 節 (2) 「データの更新」参照。
- 

● Q126 リストパーティションと主キーについて

- ① ログ情報から読み取れる。
-

● Q127 リストパーティションの範囲外の値について

② ヒント 1(期待出力): CREATE TABLE

ヒント 2: default 設定を入れると、指定外は該当テーブルにすべて登録されるようになる

---

● Q128 パーティションテーブルのデータ削除について

① 教科書編 5-2 節 (3) 「データの削除」参照。

②④ 教科書編 2-3 節 (6) 「各種削除」参照。

---

● Q129 レンジパーティション

①② 教科書編 7-2 節 (1) 「パーティショニングの種類」参照。

③ 現時点から N 年前は now() - interval 'N year' を使う。

---

● Q130 ハッシュパーティション

① 教科書編 7-2 節 (1) 「パーティショニングの種類」参照。

③ ログ情報からなぜエラーとなったかは読み取れる。

---

【7章-③: contrib と dblink】

---

● Q131 dblink による DB 作成

② 教科書編 7-2 節 (6) 「dblink の使い方」参照。

③ 教科書編 7-2 節 (6) 「dblink の使い方」「2 相コミットメントの具体的な SQL 例」参照。

---

● Q132 dblink に寄るテーブル作成

③ 期待出力 : CREATE EXTENSION

---

● Q133 dblink による挿入とレコード取得

② ‘シングルクォート’内で ‘シングルクォート’を使うには、”AAA”のような形で二重でシングルクォートで囲う(エスケープする)

③ デフォルト DB で取得したものに、リモート DB で取得したものをあわせる(教科書編 6-3 節「2 つの select の結果をまとめること」参照。)

④ union した結果を並び替える為、dblink の外側で並び替えを実施する

---

● Q134 dblink を使った update, delete

特になし

● Q135 2相コミットメントを使う為の準備

- ① 期待出力：色々出力されるが、最終的に「server started」が出て、ポート 5432 test06c に接続できればOK。  
ヒント：教科書編 7-2 節 (6)「2相コミットメントのその他」参照。
- 

● Q136 2相コミットメント(処理成功)

- ①②④ 教科書編 7-2 節 (6)「2相コミットメントの具体的な SQL 例」参照。  
③⑤ 教科書編 7-2 節 (6)「2相コミットメントのその他」参照。
- 

● Q137 2相コミットメント(処理失敗)

- ①②④ 教科書編 7-2 節 (6)「2相コミットメントの具体的な SQL 例」参照  
③⑤ 教科書編 7-2 節 (6)「2相コミットメントのその他」参照
- 

【7章-④: PL/pgSQL と自動処理】

---

● Q138 シンプルなトリガーの用意

- ① 教科書「関数の書き方について①」参照  
② 教科書「トリガーの書き方」参照  
③ 教科書「ファンクションの更新と確認について」参照  
④ 教科書「トリガー,ルールの更新、確認方法」参照
- 

● Q139 トリガーの動作確認

- ①③ insert 文を実行する  
②④ select 文を実行する  
③ 期待出力は CREATE FUNCTION。 replace でも create と表示される  
⑤ update 文を実行する
- 

● Q140 IF 文を含んだファンクション

- ① ヒント 1: 期待出力（以下参照）

```
CREATE FUNCTION
DROP TRIGGER
CREATE TRIGGER
```

ヒント 2: 教科書「関数の書き方②③」「トリガー・ルールの更新、確認方法」参照。文字の結合は || を使うと良い。

②③ なし

---

● Q141 キーワードに該当する情報がない場合の処理

特になし

---

● Q142 関数の行ごと処理と SQL 文ごと処理の違い

③ 空欄の理由 : person テーブルの更新対象レコードに理由がある。(前ページを見てみて、考えてみましょう)

---

● Q143 登録したい情報空欄化 一次改善

特になし

---

● Q144 登録したい情報空欄化 恒久改善

③ ヒント 1(期待出力) : CREATE FUNCTION

ヒント 2 : memo が null だった時に null ではなくする置換が必要。置換には coalesce 関数を使う。

---

● Q145 特定カラムが処理された場合のトリガー

① 集計した値を入れるには update table 名 set カラム名 = ( select 集計処理 ~ ) を使うと良い。

なお、人数が変動するのは更新前の部署（人数が減る）のと、更新後の部署（人数が増える）の 2 つがあることを忘れずに考慮すること。

② 特定カラムが処理された場合のみ動くトリガーを指定する場合は「of カラム名 on テーブル名」を指定する

---

● Q146 SQL 文ごと実施関数の集計処理

特になし

---

● Q147 テーブル間のズレの防止

④ 「特定の値が存在しない場合」は if not exists ( select ~ ) を使う。またエラー文は「 raise exception 'エラー文' 」とする。

⑥ 想定通りのテーブル内のデータと正しい SQL を入れた上で、UPDATE 0 となる場合は④のファンクションが誤っている。補足も参考にすること。

---

● Q148 やや複雑な入力値チェック

特になし

---

● Q149 トリガーファンクションによる処理代替

③④ 代替処理は rule にも似たような機能がある。キーワードはそのキーワードに of をつける。

---

● Q150 ルールによる代替処理

⑤ pg\_rules の中身をみて、指定に役立ちそうな列がないか確認する

---

● Q151 ルールによる追加処理について

⑤ ④の推測は一部正しい。

---

● Q152 トリガー,ルール確認に便利なメタコマンド

① テーブル情報をみるメタコマンドを使う

② テーブル情報を見るメタコマンドに追加の文字を追加する

---

● Q153 ストアドファンクション1

①

ヒント1: ベース賃金への上乗せ額を出す為の勤続年数は、now()の年から memo の年をひいて取得する。now()は timestamp だが、YYYYだけとるには to\_char 関数を使う。( to\_char(now(), 'YYYY') )で整形できる。

ヒント2: person テーブルから memo の情報をとてこないといけない。とってきた情報を変数として使うには select memo into 変数 from person という文法を使う。

ヒント3: memo は文字列型(text)で入っている。キャスト変換が必要となる。

② 教科書「ストアドファンクションについて」参照。

---

● Q154 ストアドファンクション2

①

ヒント1: 全ユーザの給与を計算するには、ユーザずつ同じ処理(給与算出)を繰り返す。よって関数内で「for」文の実施が必要となる。

ヒント2: 戻り値は複数のレコード情報となる。よって returns XXX で指定する「返却する型」はこれまでと異なる。複数の行を返す際には table 型 ( table ( 変数型, ~ ) )を指定する。

ヒント3: ②の出力の1行ずつを、for 文内で返却する table 型に埋め込んでいく必要がある。その埋め込みには「return query SQL文(select ~)」という記載方法が使える。

ヒント4: returns の table 型で指定するカラム名と、関数内の select 文で使うカラム名は重複を避けた方が良い

---

● Q155 ストアドファンクション 3

特になし。これまでの要素を組み合わせれば実現できるはず。

---

● Q156 ストアドプロシージャ

① 引数は配列を指定する。その場合、変数型[]という形で指定する。

②⑤ 教科書「ストアドプロシージャのその他処理」参照

③ 各列の情報は配列で渡す。配列を指定する際には array[ 1 , 2 , 3 ] のような指定をする。

---

【+α: PL/pgSQL や自動処理】

---

● Q157 ルールの処理実行順序 1

② 失敗理由：ログの意味合いを考える

---

● Q158 ルールの処理実行順序 2

② 5秒かかる仕組みは「pg\_sleep(n)」を使うと良い。

---

● Q159 ルールの処理実行順序 3

② delete 文にも select の結果を含める方法がある。その手法を使うと良い。

---

【+α: 更に色々試してみたい人向け】

---

● Q160 generate\_series を使った試験データ投入

① users にユーザは 5 人いる。クロスジョインを使用する場合、50 生徒分作りたいので generate\_series は 10 回分の数値払い出し。generate\_series は教科書 P69 参照

