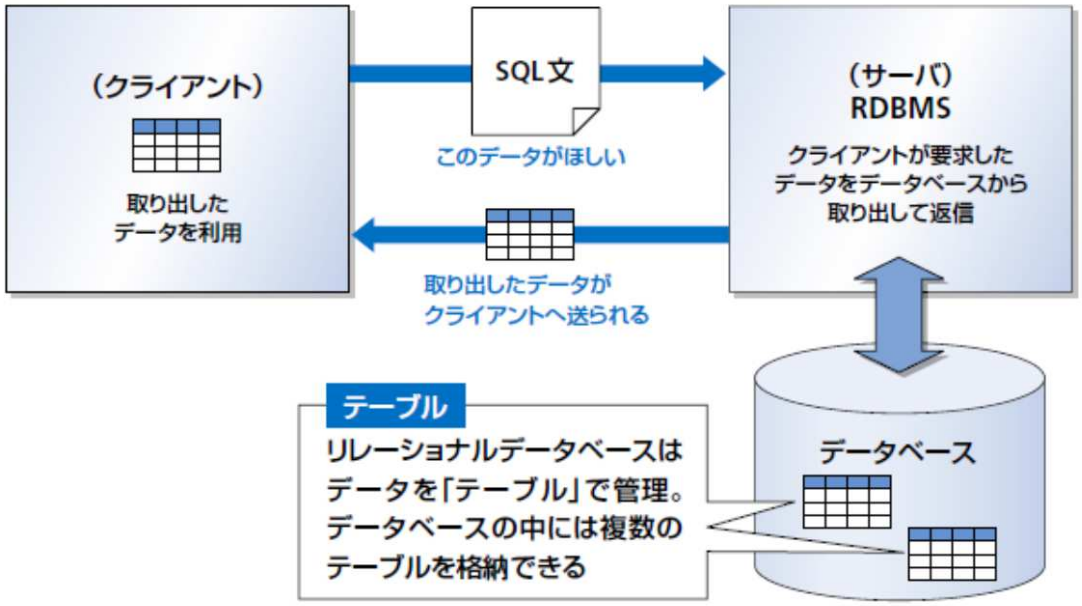
Q1 データベースの概念図を示せ

A1

システムの概念図（教科書図1-5）



Q2 DBMS とは何の略か

Q3
SQL の基本ルール
文は(ア)で区切る
(イ)は、大文字と小文字を区別しない
文字列は(ウ)
単語は、(エ)で区切る

Q4
データベースとテーブルの関係を述べよ
Q5
データベース「shop」内に表 5.1 のようなテーブル「商品」を作りたい。
テーブル作成に成功したことを確認するところまで実行するための SQL 文(および手順)を穴埋めせよ。

表 5.1 空の「商品」テーブル

商品id [PK] character (4)	商品名 character varying (100)	商品分類 character varying (32)	販売単価 integer	仕入単価 integer	登録日 date

1. pgAdmin4 をブラウザで起動する。
2. postgres を選択し、稲妻アイコンをクリックすると SQL 文の入力画面が表示される。
3. 「(ア)」と入力し、入力画面の稲妻アイコンから実行する。
4. ブラウザをリフレッシュする。
5. データベース shop を選択した上で SQL 文入力画面を表示する。
6. 次のように入力する。

```
(イ)
(
    (ウ),-- 4 文字の文字列「商品 ID」。null 以外。
    (エ),--最大 100 文字の文字列「商品名」。null 以外
    商品分類 VARCHAR(32) NOT NULL,
    (オ),--整数「販売単価」
    仕入単価 INTEGER,
    登録日（カ),
    (キ)
);
```

7. shop→スキーマ→テーブル→商品→右クリック→「データの閲覧/編集」

A2 データベースマネージメントシステム

A3
(ア)セミコロン
(イ)キーワード
(ウ)一重引用符で囲む
(エ)ホワイトスペース（空白、タブ、改行）

A4
データベースがテーブルを所有する。

A5
(ア)CREATE DATABASE shop;
(イ)CREATE TABLE 商品
(ウ)商品 ID CHAR(4) NOT NULL
(エ)商品名 VARCHAR(100) NOT NULL,
(オ)販売単価 INTEGER
(カ)DATE
(キ)PRIMARY KEY (商品 ID)

Q6
CREATE TABLE 文について穴埋めせよ。
命名ルール: テーブル名や(ア)名などの規則は、一般的なプログラミング言語と同じ。但し(イ)なら日本語可能。
主なデータ型:
数値型
┣整数: INTEGER
┣固定小数点: (ウ)
┗浮動小数点: (エ)
文字列型
┣固定長文字列: CHAR
┣可変長文字列: VARCHAR
┗長さ制限なしの可変長文字列: (オ)
日時型
┣(カ): DATE
┣(キ): (ク)
┗カとキ: (ケ)
PRIMARY KEY 制約: (コ)

A6

(ア): 列

(イ): UTF-8

(ウ): NUMERIC

(エ): REAL

(オ): TEXT

(カ): 日付

(キ): 時刻

(ク): TIME

(ケ): TIMESTAMP

(コ): NOT NULL 制約 かつ、重複不可

Q7
「商品」テーブルを削除する SQL 文

Q8
穴埋めせよ。
テーブル「商品」に列「カナ商品名」を追加する SQL 文の例は、
(ア) 商品 (イ) カナ商品名 VARCHAR(100);
である。
(ア)文でできるのは列の追加だけではない。
((列|制約)の(追加|削除))|(データ型の変更)ができる。
但し、(ウ)リスクを考え、列の追加の目的で使うにとどめることが多い。

