Akademia Górniczo – Hutnicza
Im. St. Staszica w Krakowie
Wydział EAliE
Katedra Automatyki

Laboratorium specjalizacyjne

Porównanie systemów relacyjnych baz danych **PostgreSQL** i **Oracle**

Mirosław Jąkała
Maciej Michno

Plan prezentacji

- Reprezentacja danych
- Przetwarzanie danych
- Wyzwalacze
- Perspektywy / Widoki
- Transakcje
- Rozszerzenia
- Narzędzia

Reprezentacja danych
Typy numeryczne

PostgreSQL:

- **smallint, int2** (2 bajty) zakres: -32768 do +32767
- **integer, int, int4** (4 bajty) zakres:-2147483648 do +2147483647
- **bigint, int8** (8 bajtów) zakres: -9223372036854775808 do 9223372036854775807
- **Numeric [(p,s)], Decimal [(p,s)]** (zmienna długość) zakres: bez limitu

  p – precision, musi być dodatnia

  s – scale, musi być większe bądź równe zero

  Numeric może przyjąć wartość *NaN* – not a number (niezdefiniowane wyniki operacji)

Reprezentacja danych
Typy numeryczne

- **Real, float4** (4 bajty) zakres: 1E-37 do 1E+37, 6 miejsc precyzji. Real może przyjmować *NaN, Infinity, -Infinity*

```
UPDATE table SET x = 'Infinity'
```

- **Double precision, float8** (8 bajtów) 1E-307 to 1E+308 15 miejsc precyzji
- **Serial, serial4** (4 bajty) 1 do 2147483647
- **Bigserial,serial8** (8 bajtów) 1 do 9223372036854775807

```
CREATE TABLE t (
colname SERIAL
);
```

=

```
CREATE SEQUENCE t_colname_seq;
CREATE TABLE t (
colname integer DEFAULT
nextval('t_colname_seq') NOT NULL
);
```

- **Money** (4 bajty) zakres: -21474836.48 do +21474836.47

Reprezentacja danych
Typy numeryczne

Oracle:

NUMBER – typ danych numerycznych stało- i zmienno-przecinkowych

Zakres:

- Liczby dodatnie: $1 * 10^{-130}$ do $9.99...9 * 10^{125}$ do 38 cyfr precyzji
- Liczby ujemne: $-1 * 10^{-130}$ do $-9.99...9 * 10^{125}$ do 38 cyfr precyzji

Może przechowywać także:

- Zero
- +/- nieskończoność
- Niezdefiniowane wyniki operacji (NAN – not a number)

Reprezentacja danych
Typy numeryczne

Definiowanie kolumny typu numerycznego:

```
column_name NUMBER -- bez podania precyzji i skali
column_name NUMBER (precision, scale) -- precyzja i skala
column_name NUMBER (*, scale) -- specyfikacja skali bez precyzji
```

Przykład zastosowania:

Dane wejściowe	Specyfikacja typu	Zapisane dane
7456123.89	NUMBER	7456123.89
7456123.89	NUMBER(*,1)	7456123.9
7456123.89	NUMBER(9)	7456124
7456123.89	NUMBER(9,2)	7456123.89
7456123.89	NUMBER(9,1)	7456123.9
7456123.89	NUMBER(6)	Przekroczona precyzja
7456123.89	NUMBER(7,-2)	7456100

Reprezentacja danych
Typy numeryczne

- Dane numeryczne przechowywane są w bazie ORACLE w formacie naukowym: 1 bajt – cecha, do 20 bajtów mantysy.
- Ilość zajmowanych bajtów w bazie danych można obliczyć według zależności:

```
ROUND((length(p)+s)/2))+1
```

gdzie: p – precyzja, s = 0 gdy dodatnia,

s = 1 gdy liczba ujemna.

- *Zero* oraz *–nieskończoność* przechowywane są na jednym bajcie natomiast *+nieskończoność* na 2 bajtach w unikalnej reprezentacji.