② generate\_series で timestamp を扱う場合は generate\_series(開始時点、終了時点、インクリメントする日) で記載すると良い。クロスジョインを使う場合は history には 2 件レコードがある為、日にちは 25 日分のデータを取得する (2×25 で 50 履歴作成ができる)

---

● Q161 重複を削ったうえで、null もカウントするには？

Coalesce(教科書に記載していない)を使用する。使い方や意味合いを調べながらやってみましょう。

---

● Q162 特定情報を持つ情報を除いた計算

nullif 関数(教科書に記載していない)を使用する。使い方や意味合いを調べながらやってみましょう。

---

● Q163 レコードごとの最大値・最小値

最大は greatest、最小は least を使用する。どちらも教科書に記載していない為、使い方や意味合いを調べながらやってみましょう。

---

● Q164 現在情報の取得

「システム情報関数」で検索してみましょう。

---

● Q165 様々な index

- ① create unique index 文
  - ② 条件は通常の index 作成文の後に where をつければ良い
  - ③ 文字列を大文字にするには upper 関数を使用する。カラム指定時に関数も指定。
- 

● Q166 role の DB 権限

- ⑤ 文法は grant [権限] on database [DB名] to [ロール名] ;
- 

● Q167 role の管理+α

- ① role はユーザともいえる。
  - ③ user01 を上乗せするイメージ
- 

● Q168 ロングトランザクション排除(強制停止)

実行クエリー一覧は pg\_stat\_activity を確認。  
トランザクション停止は pg\_terminate\_backend()を使用。

---

● Q169 ロングクエリの停止

pg\_cancel\_backend()を使用する

---

● Q170 サイクルシーケンスの動確

- ① オプションは通常のシーケンス作成文の最後に cycle をつけるだけ。
- 

● Q171 試験データ作成

特になし

## (2).各種問題の答え

答えは以下に配置している。

適宜ダウンロードして参考にしてみてほしい。

答えダウンロードサイト：[情報畠でつかまえて<sup>\\*1</sup>](#)（当社ブログサイト）

---

1. <https://www.ntt-tx.co.jp/column/>

## 各種問題の補足

各問題の補足を以下に記載する。

---

### ★ Q11 ビューとマテリアライズドビューの動作確認：ビューに対する変更処理

問題ではビューを指定したデータの追加・更新・削除が実施できたが、常にできるわけではない。

更新等が可能なのは、作成した際に参照したテーブルが1つの場合に限る。

テーブルを2つ以上して参照してビューを作った場合（例：JOIN等）は更新処理はできない。

JOINについては「6章 DML（応用）」参照。

7章の問題内に「2つのテーブルを結合したビュー」を作成する試験がある為、気になる方は参照。

---

### ★ Q18 ロール作成

user01やsuperuser01にパスワードを付与したが、きかれない。

この理由はpg\_hba.confという設定ファイルによる為。

（配置場所は /var/lib/postgresql/12/data01/配下にある）

pg\_hba.confは接続許可・非許可を設定するファイル。

初期は以下のようになっており、ローカルからの接続はtrust（無条件で信頼し許可する）ものとしている。

| #                                                    | TYPE | DATABASE | USER | ADDRESS      | METHOD |
|------------------------------------------------------|------|----------|------|--------------|--------|
| # "local" is for Unix domain socket connections only |      |          |      |              |        |
| local                                                | all  | all      |      |              | trust  |
| # IPv4 local connections:                            |      |          |      |              |        |
| host                                                 | all  | all      |      | 127.0.0.1/32 | trust  |
| # IPv6 local connections:                            |      |          |      |              |        |
| host                                                 | all  | all      |      | ::1/128      | trust  |

パスワードを聞かれるようにするには必要箇所をtrustからpasswordに変えればよい。

パスワード確認をするだけであればviコマンドなどでpg\_hba.confのall行をMETHODをtrustからpasswordに変更にして設定ファイル再読み込みを行えば良い。

以下に実行例を記載する。

```
vi /var/lib/postgresql/12/data01/pg_hba.conf
※ all行のMETHODをpasswordに変更する
```

```
pg_ctl reload -D /var/lib/postgresql/12/data01
```

```
psql -U superuser01 -d test02
```

上記実行後は以下のようにパスワードが聞かれる。

```
postgres@ubuntu-focal:~$ psql -U superuser01 -d test02
Password for user superuser01:
psql (12.20 (Ubuntu 12.20-0ubuntu0.20.04.1))
Type "help" for help.

test02=#
```

興味がある方は問題には含まれないが、試してみてほしい。

なお、本書の問題は trust のままで実施する事を想定している為、確認が終わったら戻しておくことを推奨する。

また pg\_ctl reload コマンドは設定ファイル読み込みに使えるが、全設定変更の反映に使えるわけではないので注意。

変更した設定によっては pg\_ctl restart をしないと行けない場合がある。

---

#### ★ Q21 権限削除

user01 で select 権限を外しただけにもかかわらず、update もできなくなってしまった。

その理由は「where 句」による条件が select 権限を使う為である。

その為、where 句を使用しない場合は update を使用できた。

これは update だけでなく delete も同様である。（= where を使う場合、同様）

update や delete を使う場合には where を使う事がほとんどだと思う為、注意しよう。

---

#### ★ Q24 ビューと組み合わせたロールの活用

本問題はビューとロールの組み合わせ以外に、ロールのグループのような使い方、ユーザのような使い方のイメージを読者に伝わる事も目的としていたが、イメージはもてただろうか。

本問題の構成は以下のようなイメージであった。

図も参考にして、グループのような使い方、ユーザのような使い方のイメージを掴んでほしい。

## 当該問題の構成イメージ

【personテーブル】

| id | name | status |
|----|------|--------|
| 1  | AAA  | Active |
| 2  | BBB  | Ended  |
| 3  | CCC  | Active |

グループのような使い方。  
このアカウントはログインしない為、  
ログイン権限を持たず、  
ビューへの権限のみ持つ。

【ope\_viewビュー】

| id | status |
|----|--------|
| 1  | Active |
| 3  | Active |

ユーザのような使い方。  
ビューへの権限は直接は持たないが  
opeロールに参加することで付与される。  
ユーザとして使う為、ログイン権限を持つ。

参照可

opeロール

ope001ロール

ope002ロール

ope003ロール

### ★ Q45 トランザクション分離レベルの永続的変更(2)

alter database と alter system の違いは設定反映タイミングのみではない。

alter database は対象となる DB(test03)のみであり、他の DB(postgres など)では変更されない。

alter system は全 DB(DB クラスタ)に設定反映がされる。

なお、alter system の反映タイミングは設定ロード(pg\_ctl reload)や postgres の再起動(pg\_ctl stop→start or restart)で反映される。(変更する設定により変わる。)

postgres の設定は基本 postgresql.conf に記載されているが、alter system を実行すると postgresql.auto.conf というファイルが出来上がり、こちらも読み込まれる。(設定の優先度は postgresql.auto.conf > postgresql.conf )

ちなみに transaction isolation level は設定ロードでも再起動でも可能である。

興味ある方は対話モード(SQL 実行モード)から抜けて、pg\_ctl reload をするか、対話モード内で「select pg\_reload\_conf();」を実行してみよう。

## ★ Q56 時刻関数の取得タイミング

### 問題内に書かれたSQL文について

#### ■ 概要

問題内では以下のようなSQLを実行した。  
SQLは「;」で完了する為、大きく分けると赤字SQLと緑SQLにわかれる。

```
select now(); select statement_timestamp() from (select pg_sleep(10)) a;
```