Q9
表 5.1 のような「商品」テーブルに、表 9.1 のようにデータ(行)を追加する SQL 文を穴埋めせよ。

表 9.1　2 つのデータを持つ「商品」テーブル

	商品id [PK] character (4)	商品名 character varying (100)	商品分類 character varying (32)	販売単価 integer	仕入単価 integer	登録日 date
1	0001	Tシャツ	衣服	1000	500	2009-09-20
2	0002	Yシャツ	衣服	2000	1500	2009-09-21

(ア) 商品 (イ) '0001', 'T シャツ', '衣服', 1000, 500,(エ);
(ア) 商品 (イ) '0002', 'Y シャツ', '衣服', 2000, 1500,(エ);

Q10
SELECT 文の基本構成は
(ア)である。(イ)句は省略できるといえども、実際にテーブル内のデータにアクセスしたい場合は省略できない(「デフォルト(イ)」や「カレント(イ)」のような概念があるわけではない)。
(イ)句を省略しても動作する例は(ウ)など、電卓代わりに使う時などだ。
今、SELECT 文を使って、商品 id が 0002 に一致するデータの、商品名および売れた場合の利益(販売単価-仕入単価)を取得する SQL 文を示すなら、
SELECT (エ) FROM 商品 WHERE (オ);
となる。

A7
DROP TABLE 商品; --テーブルの定義情報を含めすべて削除
TRUNCATE TABLE 商品; --テーブルの定義情報は残し、すべての行を一括削除
DELETE FROM 商品; --テーブルの定義情報は残し、(where 句による条件に一致した)行を一括削除

A8
(ア): ALTER TABLE
(イ): ADD COLUMN
(ウ): 既存のデータおよびアプリケーションに影響を及ぼす

A9
(ア): INSERT INTO
(イ): VALUES (
(ウ): '2009-09-20')
(エ): '2009-09-21')
太字部忘れるな

A10
(ア): SELECT 列名 (FROM テーブル名)? (WHERE 行の条件式)?
(イ): FROM
(ウ): SELECT 60*60*24;
(エ): 商品名, 販売単価-仕入単価 as 売れた場合の利益
(オ): 商品 id= '0002'
(等号は==でなく=なので注意)

Q11
テーブル全体を取得するが、列「商品分類」における重複を除去する SELECT 文を述べよ。

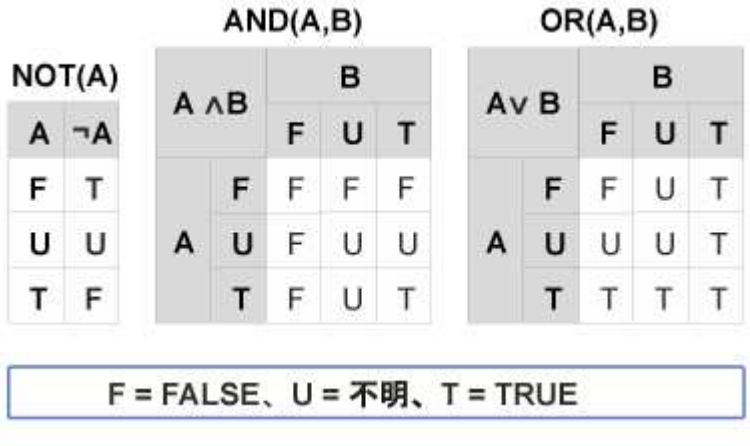
Q12
SELECT 文の WHERE 句などで使う「条件式」で、
列 A が null(以外)であることを条件とする式をいえ。
誤った条件式も挙げよ(どうなるかもいえ)。

Q13
三値論理とは何か。また and や or の三値論理値表を示せ

A11
そのようなことは(多分)できない。
重複を除去できる DISTINCT は
~~SELECT * FROM 商品 DISTINCT 商品分類;~~
のように使うことはできず、
SELECT DISTINCT 商品分類 FROM 商品;
のようにする。しかし、こうすると、「商品分類」列しかとりだせない。

A12
正しい条件式
A is null または A is not null
誤った条件式
A == null または A != null や A <> null
これらはいずれも何もマッチしない(つまり 0 行となる)

A13
true, false に null を加えたもの。null は「true か false かわからない状態」
論理演算の結果はなるべく null を避けるが、やむを得ない場合は null となる。
例えば、片方が true なときに or をとれば、たとえもう片方が null であろうとも、結果は true にできる。



https://www.ibm.com/support/knowledgecenter/ja/SSQP76_8.9.1/com.ibm.odm.itoa.develop/topics/con_unknown_values.html

Q14
集約関数とは(ア)に対して演算を行い、単一の結果を返す関数のことである。
(イ)・・・　列のデータ数即ちテーブルの(ウ)を数える
(エ)・・・(オ)
AV(カ)・・・(キ)の平均を求める
MAX・・・テーブルの列のデータの最大値を求める
MIN・・・テーブルの列のデータの最小値を求める。
集約関数の基本ルールとして、(ク)るというものがある。但し、(ケ)は例外で、(ク)ない。
また、(コ)が指定可能だ。

Q15
表 9.1 のようなテーブル「商品」から、表 15.1 のような出力を得るための SQL 文を述べよ。

表 15.1 最新商品およびその登録日

	最新商品 character varying (100)	登録日 date
1	Yシャツ	2009-09-21

Q16
PostgreSQL 独自の主な集約関数

関数	機能
	相関係数
	母共分散
	標本共分散
	母標準偏差
	標本標準偏差
	母分散
	標本分散
	線形回帰式のY切片
	線形回帰式の傾き

A14
(ア): 列
(イ): COUNT
(ウ): 行数
(エ): SUM
(オ): テーブルの数値列のデータの合計を求める
(カ): G （E ではないので注意）
(キ): テーブルの数値列のデータ
(ク): NULL 値は除外され
(ケ): COUNT(*)
(コ):DISTINCT