Reprezentacja danych
Typy numeryczne

Typ zmiennoprzecinkowy w odróżnieniu od typu **NUMBER** wykorzystuje precyzję binarna, a nie dziesiętną, co skutkuje zwiększeniem szybkości obliczeń arytmetycznych oraz zredukowaniem wymaganej przestrzeni do przechowywania danych.

- **BINARY_FLOAT** – 32 bity, pojedyncza precyzja, 5 bajtów (liczba całkowita 8 bitów, znak 1 bit, eksponenta 8 bitów, część ułamkowa 23 bity)
- **BINARY_DOUBLE** – 64 bity, podwójna precyzja, 9 bajtów (liczba całkowita 8 bitów, znak 1 bit, eksponenta 11 bitów, część ułamkowa 52 bity)

```
1234.56789 -> 1.235E+003
```

Reprezentacja danych
Typy znakowe

PostgreSQL:

Rozmiar: (4 bajty + długość ciągu + [długość dopełnienia])

- **character varying(n), varchar(n)** - zmienna długość z limitem znaków n
- **character(n), char(n)** - stała długość n z dopełnieniem spacjami
- **text** - zmienna nielimitowana długość (nie jest w standardzie SQL)

```
Character varying = text
Character = character(1)
```

Reprezentacja danych
Typy znakowe

```
CREATE TABLE test2 (b varchar(5));
INSERT INTO test2 VALUES ('ok');
INSERT INTO test2 VALUES ('good ');
INSERT INTO test2 VALUES ('too long');
ERROR: value too long for type character varying(5)
INSERT INTO test2 VALUES ('too long'::varchar(5));
SELECT b, char_length(b) FROM test2;

b      | char_length
-------+-------------
ok     | 2
good   | 5
too l  | 5
```

Specjalne typy znakowe:

- **"char"** - 1 bajt, typ wewnętrzny, pojedynczy znak
- **name** - 64 bajty, typ wewnętrzny dla nazw obiektów (63 znaki + terminator)

Reprezentacja danych
Typy znakowe

Oracle:

- **CHAR -** przechowuje ciągi znakowe, zakres: do 2000 bajtów. Domyślnie przyjmowana długość ciągu to 1 bajt.
- **VARCHAR2, VARCHAR -** przechowuje ciągi znakowe o zmiennej długości. Zakres: do 4000 bajtów.

  Semantyka znakowa i semantyka bajtowa:

```
VARCHAR2(20 BYTE) wtedy SUBSTRB(<string>, 1, 20) używa
semantyki bajtowej
VARCHAR2(10 CHAR) wtedy SUBSTR(<string>, 1, 10) używa
semantyki znakowej
```

- **NCHAR i NVARCHAR2 -** przechowują dane zapisane w Unicode: UTF8 i AL16UTF16. Jedyną semantyką jest semantyka znakowa.

  NCHAR zakres: do 2000 bajtów

  NVARCHAR2 zakres: do 4000 bajtów.
- **LONG -** przechowuje ciągi znakowe o długości do 2GB

Reprezentacja danych
Typy daty i czasu

PostgreSQL:

- **timestamp [(p)] [withouttime zone]** - data i czas (8 bajtów), zakres: 4713 BC do 5874897 AD
- **timestamp [(p)] withtime zone** - data i czas ze strefą czasową (8 bajtów), zakres: 4713 BC do 5874897AD
- **interval[(p)]** - interwał czasowy (12 bajtów), zakres: -178000000 do 178000000 lat
- **date** - tylko data (4 bajty), zakres: 4713 BC do 5874897 AD
- **time [(p)] [withouttime zone]** - tylko czas (8 bajtów), zakres: 00:00:00 do 24:00:00
- **time [(p)] with timezone** - czas ze strefą czasową (12 bajtów), zakres: 00:00:00+1359 do 24:00:00+1359

  p – od 0 do 6 dla timestamp i interval

  p – od 0 do 10 dla time

Reprezentacja danych
Typy daty i czasu

- **abstime** i **reltime** są wewnętrznymi typami o niskiej precyzji. Nie zaleca się ich stosowania, mogą wkrótce zniknąć.