#### ■ 2つのSQLをまとめて記載する事について

まとめて1回の処理で書いてしまう事で、1つ目の処理が終わったら即座に2つ目のSQLを実行できる。  
前回だと「時刻取得の結果の違い」が知りたかったため、赤字SQLで現在の時間を確認し、確認が完了後、即座に緑SQLを実行し、結果の違いを確認した。

#### ■ 緑字SQLについて

##### ①副問合せについて

「`select ~ from (select ~)`」という書式になっている。  
この書式は「副問合せ」と呼び、(カッコ)内のSQLを実行してから、最初のselectを実行するもの。  
※ 詳細は教科書P89以降を参照。

##### ②`pg_sleep(n)`について

これはnで指定した秒数、待機する、という関数。(遅延実行関数と呼ぶ)

⇒ つまり、以下のようないしをしている。(これで時刻関数を呼び出すのを遅らせている)

1. (カッコ)内の処理: SQL開始後、10秒待機
  2. (カッコ)外の処理: 時刻関数呼び出し
- これにより、SQL開始時点と、関数呼び出し時点で明確に時間の差が生じる結果となった。

## ★ Q78 3つの副問合せ

本問題のSQLは長文である為、組み立て方の流れを記載する。

### 長文SQLの分解/内容について

#### ① deviceテーブルからカウント取得

```
select user_id , count(*) from device group by user_id ;
```

| device_id | user_id | status | create_at   |
|-----------|---------|--------|-------------|
| 1         | 1       | t      | [timestamp] |
| 2         | 2       |        | [timestamp] |
| 3         | 4       | f      | [timestamp] |
| 4         | 4       | t      | [timestamp] |
| 5         | 6       | t      | [timestamp] |

aテーブル

| User_id | Count |
|---------|-------|
| 1       | 1     |
| 2       | 1     |
| 4       | 2     |
| 6       | 1     |



#### ② dept\_idごとにカウント

```
(select dept_id ,count(*) from (略) a inner join users on a.user_id = users.user_id where count = 1 group by dept_id)
```

| User_id | Count | user_id | dept_id | name | grade |
|---------|-------|---------|---------|------|-------|
| 1       | 1     | 1       | 1       | AAA  | 3.5   |
| 2       | 1     | 2       | 1       | BBB  | 2.5   |
|         |       | 3       | 2       | BX   | 3.2   |
| 4       | 2     | 4       | 3       | cDEF | 4.5   |
| 6       | 1     | 5       | 5       | AX   | 3.8   |

bテーブル

| dept_id | Count(user_id) |
|---------|----------------|
| 1       | 2              |
| 3       | 1              |



#### ③ bテーブルと部署テーブル結合

```
select * from (select 略 (略) a) b inner join dept on b.dept_id = dept.dept_id where b.count > 1 ;
```

| dept_id | Count(user_id) | dept_id | name |
|---------|----------------|---------|------|
| 1       | 2              | 1       | AX   |
|         |                | 2       | BX   |
| 3       | 1              | 3       |      |
|         |                | 4       | AX   |

| dept_id | Name |
|---------|------|
| 1       | AX   |



## ★ Q86 3つのテーブル結合時の取得範囲指定1

本問題①のSQLは、以下のような構造となっている。

②も考え方ほぼ同じである。

3つのテーブル結合で複雑に見えるが、これまでやってきたことと大きくは変わらない。

【deptテーブル】

| dept_id | Name | user_id | dept_id | Name | grade |
|---------|------|---------|---------|------|-------|
| 1       | AX   | 1       | 1       | AAA  | 3.5   |
| 2       | BX   | 2       | 1       | BBB  | 2.5   |
| 3       |      | 3       | 2       | CDef | 3.2   |
| 4       | AX   | 4       | 3       | cdEF | 4.5   |
|         |      | 5       | 5       | GGG  | 3.8   |



外部結合(left)

【usersテーブル】

| dept_id | name | user_id | dept_id | name | grade | device_id | user_id | status | create_at   |
|---------|------|---------|---------|------|-------|-----------|---------|--------|-------------|
| 1       | AX   | 1       | 1       | AAA  | 3.5   | 1         | 1       | t      | [timestamp] |
| 1       | AX   | 2       | 1       | BBB  | 2.5   | 2         | 2       |        | [timestamp] |
| 2       | BX   | 3       | 2       | CDef | 3.2   |           |         |        |             |
| 3       |      | 4       | 3       | cdEF | 4.5   | 3         | 4       | f      | [timestamp] |
| 4       | AX   |         |         |      |       | 4         | 4       | t      | [timestamp] |
|         |      |         |         |      |       | 5         | 6       | t      | [timestamp] |



外部結合(left)

| dept_id | name | user_id | dept_id | name | grade | device_id | user_id | status | create_at   |
|---------|------|---------|---------|------|-------|-----------|---------|--------|-------------|
| 1       | AX   | 1       | 1       | AAA  | 3.5   | 1         | 1       | t      | [timestamp] |
| 1       | AX   | 2       | 1       | BBB  | 2.5   | 2         | 2       |        | [timestamp] |
| 3       |      | 4       | 3       | cdEF | 4.5   | 3         | 4       | f      | [timestamp] |
| 3       |      | 4       | 3       | cdEF | 4.5   | 4         | 4       | t      | [timestamp] |
| 4       | AX   |         |         |      |       |           |         |        |             |

dept

users

device

deptにしかないものは、usersともdeviceとも一致条件がない為、どちらの列も空になるはず。  
 (どちらか一方でも埋まつていれば、deptと他1つに存在する事になり、  
 deptのみに存在、という条件とはならない)  
 =users.dept\_id is null and device.user\_id is nullを指定

## ★ Q87 3つのテーブル結合時の取得範囲指定 2

本問題①は right, left join が入り混じっているが、その理由は把握できただろうか。

以下に構造を記載する為、考え方・理解の参考としてほしい。

【deptテーブル】

| dept_id | name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |

【usersテーブル】

| user_id | dept_id | Name | grade |
|---------|---------|------|-------|
| 1       | 1       | AAA  | 3.5   |
| 2       | 1       | BBB  | 2.5   |
| 3       | 2       | CDef | 3.2   |
| 4       | 3       | cdEF | 4.5   |
| 5       | 5       | GGG  | 3.8   |

↓ 外部結合(right)

残したい情報があるテーブルのほうを指定する為、rightを指定する。  
(usersテーブルに残したい情報がある)

【deviceテーブル】

| dept_id | name | user_id | dept_id | name | grade | device_id | user_id | status | create_at   |
|---------|------|---------|---------|------|-------|-----------|---------|--------|-------------|
| 1       | AX   | 1       | 1       | AAA  | 3.5   | 1         | 1       | t      | [timestamp] |
| 1       | AX   | 2       | 1       | BBB  | 2.5   | 2         | 2       |        | [timestamp] |
| 2       | BX   | 3       | 2       | CDef | 3.2   |           |         |        |             |
| 3       |      | 4       | 3       | cdEF | 4.5   | 3         | 4       | f      | [timestamp] |
|         |      | 5       | 5       | GGG  | 3.8   | 4         | 4       | t      | [timestamp] |
|         |      |         |         |      |       | 5         | 6       | t      | [timestamp] |

↓ 外部結合(left)

残したい情報があるテーブルのほうを指定する為、leftを指定する。  
(usersテーブルに残したい情報がある)

| dept_id | name | user_id | dept_id | name | grade | device_id | user_id | status | create_at   |
|---------|------|---------|---------|------|-------|-----------|---------|--------|-------------|
| 1       | AX   | 1       | 1       | AAA  | 3.5   | 1         | 1       | t      | [timestamp] |
| 1       | AX   | 2       | 1       | BBB  | 2.5   | 2         | 2       |        | [timestamp] |
| 3       |      | 4       | 3       | CDef | 3.2   | 3         | 4       | f      | [timestamp] |
| 3       |      | 4       | 3       | cdEF | 4.5   | 4         | 4       | t      | [timestamp] |
|         |      | 5       | 5       | GGG  | 3.8   |           |         |        |             |

dept

users

device

usersにしかないものは、deptともdeviceとも一致条件がない為、どちらの列も空になるはず。  
(どちらか一方でも埋まつていれば、usersと他1つに存在する事になり、  
usersのみに存在、という条件とはならない)  
=dept.dept\_id is null and device.user\_id is null を指定