A15
select 商品名 as 最新商品, 登録日 from 商品 where 登録日=(select max(登録日) from 商品)

select 文はよくネストできないといわれるが、select を毎回つければ実はネストできる。要は句の直接ネストはできないが、句に文をネストすることができる。(少なくとも自分の環境ではそうだった。)
また、**集約関数の入れ子はダメ**

A16

関数	機能
corr(Y, X)	相関係数
covar_pop(Y, X)	母共分散
covar_samp(Y, X)	標本共分散
stddev_pop(X)	母標準偏差
stddev_samp(X)	標本標準偏差
variance_pop(X)	母分散
variance_samp(X)	標本分散
regr_intercept(Y, X)	線形回帰式のY切片
regr_slope(Y, X)	線形回帰式の傾き

Q17
SELECT 文には GROUP BY 句というものを持たせることができる。
これは、出力するテーブルの(ア)を、(イ)を基準にまとめる機能である。

Q18
表 18.1 のようなテーブル「商品」があるとする。

表 18.1　5 つのレコード(行)をもつテーブル「商品」

	商品id [PK] character (4)	商品名 character varying (100)	商品分類 character varying (32)	販売単価 integer	仕入単価 integer	登録日 date
1	0001	Tシャツ	衣服	1000	500	2009-09-20
2	0002	Yシャツ	衣服	2000	1500	2009-09-21
3	0003	鉛筆削り	文房具	150	100	2009-09-21
4	0004	鉛筆1ダース	文房具	100	60	2009-09-21
5	0005	消しゴム	文房具	50	20	2009-09-22

次の SQL 文に対してそれぞれどのような表が返ってくるか述べよ
SELECT 登録日, count(*) FROM 商品 GROUP BY 登録日;
SELECT 商品分類, count(*) FROM 商品 GROUP BY 商品分類;
SELECT 商品分類, * FROM 商品 GROUP BY 商品分類;

Q19
GROUP BY 句をもつ SELECT 文には HAVING 句を持たせることができる。
これは、(ア)を定めるものである。HAVING 句に指定できるのは(イ)である。
例えば、表 18.1 のようなテーブル「商品」から、表 18.2 のようなテーブルで登録日が 2009-09-20 より新しいもののみを抽出したい場合、つまり表 19.1 を得たい場合、SQL 文は「(ウ)」とする。

表 19.1

	登録日 date	count bigint
1	2009-09-22	1
2	2009-09-21	3

A17

(ア): 行

(イ): 集約関数と、GROUP BY 句に指定した列

(ウ): ある列における値

A18

表 18.2「SELECT 登録日, count(*) FROM 商品 GROUP BY 登録日;」の結果

	登録日 date	count bigint
1	2009-09-20	1
2	2009-09-22	1
3	2009-09-21	3

表 18.3「SELECT 商品分類, count(*) FROM 商品 GROUP BY 商品分類;」の結果

	商品分類 character varying (32)	count bigint
1	文房具	3
2	衣服	2

「SELECT 商品分類, * FROM 商品 GROUP BY 商品分類;」の結果

→エラーになる。

GROUP BY を指定した場合、SELECT 句には GROUP BY で指定の列か、集約関数しか書けない。

A19

(ア): グループとしての各レコードの出力条件

(イ): select 登録日, count(*) from 商品 group by 登録日 having 登録日>'2009-9-20'

Q20
SELECT 文による出力データは並び替えることができる。これをするためには(ア)句以降に(イ)句を入れる。行を 1 つ以上指定し、デフォルトでは昇順で並び替える。昇順であることを明示するのは(ウ)、降順にするのは(エ)である。(ウ)や(エ)は各行の直後に入れる。
複数の行を指定した場合は、先に指定した行を優先して並び替えを行う。
https://www.dbonline.jp/mysql/select/index11.html

Q21
SELECT 文による出力データに対して出力範囲を指定することができる。これをするためには(ア)句と(イ)句を使う。(ア)句に(ウ)を指定することで(最大)出力数を決め、
(イ)句に(ウ)を指定することで、(エ)を決める。
したがって、「5 件目から 7 件目が欲しい」場合は、
「(ア) (オ) (イ) (カ)」を文の後ろに追加する。

Q22
FROM, GROUP BY, HAVING, LIMIT, ORDER BY, SELECT, WHERE
句を SELECT 文として適切に並び替えよ

Q23
Q22 の句を、実行順に並び替えよ

Q24
あれば違反を指摘せよ。

1. SELECT 注文日, AVG(数量) FROM 注文明細;
2. SELECT 注文日, AVG(SUM(数量)) FROM 注文明細 GROUP BY 注文日;
3. SELECT 注文日 FROM 注文明細 WHERE SUM(数量) > 1000 GROUP BY 注文日;
4. SELECT 注文日, AVG(数量) FROM 注文明細 GROUP BY 注文日;

A20
(ア): FROM
(イ): ORDER BY
(ウ): ASC
(エ): DESC

A21
(ア): LIMIT
(イ): OFFSET
(ウ): 非負な整数
(エ): 先頭から何件のレコードを飛ばすか
(オ): 3
(カ): 4

A22
SELECT→FROM→WHERE→GROUP BY→HAVING→ORDER BY→LIMIT

省略可能性については、各社 sql によって違いがあるようだ
(from がないと where 以降が指定できない場合もあれば、
select true where true;が可能な場合もあるし、
そもそも from が必須の環境もあるように思える。)