Specjalne wartości:

String wejściowy	Typy	Opis
epoch	date, timestamp	1970-01-01 00:00:00 zerowy czas systemu
infinity	timestamp	+ nieskończoność
-infinity	timestamp	- nieskończoność
now	date, time	Aktualny czas
today	date, timestamp	Północ
tomorrow	date, timestamp	Północ dnia następnego
yesterday	date, timestamp	Północ dnia poprzedniego
allballs	time	00:00:00.00 UTC

Reprezentacja danych
Typy daty i czasu

Oracle:

- **DATE** - przechowuje datę i czas (rok, miesiąc, dzień, godzina, minuta, sekunda) w polach o stałej długości (7 bajtów każde).

  Zakres: January 1, 4712 BC - December 31, 4712 AD

  Standardowy format daty: DD-MON-YY

  Standardowy 24 godzinny format czasu: HH:MI:SS

Typ daty	Strefa czasowa	Części sekundy
DATE	Nie	Nie
TIMESTAMP	Nie	Tak
TIMESTAMP WITH TIME ZONE	Bezpośrednia	Tak
TIMESTAMP WITH LOCAL TIME ZONE	Względna	Tak

Reprezentacja danych
Typy binarne

PostgreSQL:

- **bytea** – (4 bajty + string binarny) zmiennej długości string binarny.
  Pozwalają przechowywać:
  Ciąg binarny dla wartości zero,
  Ciągi binarne typu „non-printable” spoza zakresu 32 – 126

Wartość dziesiętna	Opis	Reprezentacja wejściowa	Przykład	Reprezentacja wyjściowa
0	Zerowy bajt	'\\000'	SELECT '\\000'::bytea;	\000
39	Apostrof	'\" lub '\\047'	SELECT '\"::bytea;	'
92	backslash	'\\\\' lub '\\134'	SELECT '\\\\'::bytea;	\\
0 do 31 oraz 127 do 255	"non-printable"	'\\xxx' (ósemkowo)	SELECT '\\001'::bytea;	\001

Reprezentacja danych
Typy binarne

Oracle:

BLOB, CLOB, NCLOB oraz BFILE umożliwiają przechowywanie dużych bloków danych (tekst, grafika, pliki video, dźwiękowe itp.) w postaci znakowej lub binarnej.

- BLOB – przechowuje dane binarne
- CLOB – przechowuje dane typu znakowego
- NCLOB – przechowuje dane typu znakowego z użyciem kodowania UNICODE
- BFILE – przechowuje binarne dane w pliku poza bazą danych.

Reprezentacja danych
Typy binarne

Różnice pomiędzy LOB a LONG:

- Tabela może zawierać wiele kolumn typu LOB natomiast tylko jedną typu LONG
- Tabela zawierająca LOB'y może być partycjonowana, natomiast maksymalny rozmiar LOB'a 8 TB, LONG'a 2 GB
- LOB'y (prócz NCLOB) mogą być atrybutem w stworzonym przez użytkownika typie obiektowym
- Tymczasowe LOB'y działające jako lokalne zmienne mogą być wykorzystywane do transformacji na danych typu LOB. Tymczasowe LOBy są tworzone w tymczasowej niezależnej od innych tablic przestrzeni. Dla LONG'ów nie istnieje struktura tymczasowa.
- Tablice zawierające LOB'y mogą być replikowane.

Reprezentacja danych
Typ bitowy i ciągów bitowych

PostgreSQL:

- **boolean** - (1 bajt) trójstanowy:

  PRAWDA: ’t’ ’true’ ’y’ ’yes’ ’1’

  FAŁSZ: ’f’ ’false’ ’n’ ’no’ ’0’

  NIEZNANY: NULL

- Ciągi bitowe (ciągi 1 lub 0)

  bit(n) - dokładnie n bitów

  bit varying(n) – maksylanie n bitów

  bit = bit(1)

  bit varying – nieograniczony ciąg bitowy

Reprezentacja danych
Typy geometryczne

PostgreSQL:

Nazwa	Rozmiar	Reprezentacja	Opis
point	16 bajtów	Punkt na płaszczyźnie	(x,y)
line	32 bajty	Prosta	((x1,y1),(x2,y2))
lseg	32 bajty	odcinek	((x1,y1),(x2,y2))
box	32 bajty	Prostokąt	((x1,y1),(x2,y2))
path	16 + 16 n bajtów	Zamknięta scieżka	((x1,y1),…)
path	16 + 16 n bajtów	Otwarta ścieżka	[(x1,y1),…]
polygon	40 + 16 n bajtów	Poligon	((x1,y1),…)
circle	24 bajty	Okrąg	<(x,y),r>

Reprezentacja danych
Typy adresów sieciowych

PostgreSQL:

Nazwa	Rozmiar	Opis
Cidr	12 lub 24 bajty	Adres sieci IPv4 lub IPv6
Inet	12 lub 24 bajty	Adres hosta i sieci IPv4 lub IPv6
Macaddr	6 bajtów	Adres MAC

Reprezentacja danych
Typ tablicowy

PostgreSQL:

Kolumny tabel mogą być wielowymiarowymi tablicami typów wbudowanych lub zdefiniowanych przez użytkownika

```
CREATE TABLE sal_emp (
name text,
pay_by_quarter integer[4],
schedule text[2][2]
);

INSERT INTO sal_emp
VALUES ('Bill',
'{10000, 10000, 10000, 10000}',
'{{"meeting", "lunch"}, {"training", "presentation"}}');
```

Reprezentacja danych

Oracle:

- **RAW, LONG RAW -** zmiennej długości, służą do przechowywania danych nie interpretowanych podczas przenoszenia między systemami.
- **XMLType -** służy do przechowywanie danych XML. Może być używany do definicji kolumn tabel i widoków, w PL/SQL, SQL, Java oraz przez JDBC, OCI.
- **URI** (uniform resource identifier) (URIType, DBURIType, XDBURIType, HTTPURIType) jest uogólnieniem URL. Wykorzystywany do przechowywania adresów typu URL, wskazującego na dokument lub specyficzną część dokumentu.
- **Any Types** (anytype, anydata, anydataset) pozwalają na modelowanie parametrów procedur oraz kolumn tablic aktualnie nieznanego typu danych.

Reprezentacja danych

Oracle:

Spatial Types pozwalają na przechowywanie informacji geograficznych (GIS). Do obsługi tych typów danych wymagany jest Oracle Spatial.

- **SDO_GEOMETRY** – opis geometryczny obiektów przestrzennych
- **SDO_TOPO_GEOMETRY** – opis geometrii topologii
- **SDO_GEORASTER** – sieć rastrowa lub obraz

Reprezentacja danych - Media Types

Oracle:

- **ORDAudio** – typ obiektowy służący do przechowywania i zarządzania danymi audio.
- **ORDImage** - typ obiektowy służący do przechowywania i zarządzania obrazkami.
- **ORDImageSignature** - typ obiektowy służący do pełnej reprezentacji koloru, tekstury oraz kształtu przechowywanych obrazów.
- **ORDVideo** - typ obiektowy służący do przechowywania i zarządzania danymi video.
- **ORDDoc** - typ obiektowy służący do przechowywania i zarządzania danymi multimedialnymi – przechowywanie wszystkich typów mediów w jednej kolumnie.

Reprezentacja danych - Media Types

Oracle:

- **SI_StillImage** - typ obiektowy służący do reprezentacji obrazów cyfrowych wraz z wysokością, szerokością oraz formatem.
- **SI_Color** – typ obiektowy przechowujący wartości kolorów.
- **SI_AverageColor** - typ obiektowy reprezentujący cechę charakteryzującą obraz za pomocą jego średniej wartości koloru.
- **SI_ColorHistogram** - typ obiektowy reprezentujący cechę charakteryzującą obraz za pomocą jego histogramu.
- **SI_PositionalColor** – obraz przedstawiony jest jako n x m prostokątów. Ten typ obiektowy reprezentuje cechę obrazu za pomocą najbardziej znaczącego koloru n x m prostokątów.
- **SI_Texture** – typ obiektowy reprezentujący cechę charakteryzującą obraz za pomocą rozmiaru powtarzających się elementów, kontrastu oraz kierunkowości.
- **SI_FeatureList** – typ obiektowy zawierający listę do czterech cech obrazu zdefiniowanych przy pomocy następujących typów obiektowych: SI_AverageColor, SI_ColorHistogram, SI_PositionalColor i SI_Texture, powiązanych za pomocą wagi każdej z cech.