あわせて問題②はこれまでやってきた outer join だけでなく、inner join も使用した。

これも以下に構造を記載する。

なぜ inner join を活用できるのか、理解する為に役立てば幸いである。

【deptテーブル】

| dept_id | Name |
|---------|------|
| 1       | AX   |
| 2       | BX   |
| 3       |      |
| 4       | AX   |

【usersテーブル】

| user_id | dept_id | Name | grade |
|---------|---------|------|-------|
| 1       | 1       | AAA  | 3.5   |
| 2       | 1       | BBB  | 2.5   |
| 3       | 2       | CDef | 3.2   |
| 4       | 3       | cdEF | 4.5   |
|         |         |      |       |
| 5       | 5       | GGG  | 3.8   |



内部結合

残したい情報がどちらにもない場合、outer join である必要がない。  
Inner join でも可。

【deviceテーブル】

| dept_id | name | user_id | dept_id | name | grade | device_id | user_id | status | create_at   |
|---------|------|---------|---------|------|-------|-----------|---------|--------|-------------|
| 1       | AX   | 1       | 1       | AAA  | 3.5   | 1         | 1       | t      | [timestamp] |
| 1       | AX   | 2       | 1       | BBB  | 2.5   | 2         | 2       |        | [timestamp] |
| 2       | BX   | 3       | 2       | CDef | 3.2   |           |         |        |             |
| 3       |      | 4       | 3       | cdEF | 4.5   | 3         | 4       | f      | [timestamp] |
|         |      |         |         |      |       | 4         | 4       | t      | [timestamp] |
|         |      |         |         |      |       | 5         | 6       | t      | [timestamp] |



外部結合(left)

| dept_id | name | user_id | dept_id | name | grade | device_id | user_id | status | create_at   |
|---------|------|---------|---------|------|-------|-----------|---------|--------|-------------|
| 1       | AX   | 1       | 1       | AAA  | 3.5   | 1         | 1       | t      | [timestamp] |
| 1       | AX   | 2       | 1       | BBB  | 2.5   | 2         | 2       |        | [timestamp] |
| 3       |      | 4       | 3       | CDef | 3.2   | 3         | 4       | f      | [timestamp] |
| 3       |      | 4       | 3       | cdEF | 4.5   | 4         | 4       | t      | [timestamp] |
|         |      |         |         |      |       | 5         | 6       | t      | [timestamp] |

dept users device

deviceにしかないものは、deptともusersとも一致条件がない為、どちらの列も空になるはず。  
(どちらか一方でも埋まつていれば、deviceと他1つに存在する事になり、  
deviceのみに存在、という条件とはならない)  
 $=dept.dept_id \text{ is null and users.user_id is null}$ を指定

### ★ Q88 3つのテーブル結合時の取得範囲指定 3

本問題もこれまでと考え方は同じである。

full outer join で情報を落とさないようにしつつ、前問題で実施した各テーブルにしか存在しない情報を指定するものを or 条件で繋げると取得できる。

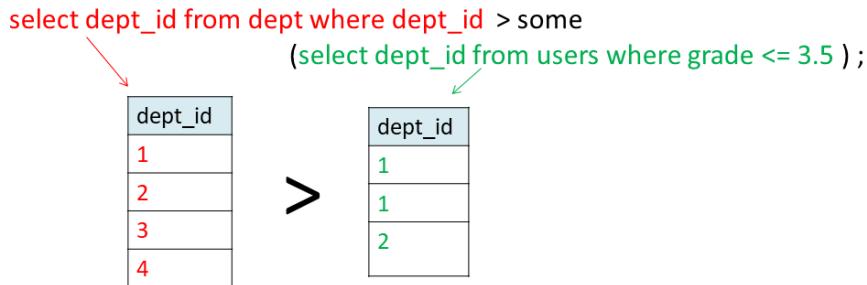
## ★ Q98 複数のレコードを条件とした副問合せ(some/any)

問題①から③を実施する際に比較演算子を変化させるだけでは上手くいかず、悩んだ方も多かったのではないだろうか。

欲しい情報は「3」と「4」だったが、比較演算子を「=」から「>」に変えるだけでは「2」「3」「4」の情報が取得されてしまう。

これにはsome/anyの条件に関する動作が関係している。

実務で使用しないのであればここまで理解しておく必要はないと考えるが、気になる方は以下を参照してほしい。



上記の時、副問合せ(緑字)で1と2より上の数値となる為、最終文(赤字)では3と4が残りそうに見える。

しかし、some/anyの条件は「条件のどれかに一致する事」である。

この場合、「2を超える(3と4)」と「1を超える(2,3,4)」の2つの条件がある。

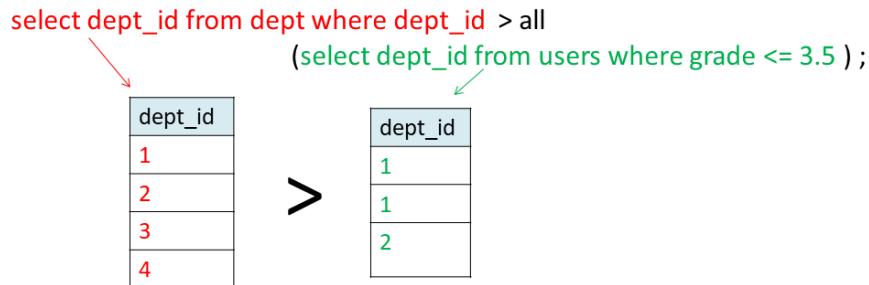
⇒ 上記SQLではこの通り、「2を超える」「1を超える」の両方を見たとき、どちらかに一致している値を全て返すという動きとなる。(2,3,4が返る)

⇒ 問題は副問合せで得た値より上の値を取得したい為、回答は  
max関数を使い、副問合せからの回答に「1」をいれず、  
2だけ取得する形にすることで対処している。  
(条件が「2を超える(=3,4)」だけの為、目的の数値が取れる)

## ★ Q99 some/any と対となる動作をする指定

some/any の対となる all に関する動作も以下に記載する。

some/any 同様、実務で使用しない場合、ここまで理解する必要はないと考えるが、気になる方はざっくり理解してもらえばと思う。



allの条件は「条件の全てに一致する事」が条件。

この場合、「2を超える(3と4)」と「1を超える(2,3,4)」の2つの条件がある。

⇒ 上記SQLではこの通り、「2を超える」「1を超える」の両方を見たとき、  
両方に一致している値を全て返すという動きとなる。(3,4が返る)