A23
A22 で、SELECT が ORDER BY と LIMIT の間に移動するだけだ。

A24
1. 「注文日」は複数行、AVG は 1 行なので行数に食い違いが出るので違反
2. 「注文日」は GROUP BY で指定されているため 1 行なので、この点に問題はない。
が、集約関数のネストが違反。
3. WHERE 句は行を選択するので、集約関数を WHERE 句に入れてしまうと「全選択か 0 行か」になってしまう。違反。(本当に文法的にエラーなのかな？)
4. 合致

Q25
表 5.1 や表 9.1 のようなテーブル「商品」に、レコードを追加したい。
Q9 のようにすれば追加できるが、その場合は VALUES 内の「値リスト」は列の定義順に並べる必要がある。
(ア)を明示することで、この制約から解放される。そういった書き方の例は「(イ)」。
(ア)については、NULL が可能な場合や(ウ)値が与えられている場合は、その列を(エ)できる。
(ア)と値リストは基本 1 対 1 対応だが、値リストは(オ)可能。
(オ)された場合、(ウ)が入れられるが、その列で(ウ)が定義されていなければ NULL となる。
また、(ウ)や NULL は値リスト内で明示してよい。

Q26
既存テーブルへ、別のテーブルをコピーする SQL 文の例を示せ

Q27
新規テーブルに、別のテーブルをコピーする SQL 文の例を示せ

Q28
表 18.1 で、販売単価が 1000 円以上のものを削除する SQL 文を示せ

Q29
表 18.1 で、販売単価が 1000 円以上のものを 1 度だけ半額にする SQL 文を示せ

A25
(ア): 列リスト
(イ): INSERT INTO 商品(商品 ID, 商品名, 商品分類, 販売単価, 仕入単価, 登録日)VALUES ('0001', 'T シャツ', '衣服', 1000, 500, '2009-09-20');
(ウ): DEFAULT
(エ): 省略
(オ): 最後だけ省略(多分複数でも OK)

A26
INSERT INTO 商品新(商品 ID, 商品名, 商品分類, 販売単価, 仕入単価, 登録日)
SELECT 商品 ID, 商品名, 商品分類, 販売単価, 仕入単価, 登録日 FROM 商品;

A27
CREATE TABLE 商品新 AS SELECT 商品 ID, 商品名, 商品分類, 販売単価, 仕入単価, 登録日 FROM 商品;
または
CREATE TABLE 商品新 AS SELECT * FROM 商品;

A28
delete from 商品 where 販売単価>=1000;

A29
UPDATE 商品 SET 販売単価= 販売単価/2 WHERE 販売単価>= 1000;

Q30
SQL 文の(ア)(つまり(イ))を(ウ)という。
操作の前後で通信に障害が出たなどして、データベースに矛盾が出てしまうと困るため、これを防ぐ技術だ。
(ウ)では、(エ)、(オ)、(カ)、(キ)の 4 つの性質がある。
(エ)・・・アプリケーションから見たときのデータベース処理の最小単位。
データ操作は(ク)。
(オ)
(カ)・・・(ケ)。隔離性水準が SERIALIZABLE の場合の処理結果は，トランザクションを逐次処理した場合と同一になる（別途)。
(コ)・・・いったん確定したデータ操作は，その後の障害等で消滅しない。

Q31
トランザクションの状態遷移図を示せ

Q32
PostgreSQL でトランザクションを明示する方法をいえ。

A30

(ア): シーケンス

(イ): 一連のデータベース操作

(ウ): トランザクション

(エ): 原子性

(オ): 一貫性

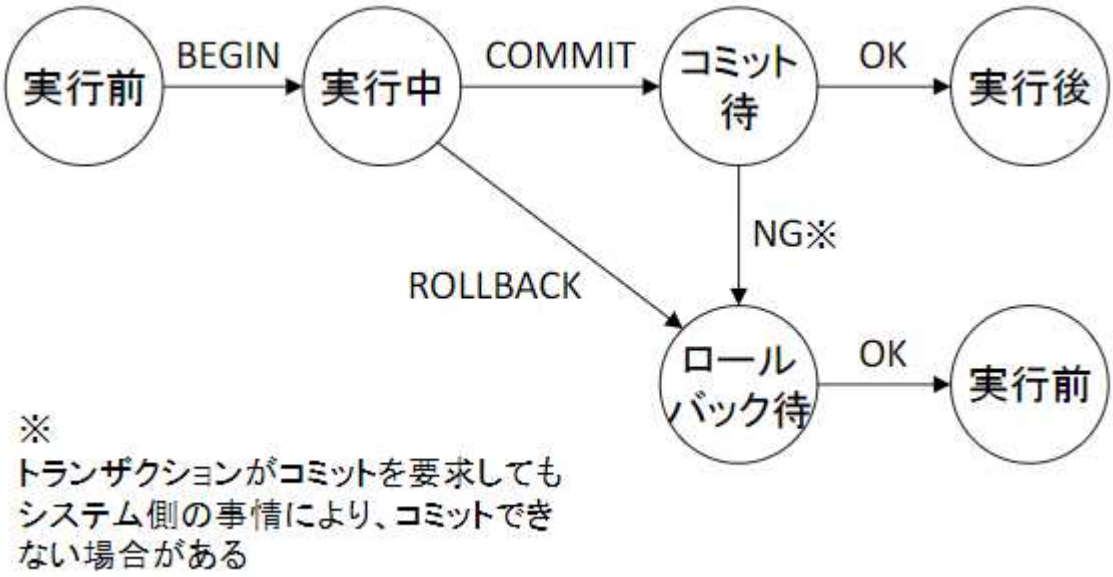
(カ): 独立性

(キ): 永続性

(ク): すべて反映されるか，取り消されるかのいずれか(「コミット（commit)」か「ロールバック（rollback)」)