## ★ Q111 UPSERT1

「on conflict」句を使用するには、指定する列に主キー(primary key)か、一意性制約(unique key)が付与されている必要がある。

## ★ Q120 複合外部キー制約

本問題では複合一意性(unique)制約、複合外部キー制約を扱った。

複数列を指定する「複合」系は他にも複合主キーや複合 index などがある。

また、本書では様々な制約について扱ったが、他に「排他制約」と呼ばれるものも存在する。

興味があれば調べてみてほしい。

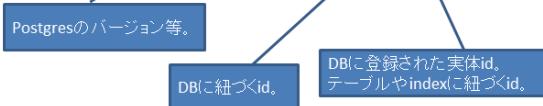
## ★ Q124 リストパーティションとテーブル空間の動作確認

## [補足] テーブル空間のディスクに置かれたファイルについて

- Q2-⑤にてテーブル空間に指定したディスクを確認したが、以下のような構造になつたかと思う。

※ 赤字がDBにテーブル空間設定を入れ、データを入れる事が増えたところ

/var/lib/postgresql/12/data01\_ts1/**PG\_12\_201909212/16391/16892**



- PG\_12\_～以下の謎の数値は**oid**と呼ぶ。  
oidと実際のDB名を紐づけを確認するには以下を実行する。

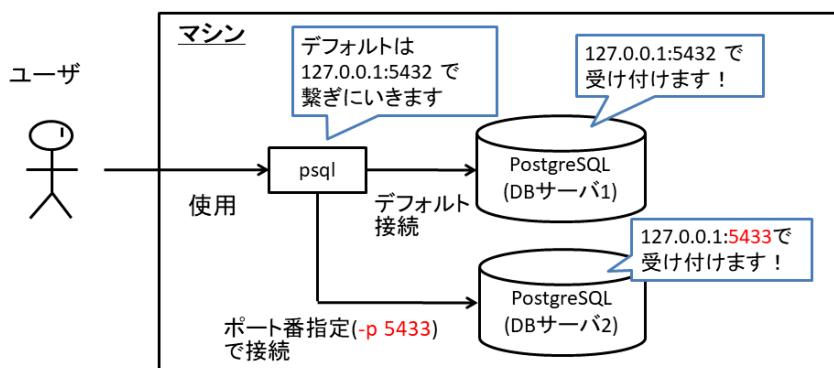
```
select oid, datname from pg_database where oid = 番号;
```

oidとテーブル情報等の紐づけを確認するには以下を実行する。  
select oid, relname, relkind from pg\_class where oid = 番号 ;

## ★ 事前準備(DB クラスタ追加作成)

事前準備のなかで実施した内容について補足する。

- 1つのマシンにDBクラスタは複数作成でき、複数立ち上げる事もできる。
- ただし複数立ち上げると、初期設定ではどこに繋げればいいかわからなくなってしまう。  
その為、接続先を区別する方法として「port番号」を変更した。  
⇒ PostgreSQLの対話モードに接続する際には  
IPアドレスとポート番号を使って接続する。  
psqlのデフォルト動作とpostgresql.confの初期設定は  
127.0.0.1(IPアドレス), 5432(ポート番号)でアクセスできるようになっている。  
⇒ 上記のIPアドレスをそのままとし、ポート番号を変更した形。



DB サーバを複数立ち上げると、関連して postgres のプロセスも複数起動する。

プロセスの確認方法は以下コマンドを実行すると良い。

```
ps -aux | grep postgres
```

上記コマンド結果を参考として以下に記載する。

```
postgres 1210792 0.0 0.3 217288 28000 ? Ss 17:22 0:00 /usr/lib/postgresql/12/bin/postgres -D /var/lib/postgresql/12/data01
postgres 1210794 0.0 0.0 217400 7968 ? Ss 17:22 0:00 postgres: checkpointer
postgres 1210795 0.0 0.0 217288 5932 ? Ss 17:22 0:00 postgres: background writer
postgres 1210796 0.0 0.1 217288 10156 ? Ss 17:22 0:00 postgres: walwriter
postgres 1210797 0.0 0.1 217956 8768 ? Ss 17:22 0:00 postgres: autovacuum launcher
postgres 1210798 0.0 0.0 71932 5560 ? Ss 17:22 0:00 postres: stats collector
postgres 1210799 0.0 0.0 217708 6940 ? Ss 17:22 0:00 postgres: logical replication launcher
postgres 1211651 0.3 0.3 217300 27824 ? Ss 17:36 0:00 /usr/lib/postgresql/12/bin/postgres -D /var/lib/postgresql/12/data02
postgres 1211653 0.0 0.0 217300 4308 ? Ss 17:36 0:00 postgres: checkpointer
postgres 1211654 0.0 0.0 217300 5856 ? Ss 17:36 0:00 postgres: background writer
postgres 1211655 0.0 0.0 217300 4308 ? Ss 17:36 0:00 postgres: walwriter
postgres 1211656 0.0 0.1 217840 8576 ? Ss 17:36 0:00 postgres: autovacuum launcher
postgres 1211657 0.0 0.0 71664 4952 ? Ss 17:36 0:00 postres: stats collector
postgres 1211658 0.0 0.0 217724 6984 ? Ss 17:36 0:00 postgres: logical replication launcher
```

postgres 関連のプロセス(\*)が複数(上記だと 2 セット)立ち上がっている事がわかる。

\* checkpointer や walwriter 等。ここでは各プロセスの役割は触れない。興味があれば調べてみてほしい。

2 セットあるのは data01 と data02 の DB クラスタを指定して立ち上げている為である。

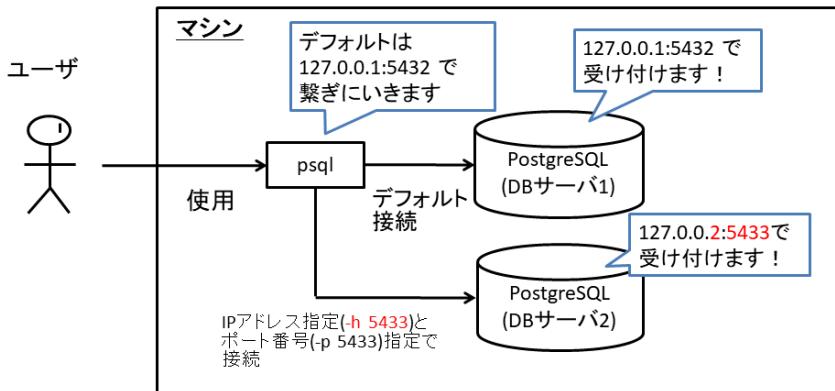
片方のみ起動している場合は半分となる。

この辺も余裕があれば実際に動かしてみてみると、より理解が深まるだろう。

また、本書ではポート番号を変更する手段をとっているが、IP アドレスを変える方法もある。

参考として以下にその内容を記載する為、気になれば確認してみてほしい。

- 本書では2つのDBサーバを立ち上げる為にポート番号の変更で対応したが、受け付けるIPアドレスを変更する案もある。  
その場合、DBクラスタ側はpostgresql.confのlisten\_addressesに127.0.0.2等の設定をして起動する。  
接続時は「psql」は「-h IPアドレス(-h 127.0.0.2)」を指定する。  
ただし同じマシン内で重複したポート番号の使用は許容されない為、  
本案でもポート番号の変更は必要である。  
⇒つまり設定変更箇所とpsqlで接続時に必要なオプションが増えるだけである。。。



なお、上記で説明している「listen\_addresses」はリモートにある PostgreSQL DB(自身の端末ではなく、別端末に postgres がインストールされており、自身の端末から psql で直接リモートにある DB)に接続する場合に、DB 側の接続許可設定をする項目である。

上記のようなリモート接続をする際に使う項目でもある為、覚えておけると良いだろう。

#### ★ Q119 キーワードに該当する情報がない場合の処理

### ファンクションに関する補足

- ファンクション内で、キーワードを使ったSQL文は、該当キーワードがないときは「null」として扱われる。

#### 【本問題の例】

##### ● ファンクション内の文

```
insert into tbl_history2(id, name) values (new.id , new.name);
```

↓ new.nameがなかった場合

```
insert into tbl_history2(id, name) values (new.id ,);
```

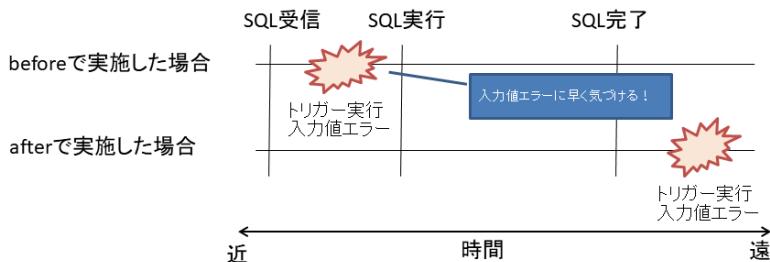
実行SQL文はこんな感じで  
欠けた状態となりうる。  
通常SQL実施時にはエラーとなる  
構文だが、関数ではnullに変換される。  
前問題③のように結合していた場合、  
一部でも存在しないものがあれば  
nullとなる。

- 空欄を避けつつ、履歴を残したい場合には  
処理されるユースケースごとにif文で条件を区切る必要がある
- なお、処理したSQL文を履歴として残したい、かつDBMSに頼りたい場合  
トリガーを使わずにログを使う、という手もある。  
※ DBMSではなく、アプリ側でログを出させるという手もある  
どちらを使うにしても、データ肥大化に注意する事。  
(レコードの定期削除やログのローテート(上書き)等の  
肥大化対策を考えておくこと)

## ★ Q125 テーブル間のズレの防止

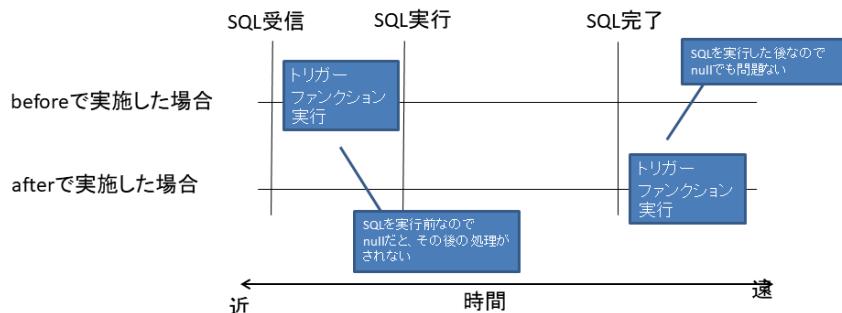
## トリガーのbefore, afterについて

- これまでではトリガー契機はafterだった。  
これまでの演習でもわかるようにafterのユースケースは、  
**対象テーブルの更新後の値集計等**がある。
  - beforeはSQLの処理がされる前にトリガーとファンクションを動かすもので、  
ユースケースとしては**処理対象の値(入力値)のチェック**などがあげられる。  
⇒ 本問題の1つ目の問題④は personテーブルのdeptを更新する前に、  
deptテーブルのnameにあるかどうかをチェックするケースの為、  
トリガーはbeforeを指定するのが良い。
- ※ なお処理結果はafterを選んでも同じ結果となる  
入力値チェックをbeforeでやれると良い理由は以下の通り。



## before/after指定時の関数の返り値について

- afterトリガーでは関数の戻り値をnullとしていた。(return null)
- beforeトリガーでは上記同様に戻り値をnullとするとUPDATE 0が発生する。  
上記を避ける為、戻り値にnewを指定する必要がある。(return new)  
⇒ beforeでreturn nullを指定するとトリガー処理後のSQL処理が止まる  
※ afterでは戻り値は重要ではないが、beforeでは意識する必要がある



## ファンクションのデバッグについて

- ・ プログラムやスクリプト等では上手く動かない時に「デバッグ」を行う。  
⇒ print文で途中の変数等に期待通りの値が入っているかチェックしたりする。
- ・ PL/PGSQLも長くなるとどこが上手くいっていないかわかりづらくなる。  
その為、処理の途中で変数の値などをチェックできると良い。  
⇒ エラー文の出力に使った「raise」を使用する。  
    raise noticeを指定する事で、特定変数の値を出力できる。

**【例：関数内に埋め込む文言】**

```
raise notice 'OLD.psn_cnt: %, NEW.psn_cnt: %', old.psn_cnt, new.psn_cnt;
```

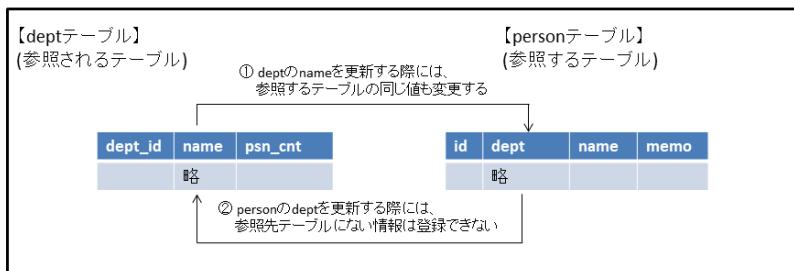
**【出力】**

```
NOTICE: OLD.psn_cnt: 2, NEW.psn_cnt: 10
```

（シングルクオート内の%が、  
シングルクオート後の指定変数と紐づく。  
(%の箇所に、指定変数の中身が出来られる)

## 外部キー制約とトリガーファンクションの使い分け

- ・ 本問題で実行した以下はテーブルに対する外部キー制約で同じ事ができる。



- ・ 外部キー制約、トリガーファンクションのどちらも他テーブルを参照する為、  
insert/update/deleteに影響を与える。
- ・ 外部キー制約は上記の①②の両方が必ず制約となる。  
ファンクションは① or ②と言う選択肢がとれる。  
⇒ 例えば、アプリ側で人の所属部署名はプルダウンで選択される  
(ユーザに自由に入力させない)等の対応をすると、②の制約はDB側では不要となる。  
一般的にはファンクション処理のほうが行ごと処理となる為、処理が重いが、  
とはいっても性能影響がある。  
アプリの動作やデータ諸元、更新頻度等も加味して、  
どう実装すると良いか考える必要がある。

★ Q126 やや複雑な入力値チェック

### チェック制約とトリガーファンクションの使い分け

- トリガーのbeforeは入力値チェックがユースケースとなるのは前述の通り。  
入力値チェックという観点でいえば、テーブル定義時の制約、チェック制約でも対応できる。
- チェック制約はシンプルな例は設定できるが、本問題のように複雑なチェック処理はできない。  
例えば本問題では「-2以下は入れない」という制約があった。  
このレベルであれば、チェック制約で実現可能である。

一方で0の時だけ-1が入るのを許可する、といった特殊条件はチェック制約で実現する事は難しい。

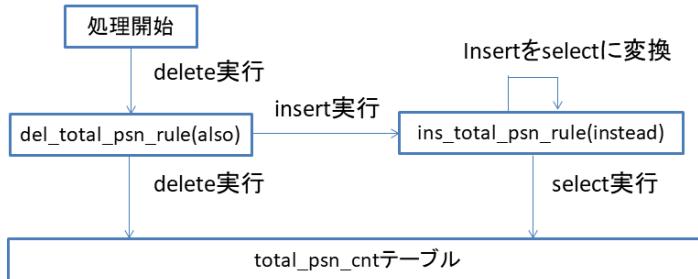
よって使い分けとしてはシンプルな内容でなければチェック制約を、  
チェック制約で対応できない複雑な制約を設ける場合は  
トリガーファンクションで実現する、という使い分けとなる。

---

## ★ Q129 ルールによる追加処理について

### 本問題-②が正しく動かなかった背景

- 一言でいうと、前問題で作ったルールと動きと連携してしまった為。
- 本問題-①完了時点では、total\_psn\_cntテーブルには  
insertを受けたらselectに変換するルール(前問題で作成, ins\_total\_psn\_rule)  
deleteを受けたらinsertも行うルール(本問題で作成, del\_total\_psn\_rule)  
の2つがあった。  
これにより、deleteを受けた後のinsert処理が、ins\_total\_psn\_ruleによる  
変換でselectに変えられていた。  
⇒ deleteだけがされる動きとなっていた。