(ケ): 同時に実行されている他のトランザクションの影響を受けない

A31



A32

BEGIN TRANSACTION; と COMMIT;文でかこむ

Q33
問題点を指摘せよ。
INSERT INTO 商品 VALUES ('0010', 'ソックス', '衣服',, 500);

Q34
ビューとは(ア)である。(イ)時に動的に生成されるため静的な実体はない。
(ウ)に制限がある。

Q35
ビューを使う利点を 4 つ挙げよ

Q36
表 18.1 の通りのテーブル「商品」から、表 36.1 の通りのビュー「商品合計」を得るための SQL 文を言え。

表 36.1 ビュー「商品合計」

	商品分類 character varying (32)	商品数 bigint
1	文房具	3
2	衣服	2

Q37
ビューの基本ルール。
(ア)句を除く任意の select 文を使用できる。
select 句には(イ)も指定可能。
from 句には(ウ)も指定可能
view の列リストを省略した場合、(エ)。

Q38
特定の条件を満たせば、ビューに対して更新を試みることで、ビューの元となるテーブルに対する更新を行える。
・(ア)からつくられていること。
・(おそらく as 直後の select 文で)(イ)が使用されていないこと
・(おそらく as 直後の)select 句に(ウ)が使われていないこと

A33
途中は省略できない。(最後に寄せればよい)

A34
(ア): 仮想的なテーブル
(イ): SQL 文の実行
(ウ): 更新

A35
・記憶容量の節約
・SQL 文の記述の効率化
・データベースとアプリケーションプログラムの独立性向上
・セキュリティの向上(アクセス制限)

A36
CREATE VIEW 商品合計(商品分類, 商品数)AS SELECT 商品分類, COUNT(*) FROM 商品 GROUP BY 商品分類;

A37
(ア): order by
(イ): 関数や式
(ウ): 複数のテーブル、ビュー
(エ): select 句で選択された列名が使用される

A38
(ア): 単一のテーブル
(イ): distinct, group by, havig, limit
(ウ): 関数や式

Q39
サブクエリとは(ア)である。必ず(イ)。
サブクエリを使うと、ビューが不要になる。
例えば「(ウ|2 つの sql 文)」を、サブクエリを用いずに 1 つの sql 文で書くと、
「

```
select  商品分類,  商品数
from
(
      select  商品分類, count(*) as  商品数
      from  商品
      group by  商品分類
)
```

」となる。

Q40
(ここでいう「サブクエリ」は、内側の select 文のこととする)
サブクエリを挿入できる場所にはいくつか種類があり、その種類は(ア)。
(イ)→from 句で指定可能
(ウ)→where 句で指定可能
(エ)→from 句以外のどこでも可能

Q41
関数の種類
(ア)関数（ABS, CEIL, FLOOR, MOD, ROUND, SIN, COS, SQRT 等）,
文字列関数,
日付・時刻関数（CURRENT_DATE, CURRENT_TIME,(イ)等）,
集約関数,
文字列の連結（例（ウ|'abc'と'def'を連結せよ))
型変換（例（エ|'123'を INTEGER 型に変換せよ))：⇒123
NULL 除去：(オ)(NULL, '123', NULL) ⇒'123'
のうち、列方向に働く(＝複数レコードを対象とする)関数は(カ)だ。それ以外は行方向に働く(＝単一レコードを対象とする)。
また、関数は原則として、引数に null があると、(キ)。

A39
(ア): 入れ子になった SQL 文(における内側の文)
(イ): 括弧で囲む
(ウ):

```
create view tmp(商品分類, 商品数)
as
    select 商品分類, count(*)
    from 商品
    group by 商品分類;

select * from tmp;
```

A40
(ア): サブクエリの結果の形状によってきまる
(イ): 結果が複数行、複数列の表になりうる
(ウ): 結果の列数は必ず 1 になる
(エ): 結果は必ず 1 行 1 列になる(スカラサブクエリ)

A41
(ア): 算術
(イ): EXTRACT (日時から何月かを求めるなど)
(ウ): 'abc' || 'def'
(エ): CAST('123' AS INTEGER)
(オ): COALESCE
(カ): 集約関数のみ
(キ): 結果は null になる

Q42
次の sql 文は 1 行複数列のテーブルを返す。各列のデータ型と値を述べよ。(要は暗黙のキャストや固定小数点の精度を問うている)
select 1+1.0, 1+'1', concat(1, '1'),1/3,1.0*3.0;

Q43
SQL では(ア)を述語という。主に(イ)句で使われる。
スカラ値を扱う述語・・・
(ウ)
(エ)
文字列の部分一致(オ)
NULL 値の判定(IS NULL, IS NOT NULL)

集合を扱う述語・・・
OR 条件に対応((カ))
AND 条件に対応((キ))
存在条件に対応(EXISTS)

Q44
述語 LIKE では、ワイルドカードが使える。ワイルドカード文字「_」「%」をそれぞれ説明せよ。

A42

表 42.1 を得る。(但し、実際には各列に as 句を追加した sql 文を実行して得たものである)

表 42.1

	integer＋numeric numeric	integer＋text integer	concat text	integer／integer integer	numeric＊numeric numeric
1	2.0	2	11	0	3.00