### ルールに関する補足

- ルールはinsert/update/deleteを1つのルールでまとめて設定はできない。  
別々のルールを設けて設定する。(前問題の通り)
  - ファンクションはトリガーの起因となったSQLの処理前 or 処理後に実行する  
指定(実行タイミングの指定)ができた。  
ルール(also)にはその指定がない。  
⇒ 本問題-①と④の2パターンで試したが、どちらも期待通り動かなかった。  
原因はルール起動の要因となるSQLの処理により動きが変わるもの。  
(A) insertは、ルール起動SQLを先に実施し、その後、also指定の追加SQLを動かす。  
(B) update/deleteはalso指定の追加SQLの後に、ルール起動SQLが動く。
  - ⇒ 本問題-①のルールはdelete処理を受けたら、同じテーブルにinsert処理をする、  
というルールを設定していた。  
これは上記ルールの(B)が該当する。  
よってdelete処理を受けたら、先にinsert処理(also指定SQL)をして、  
その後にdelete(ルール起動SQL)を行う。
  - 同じように本問題-④ではinsert処理を受けたら、同じテーブルにdelete処理をする、  
というルールを設定していた。  
これはルール(A)に該当する。  
よって先にinsert(ルール起動SQL)を実行し、その後にdelete(also指定SQL)を実施する
- ※ 上記動作確認は「+α」問題にある為、余裕があれば  
動かして理解を深めてみてほしい。

### ファンクションによる代替とルールの使い分け

#### 【代替処理について】

- 特定のテーブルやビューに処理をしようとすると、自動で処理を変える仕組みが2つある。
  - トリガーファンクション(instead of)
  - ルール(instead)
- 上記2つの違いは、トリガーファンクションはビューのみに設定可能。テーブルにinstead ofトリガーを設定しようとするとエラーとなる。ルールはテーブルとビューどちらにも付与可能。
- また、トリガーファンクションは複雑な処理が指定可能だが、ルールはシンプルな変換となる。

#### 【追加処理について】

- 「ある処理を行うと、あわせて別処理を行う」場合も、トリガーファンクションとルール(also)で実施できる。これもやりたい事がシンプルか、複雑かで、使い分ける。

#### 【その他】

- トリガーファンクションの起動タイミングにはtruncateを指定できるが、ルールではtruncateは指定できない。  
truncate起因で何かしたい場合はトリガーファンクションを選択する。  
⇒ただしテーブルに対するtruncateを何か別の処理に変換する機能はないので注意。

★ Q138 generate\_series を使った試験データ投入

## generate\_seriesによる試験データ投入について①

- 最初はinsert into select文の為、説明は省略。  
select文について補足する

```
select person , action ,create_time from history , (select
generate_series(current_timestamp - interval '364 days' , current_timestamp -
interval '340 days' , '1 days') as create_time)a order by a.create_time ;
```

- 上記は赤字において、historyと副問合せaをクロスジョイン(紫部)している



- では以下で50行追加できるのはなぜか。(クロスジョインしていない)

```
select person , action , (generate_series(current_timestamp - interval '364 days' ,
current_timestamp - interval '340 days' , '1 days')) as create_time from history order
by create_time , person ;
```

## generate\_seriesによる試験データ投入について②

- 結論から記載すると、select の後に指定したカラム名(赤字)もfrom [table名]で指定した取得情報以外に、複数情報が加わるとクロスジョインのような動きとなる。

```
select person , action ,(generate_series(current_timestamp - interval '364 days' , current_timestamp -
interval '340 days' , '1 days')) as create_time from history order by create_time , person ;
```

- これまで見てきた数値を単純に指定する場合や集約関数(avg)は、from tableで受け取ったものに対して、1つしか情報がなかった。

### 例1) deptの結果に[1]を加える

```
select * , 1 from dept ;
```

| dept_id | name | ?column? |
|---------|------|----------|
| 1   AX  | 1    |          |
| 2   BX  | 1    |          |
| 3       | 1    |          |
| 4   AX  | 1    |          |

上記はdeptの結果(緑字)に対して、dept表にない追加情報(赤字)は一意(1で固定)

### 例2) usersのgradeを部署ごとに平均(avg)を出す

```
select user_id , avg(grade) from users group by user_id;
```

| user_id | avg               |
|---------|-------------------|
| 1       | 3.500000000000000 |
| 3       | 3.200000000000000 |
| 5       | 3.800000000000000 |
| 4       | 4.500000000000000 |
| 2       | 2.500000000000000 |

上記もusersの結果(赤字)に対して、user\_idごとに値は変わるもの追加情報(avg)は一意(user\_idに対して3.5とか4.0とか2つの値は取れない)

- generate\_seriesを埋め込むと上記前提が崩れる為、結果クロスジョインのような動きとなる

```
select * , generate_series(1, 2) from dept ;
```

```
dept_id | name | generate_series
-----+-----+-----+
1 | AX | 1
1 | AX | 2
2 | BX | 1
(以下略)
```

deptの結果に対して、追加情報は2つある(1と2)  
(例えはdept\_id 1に対して追加情報は1つだけではなく、1と2の2つある)

dept\_id 1  
... (他3行)

Generate\_series 1  
Generate\_series 2

= 8行

### generate\_seriesによる試験データ投入について③

- 前ページのパターンをgenerate\_series以外に試してみる
- selectの後に指定するカラム名にはテーブル名を指定することができる

例) dept表からdeptテーブルそのものを取得

| <code>select dept from dept ;</code> | <table border="1"> <thead> <tr><th>dept</th></tr> </thead> <tbody> <tr><td>-----</td></tr> <tr><td>(1,AX)</td></tr> <tr><td>(2,BX)</td></tr> <tr><td>(3,)</td></tr> <tr><td>(4,AX)</td></tr> </tbody> </table> | dept | ----- | (1,AX) | (2,BX) | (3,) | (4,AX) | ※ カラム名欄に指定するテーブル名はfrom以降に指定したテーブル名のみ取得可能。 | ※ 結果は* 指定と異なり、1レコード分を1列内に配列として表示 |
|--------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|-------|--------|--------|------|--------|-------------------------------------------|----------------------------------|
| dept                                 |                                                                                                                                                                                                                |      |       |        |        |      |        |                                           |                                  |
| -----                                |                                                                                                                                                                                                                |      |       |        |        |      |        |                                           |                                  |
| (1,AX)                               |                                                                                                                                                                                                                |      |       |        |        |      |        |                                           |                                  |
| (2,BX)                               |                                                                                                                                                                                                                |      |       |        |        |      |        |                                           |                                  |
| (3,)                                 |                                                                                                                                                                                                                |      |       |        |        |      |        |                                           |                                  |
| (4,AX)                               |                                                                                                                                                                                                                |      |       |        |        |      |        |                                           |                                  |