整数同士の割り算の結果が整数なのか小数点なのかは DBMS 依存。

小数点の精度は「最低限必要なだけ(有効数字とは別)」

A43

(ア): 戻り値や結果が真偽値となる条件式

(イ): where

(ウ): 比較演算子（=, <, >, <=, >=, <>）

(エ): 範囲指定（BETWEEN）

(オ): LIKE

(カ): IN, ANY

(キ): ALL

A44

「_」・・・任意の 1 文字

「%」・・・任意の、任意の長さの文字

Q45

表 45.1 および表 45.2

商品

商品ID	商品名	商品分類	販売単価	仕入単価	登録日
0001	Tシャツ	衣服	1000	500	2009-09-20
0002	穴あけパンチ	事務用品	500	320	2009-09-11
0003	カッターシャツ	衣服	4000	2800	
0004	包丁	キッチン用品	3000	2800	2009-09-20
0005	圧力鍋	キッチン用品	6800	5000	2009-01-15
0006	フォーク	キッチン用品	500		2009-09-20
0007	おろしがね	キッチン用品	880	790	2008-04-28
0008	ボールペン	事務用品	100		2009-11-11

店舗商品

店舗ID	店舗名	商品ID	数量
000A	東京	0001	30
000A	東京	0002	50
000A	東京	0003	15
000B	名古屋	0002	30
000B	名古屋	0003	120
000B	名古屋	0004	20
000B	名古屋	0006	10
000B	名古屋	0007	40
000C	大阪	0003	20
000C	大阪	0004	50
000C	大阪	0006	90
000C	大阪	0007	70
000D	福岡	0001	100

表 45.1 および表 45.2 のように 2 つのテーブルがあるとして、次の sql 文を実行した結果、得られるテーブルを求めよ。

```
select 商品名, 販売単価
from 商品
where 商品 id in
(
    select 商品 id
    from 店舗商品
    where 店舗 id= '000C'
);
```

Q46

「BOOKS」表から書名の一部に「UNIX」という文字列を含む行を全て探す SQL 文をいえ

A45

まず in 内のサブクエリにより、(「店舗商品」テーブルから)商品 id 0003, 0004, 0006, 0007 から成る 4 行 1 列のテーブルが抽出される。この中に商品 id があるような行を、「商品」テーブルから取り出すので、最終的に表 45.3 を得ることになる。

表 45.3

	商品名 character varying (100)	販売単価 integer
1	カッターシャツ	4000
2	包丁	3000
3	フォーク	500
4	おろしがね	880

A46

SELECT * FROM BOOKS WHERE 書名 LIKE '%UNIX%'

Q47
次の sql 文は文法的に正しい。
SELECT * FROM (SELECT * FROM 商品 WHERE 販売単価 >= 4000) AS 高額商品;
「select from as」という構文がどのような働きをするか述べよ。またそれはどのようなときに有用か。例も示せ。

Q48
次の sql 文の違反点を指摘せよ
SELECT * FROM 商品 WHERE 商品 ID = (SELECT 商品 ID FROM 商品 WHERE 販売単価 >= 4000);

Q49
CASE 式では、SQL で条件分岐を実現することができる。
例えば表 9.1 のようなテーブル「商品」にたいして
select (ア) as アルファベット from 商品;
あるいは
select (イ) as アルファベット from 商品;

を実行すると、表 49.1 を得る。
(ア)は単純 case 式と呼ばれ、可読性が高いが、比較演算子「=」に限られる。
(イ)は検索 case 式と呼ばれ、複雑な条件判定が可能だ。

表 49.1

	アルファベット text
1	B:文房具
2	B:文房具
3	B:文房具
4	A:衣服
5	A:衣服

A47
select した結果としてのテーブルに別名を付ける。
複数のテーブルから select するときに有用。
select A.column1, B.column2 from tableA as A, tableB as B;

A48
スカラ値を扱う述語「=」の右辺が、スカラサブクエリでない(複数行の結果を返すサブクエリだ)。

A49
(ア):

```
case 商品分類
    when '衣服' then 'A:' || 商品分類
    when '文房具' then 'B:' || 商品分類
    else null
end
```

(イ):

```
case
    when 商品分類 = '衣服' then 'A:' || 商品分類
    when 商品分類 = '文房具' then 'B:' || 商品分類
    else null
end
```

Q50
和両立とは何か

Q51
和集合(UNION)の例を示そう。
表 18.1 のようなテーブル「商品」があるとする。
(ア)とすると、表 18.1 と全く同じものが得られる。
これは、デフォルトで(イ)からである。
(イ)のを防ぐには(ウ)とすればよい。
(ウ)を実行すると、(エ)を得る。

Q52
積集合は(ア)で行う。
列「部員名」を持つ 2 つのテーブル「野球部員表」、「サッカー部員表」があるとする。
このどちらにも属している部員をみつけるための sql 文は(イ)である。

Q53
差集合を計算するコマンド

A50
列の数が等しい、対応する各列のデータ型が等しい
(即ち列の名前は異なっていてよい)

A51
(ア): select * from 商品 union select * from 商品;
(イ): 重複行が除去される
(ウ): select * from 商品 union all select * from 商品;
(エ)表 18.2

表 18.2

	商品id character (4)	商品名 character varying (100)	商品分類 character varying (32)	販売単価 integer	仕入単価 integer	登録日 date
1	0001	Tシャツ	衣服	1000	500	2009-09-20
2	0002	Yシャツ	衣服	2000	1500	2009-09-21
3	0003	鉛筆削り	文房具	150	100	2009-09-21
4	0004	鉛筆1ダース	文房具	100	60	2009-09-21
5	0005	消しゴム	文房具	50	20	2009-09-22
6	0001	Tシャツ	衣服	1000	500	2009-09-20
7	0002	Yシャツ	衣服	2000	1500	2009-09-21
8	0003	鉛筆削り	文房具	150	100	2009-09-21
9	0004	鉛筆1ダース	文房具	100	60	2009-09-21
10	0005	消しゴム	文房具	50	20	2009-09-22

A52
(ア):intersect
(イ):
SELECT 部員名 FROM 野球部員表
INTERSECT
SELECT 部員名 FROM サッカー部員表;

A53
except

Q54
結合演算について。
大きく(ア)結合、(イ)結合、(ウ)がある。
(ア)結合や(イ)結合は(エ)する処理だ。
(エ)したときに(オ)のようになって、(カ)くなってしまう。ここで生じる(キ)を「耳」と呼ぶことにしよう。
(ア)結合は、「(ク) (ケ)」で行う。耳は(コ)。
(イ)結合は、3 種類あって、「(サ)(シ)(ケ)」、「(ス)(シ)(ケ)」、「(セ)(シ)(ケ)」
(サ)(シ)(ケ)では右耳、(ス)(シ)(ケ)では左耳を(コ)が、(セ)(シ)(ケ)では両耳を(ソ)。
最後に(ウ)については特殊だが、すべての結合の基礎にある。
というのも(ウ)は(タ)を返すもので、これを(チ)ことができるからだ。
また、(ウ)以外の各結合は(ク)、(シ)がなくても一意に特定できるため、(ク)、(シ)は省略できる。

A54

(ア): 内部

(イ): 外部

(ウ): 直積

(エ): 同じとされた(※)列を基準に 2 つのテーブルを連結

※「結合条件」という。等号以外にも使用可能なので「同じ」というのは厳密には誤り(?)

補足:テーブルの左右のふちにのりを付けて、上下にずらしながらくっつけることをイメージせよ

(オ): 図 54.1

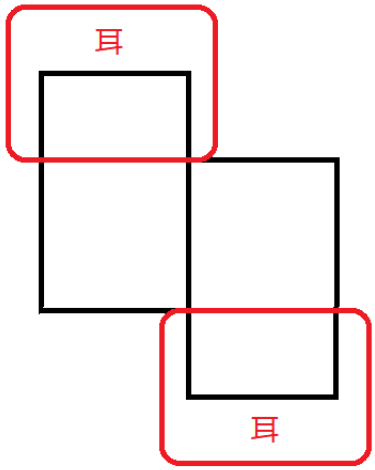


図 54.1 結合

(カ): 長方形でな

(キ): 上下の余計な部分

(ク): inner

(ケ): join

(コ): 切り落とす

(サ): left

(シ): outer

(ス): right

(セ): full

(ソ): 残す

(タ): 全部の組み合わせ

(チ): フィルタリングすることで他の結合の結果を得る

Q55

表 55.1 のような「ポケモン」テーブルを考えよう。

(1)進化すると属性 A がある条件 f を満たすようなポケモンを求める SQL 文をかけ

(2)2 段階で進化するポケモンを求める SQL 文をかけ

表 55.1 ポケモン

番号	名前	分類	タイプ	高さ	重さ	特性	かくれ特性	進化前	進化後
1	フシギダネ	たねポケモン	くさ/どく	0.7	6.9	しんりょく	ようりょくそ	null	フシギソウ
2	フシギソウ	たねポケモン	くさ/どく	1	13	しんりょく	ようりょくそ	フシギダネ	フシギバナ
3	フシギバナ	たねポケモン	くさ/どく	2	100	しんりょく	ようりょくそ	フシギソウ	null
4	ヒトカゲ	とかげポケモン	ほのお	0.6	8.5	もうか	サンパワー	null	リザード
5	リザード	かえんポケモン	ほのお	1.1	19	もうか	サンパワー	ヒトカゲ	リザードン
6	リザードン	かえんポケモン	ほのお/ひこう	1.7	90.5	もうか	サンパワー	リザード	null
7	ゼニガメ	かめのこポケモン	みず	0.5	9	げきりゅう	あめうけざら	null	カメール
8	カメール	かめポケモン	みず	1	22.5	げきりゅう	あめうけざら	ゼニガメ	カメックス
9	カメックス	こうらポケモン	みず	1.6	85.5	げきりゅう	あめうけざら	カメール	null
10	キャタピー	いもむしポケモン	むし	0.3	2.9	りんぷん	にげあし	null	トランセル
11	トランセル	さなぎポケモン	むし	0.7	9.9	だっぴ	null	キャタピー	バタフリー
12	バタフリー	ちょうちょポケモン	むし/ひこう	1.1	32	ふくがん	いろめがね	トランセル	null
13	ビードル	けむしポケモン	むし/どく	0.3	3.2	りんぷん	にげあし	null	コクーン
14	コクーン	さなぎポケモン	むし/どく	0.6	10	だっぴ	null	ビードル	スピアー
15	スピアー	どくばちポケモン	むし/どく	1	29.5	むしのしらせ	スナイパー	コクーン	null
16	ポッポ	ことりポケモン	ノーマル/ひこう	0.3	1.8	するどいめ/ちどりあし	はとむね	null	ピジョン
17	ピジョン	とりポケモン	ノーマル/ひこう	1.1	30	するどいめ/ちどりあし	はとむね	ポッポ	ピジョット
18	ピジョット	とりポケモン	ノーマル/ひこう	1.5	39.5	するどいめ/ちどりあし	はとむね	ピジョン	null
19	コラッタ	ねずみポケモン	ノーマル	0.3	3.5	こんじょう/にげあし	はりきり	null	ラッタ
20	ラッタ	ねずみポケモン	ノーマル	0.7	18.5	こんじょう/にげあし	はりきり	コラッタ	null
21	オニスズメ	ことりポケモン	ノーマル/ひこう	0.3	2	するどいめ	スナイパー	null	オニドリル
22	オニドリル	くちばしポケモン	ノーマル/ひこう	1.2	38	するどいめ	スナイパー	オニスズメ	null
23	アーボ	へびポケモン	どく	2	6.9	いかく/だっぴ	きんちょうかん	null	アーボック
24	アーボック	コブラポケモン	どく	3.5	65	いかく/だっぴ	きんちょうかん	アーボ	null
25	ピカチュウ	ねずみポケモン	でんき	0.4	6	せいでんき	ひらいしん	ピチュー	ライチュウ
26	ライチュウ	ねずみポケモン	でんき	0.8	30	せいでんき	ひらいしん	ピカチュウ	null