- 上記特性を使い、2つのテーブル情報をselectの後に指定する  
結果は同じようにクロスジョインのような動きとなった。

◆ テーブル名指定時

| <code>select dept , users from dept , users ;</code>                                                                                                                                                                                                                                                                                                                                     |              | [参考] *指定時 |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|-----------|----------------------|----------------------|--------------------|----------------------|------------------------|------------------------|-------|-------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|----------------------------|----------------------------|-------------------------|----------------------------|----------------------------|----------------------------|-------|
| <table border="1"> <thead> <tr><th>dept   users</th></tr> </thead> <tbody> <tr><td>-----</td></tr> <tr><td>(1,AX)   (1,AAA,3.5)</td></tr> <tr><td>(2,BX)   (1,AAA,3.5)</td></tr> <tr><td>(3,)   (1,AAA,3.5)</td></tr> <tr><td>(4,AX)   (1,AAA,3.5)</td></tr> <tr><td>(1,AX)   (2,1,BBB,2.5)</td></tr> <tr><td>(2,BX)   (2,1,BBB,2.5)</td></tr> <tr><td>(以下略)</td></tr> </tbody> </table> | dept   users | -----     | (1,AX)   (1,AAA,3.5) | (2,BX)   (1,AAA,3.5) | (3,)   (1,AAA,3.5) | (4,AX)   (1,AAA,3.5) | (1,AX)   (2,1,BBB,2.5) | (2,BX)   (2,1,BBB,2.5) | (以下略) | deptの結果行に対して、結果が一意でない。<br>(dept(1,AX)に対して、users情報は1つで<br>(少なく4)パターンある)<br>=クロスジョインの<br>ような動きになる | <table border="1"> <thead> <tr><th>dept_id   name   user_id   dept_id   name<br/>  grade</th></tr> </thead> <tbody> <tr><td>1   AX   1   1   AAA   3.5</td></tr> <tr><td>2   BX   1   1   AAA   3.5</td></tr> <tr><td>3     1   1   AAA   3.5</td></tr> <tr><td>4   AX   1   1   AAA   3.5</td></tr> <tr><td>1   AX   2   1   BBB   2.5</td></tr> <tr><td>2   BX   2   1   BBB   2.5</td></tr> <tr><td>(以下略)</td></tr> </tbody> </table> | dept_id   name   user_id   dept_id   name<br>  grade | 1   AX   1   1   AAA   3.5 | 2   BX   1   1   AAA   3.5 | 3     1   1   AAA   3.5 | 4   AX   1   1   AAA   3.5 | 1   AX   2   1   BBB   2.5 | 2   BX   2   1   BBB   2.5 | (以下略) |
| dept   users                                                                                                                                                                                                                                                                                                                                                                             |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| -----                                                                                                                                                                                                                                                                                                                                                                                    |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| (1,AX)   (1,AAA,3.5)                                                                                                                                                                                                                                                                                                                                                                     |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| (2,BX)   (1,AAA,3.5)                                                                                                                                                                                                                                                                                                                                                                     |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| (3,)   (1,AAA,3.5)                                                                                                                                                                                                                                                                                                                                                                       |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| (4,AX)   (1,AAA,3.5)                                                                                                                                                                                                                                                                                                                                                                     |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| (1,AX)   (2,1,BBB,2.5)                                                                                                                                                                                                                                                                                                                                                                   |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| (2,BX)   (2,1,BBB,2.5)                                                                                                                                                                                                                                                                                                                                                                   |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| (以下略)                                                                                                                                                                                                                                                                                                                                                                                    |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| dept_id   name   user_id   dept_id   name<br>  grade                                                                                                                                                                                                                                                                                                                                     |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| 1   AX   1   1   AAA   3.5                                                                                                                                                                                                                                                                                                                                                               |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| 2   BX   1   1   AAA   3.5                                                                                                                                                                                                                                                                                                                                                               |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| 3     1   1   AAA   3.5                                                                                                                                                                                                                                                                                                                                                                  |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| 4   AX   1   1   AAA   3.5                                                                                                                                                                                                                                                                                                                                                               |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| 1   AX   2   1   BBB   2.5                                                                                                                                                                                                                                                                                                                                                               |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| 2   BX   2   1   BBB   2.5                                                                                                                                                                                                                                                                                                                                                               |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |
| (以下略)                                                                                                                                                                                                                                                                                                                                                                                    |              |           |                      |                      |                    |                      |                        |                        |       |                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                      |                            |                            |                         |                            |                            |                            |       |

### ★ Q140 特定情報を除いた計算

#### coalesceとnullifについて

■ coalesceについて

**coalesce(引数1, 引数2)**

引数1においてnullがあれば引数2に置き換える  
nullのままだと計算されなくて不都合がある場合によく使用する。

```
select count(coalesce(a.name, 'xx')) from (select distinct(name) from dept) a ;
```

上記は赤字の副問合せの結果、nullの点をxxに置き換えてカウントしている



■ nullifについて

**nullif(引数1, 引数2)**

引数1において、引数2の値があればnullに変える(coalesceの逆パターン)  
特定の値を計算に入れたくない場合等によく使用する。

```
select avg(nullif(grade, -1)) from grade ;
```

# おわりに

この度は数ある書籍から本書を手に取っていただきありがとうございました。

また最後まで練習問題を解き切った方はお疲れさまでした。

これだけ SQL 潟けとなれば、苦手意識があつた方も苦手意識が吹っ飛んだのではないでしょか。

(ついでに意外と想定通り動かなくて事前確認の重要性を身に染みたかもしれませんね。)

本書で SQL や PostgreSQL の理解を深められたのであれば、本書を記載した者としてこれ以上嬉しい事はありません。

もう随分前ですが、筆者が入社して最初に配属された組織は(当社の中でも特に)技術コテコテの組織でした。結果的には良かったものの、当時はわからない事だらけ、しかも書いても書いても(SQLだけではないですが)エラーになるしで泣きそうになりながら 1つ1つの技術と戦っていました。書籍を読んでみようにも初步的なところで終わってしまったり、試験的な理解度チェック問題はあってもハンズオンはなかったのでわかった気になってしまったり、公式のマニュアルに助けを求めて読んでみようにも書いてある事が難しすぎて何言っているかわからなかったりしました。

自分は幸いにもチームや人にも恵まれ、なんとか乗り切れましたが、同じような境遇で技術に苦手意識をもつたり苦戦している人は沢山いるんだろうな、と思い、そういう方やこれから次世代を担う初心者が同じような苦労をしないように、なるべくわかりやすく・かつ動かして試せるものがあると良いなと思い、用意したのが本書となります。

もし読者の皆様が役に立つと感じたら、是非まわりの人に広めて頂ければ、と思います。

また、読んだ感想や誤りのご指摘、ご要望等ございましたら是非ともお寄せください。

皆様のご意見を取り入れて、本書をよりよくしていきたいと思ってあります。

また、当社(NTT テクノクロス)は技術ブログ([情報畠でつかまえて<sup>\\*1</sup>](#))も執筆しております。

最初に触れた「NoSQL」や「NewSQL」に関する記事や PostgreSQL のより深い技術内容に関する記事の他、様々な情報を発信しています。

こちらもご興味があれば是非答えのダウンロードとあわせてご覧頂ければ、と思います。

最後に当社は技術書展にて本書以外にも様々な書籍([書籍一覧<sup>\\*2</sup>](#))を出版しています。

こちらもご興味があればご確認ください。

ここまでご覧いただきありがとうございました。

---

1. <https://www.ntt-tx.co.jp/column/>

2. <https://techbookfest.org/organization/5452280578965504>

## 著者紹介

山口 佳輝([yamaguchi.yoshiki@ntt-tx.co.jp](mailto:yamaguchi.yoshiki@ntt-tx.co.jp))

ネットワークを最適に扱う為のシステム開発や検討業務に従事しています。

また、[Qiita<sup>\\*3</sup>](#) や当社技術ブログ([情報畠でつかまえて<sup>\\*4</sup>](#))での情報発信や、当社内での DB に関する研修、PostgreSQL に関する PJ 支援(質問回答)等も行っています。

最近のちょっとした自慢は、最近に取得した IT 関連の資格が 15 個を超えたことです。(IPA の高度資格から、NW・Linux・PostgreSQL(DB)等のベンダー資格など幅広にやってますが、共通して取得を目的にするのではなく実務で活かす事を目標にしています)

趣味はロードバイクと登山。乗鞍岳と甲斐駒ヶ岳にいきたい。

★ 表紙デザイン：足達 俊雅

---

3. [https://qiita.com/yamaguchi\\_yy449/items/4ffae4ef334e741edb37](https://qiita.com/yamaguchi_yy449/items/4ffae4ef334e741edb37)

4. <https://www.ntt-tx.co.jp/column/>