Q56

SELECT * FROM A JOIN B ON A.C = B.D;

を、join を使わない sql 文に書き換えよ。

A55

(1)

```
select
前.名前 as before , 前.重さ, 後.名前 as after, 後.重さ
from ポケモン as 前
join ポケモン as 後
on 前.進化後=後.名前
where f(前.A, 後.A);
```

(2)

```
select
前.名前 as before , 前.重さ, 後.名前 as after, 後.重さ, 再.名前 as second_after, 再.重さ
from ポケモン as 前
join ポケモン as 後 on 前.進化後=後.名前
join ポケモン as 再 on 後.進化後=再.名前;
```

解説: 内部結合をとるから、2 段階以外の進化をするものは null の発生により勝手に消える

A56

```
SELECT * FROM A, B WHERE A.C = B.D;
```

Q57

「商品」テーブルは表 18.1 のとおりとする。

group by 同様に(ア)が、(イ)ことはしない場合、partition by が便利である。

どうやら partition by は(ウ)の中で使われることが多いようだ。

select count(*) from 商品;

を実行すると、商品テーブルの行数である 5 が結果として返る。

select *, count(*) from 商品;

を実行すると、(エ)る。

select *, count(*) (ウ)() from 商品;

を実行すると、(オ)る。

ここから分かる通り、(ウ)は、(カ)する機能を持つ。

(ウ)に引数を与えると、(キ)ができる。

select *, count(*) (ク) from 商品;

を実行すると、表 57.1 を得る。

確かに(ア)ことに成功しているから、各行の商品分類によって、count 列の結果が決まっているわけだ。この場合は、「1 行と 2 行」、「3 行と 4 行と 5 行」で仕切られている。このまとまりを(ケ)という。

(ウ)のような関数を(コ)関数という。

(コ)関数が右に来ることで、count(*)は(サ)関数としてふるまうようになったといえる。

(count(*)が(サ)関数になることによって、(カ)できるようになった)

表 57.1

	商品id character (4)	商品名 character varying (100)	商品分類 character varying (32)	販売単価 integer	仕入単価 integer	登録日 date	count bigint
1	0001	Tシャツ	衣服	1000	500	2009-...	2
2	0002	Yシャツ	衣服	2000	1500	2009-...	2
3	0003	鉛筆削り	文房具	150	100	2009-...	3
4	0004	鉛筆1ダース	文房具	100	60	2009-...	3
5	0005	消しゴム	文房具	50	20	2009-...	3

A57
(ア): 任意の列を基準にテーブルを「パーティション」に切り分ける
(イ): 各パーティションを集約して 1 行にまとめる
(ウ): over
(エ): エラーにな
(オ): 商品テーブルの各行の右に 5 を追加したものを得
(カ): それぞれの行で結果を出力
補足:「行をまとめないのに集合関数のように集計する」ことができる
(キ): 範囲指定
(ク): over(partition by 商品分類)
(ケ): ウィンドウ
(コ): 分析
(サ): ウィンドウ

Q58
ウィンドウ関数には、一部の集約関数およびウィンドウ専用関数が使える。
以下、ウィンドウ専用関数を挙げる。
ROW_NUMBER・・・(ア)する。
select *, row_number() over(partition by 商品分類) from 商品;
の結果は(イ)る。
RANK・・・各行をランク付け
(ウ)・・・各行をランク付け（タイの直後の数字を飛ばさない）
FIRST_VALUE・・・最初の行の値
LAST_VALUE・・・最後の行の値

Q59
表 18.1 のような「商品」テーブルがあるとき、
select 商品名, 商品分類, 販売単価,
(ア)
from 商品;
を実行すると、表 59.1 を得る。

表 59.1

	商品名 character varying (100)	商品分類 character varying (32)	販売単価 integer	rank bigint
1	Tシャツ	衣服	1000	1
2	Yシャツ	衣服	2000	2
3	消しゴム	文房具	50	1
4	鉛筆1ダース	文房具	100	2
5	鉛筆削り	文房具	150	3

A58

(ア): 各行に行番号を付与

(イ): 表 58.1 のようにな

(ウ): DENSE_RANK

表 58.1

	商品id character (4)	商品名 character varying (100)	商品分類 character varying (32)	販売単価 integer	仕入単価 integer	登録日 date	row_number bigint
1	0001	Tシャツ	衣服	1000	500	2009-...	1
2	0002	Yシャツ	衣服	2000	1500	2009-...	2
3	0003	鉛筆削り	文房具	150	100	2009-...	1
4	0004	鉛筆1ダース	文房具	100	60	2009-...	2
5	0005	消しゴム	文房具	50	20	2009-...	3

A59

rank() over(partition by 商品分類 order by 販売単価)