Przetwarzanie danych
Łączenie tablic

```
T1 CROSS JOIN T2

T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 ON
boolean_expression

T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 USING
( join column list )

T1 NATURAL { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2
```

- INNER i OUTER są opcjonalne. INNER jest wartością domyślną. LEFT, RIGHT i FULL występują jedynie w OUTER JOIN.
- ON jest najbardziej ogólnym warunkiem przyjmującym wartości binarne na takiej samej zasadzie jak warunek WHERE. Para wierszy zostanie połączona, gdy wyrażenie po ON jest prawdziwe.
- USING zawiera listę nazw kolumn oddzielonych przecinkami, które są wspólne dla złączanych tablic.

Przetwarzanie danych
Łączenie tablic – CROSS JOIN

```
t1
 num | name
-----+------
   1 | a
   2 | b
   3 | c
```

```
t2
 num | value
-----+-------
   1 | xxx
   3 | yyy
   5 | zzz
```

```
SELECT * FROM t1 CROSS JOIN t2;
 num | name | num | value
-----+------+-----+-------
   1 |    a |   1 | xxx
   1 |    a |   3 | yyy
   1 |    a |   5 | zzz
   2 |    b |   1 | xxx
   2 |    b |   3 | yyy
   2 |    b |   5 | zzz
   3 |    c |   1 | xxx
   3 |    c |   3 | yyy
   3 |    c |   5 | zzz
(9 rows)
```

Przetwarzanie danych
Łączenie tablic – INNER JOIN

t1

num	name
1	a
2	b
3	c

t2

num	value
1	xxx
3	yyy
5	zzz

```
SELECT * FROM t1 INNER JOIN t2 ON t1.num = t2.num;
 num | name | num | value
-----+------+-----+-------
   1 |    a |   1 | xxx
   3 |    c |   3 | yyy

SELECT * FROM t1 INNER JOIN t2 USING (num);
 num | name | value
-----+------+-------
   1 |    a | xxx
   3 |    c | yyy

SELECT * FROM t1 NATURAL INNER JOIN t2;
 num | name | value
-----+------+-------
   1 |    a | xxx
   3 |    c | yyy
```

Przetwarzanie danych
Łączenie tablic – LEFT OUTER JOIN

```
t1
 num | name
-----+------
   1 | a
   2 | b
   3 | c
```

```
t2
 num | value
-----+-------
   1 | xxx
   3 | yyy
   5 | zzz
```

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.num = t2.num;
-- składnia Oracle
SELECT * FROM t1, t2 WHERE t1.num = t2.num(+);
num | name | num | value
-----+------+-----+-------
   1 |    a |   1 | xxx
   2 |    b |     |
   3 |    c |   3 | yyy
(3 rows)

SELECT * FROM t1 LEFT JOIN t2 USING (num);
 num | name | value
-----+------+-------
   1 |    a | xxx
   2 |    b |
   3 |    c | yyy
(3 rows)
```

Przetwarzanie danych
Łączenie tablic – RIGHT OUTER JOIN

```
t1
 num | name
-----+------
   1 | a
   2 | b
   3 | c
```

```
t2
 num | value
-----+-------
   1 | xxx
   3 | yyy
   5 | zzz
```

```
SELECT * FROM t1 RIGHT JOIN t2 ON t1.num = t2.num;

-- składnia Oracle
SELECT * FROM t1, t2 WHERE t1.num(+) = t2.num;

 num | name | num | value
-----+------+-----+-------
   1 |    a |   1 | xxx
   3 |    c |   3 | yyy
     |      |   5 | zzz
(3 rows)
```

Przetwarzanie danych
Łączenie tablic – FULL OUTER JOIN

```
t1
 num | name
-----+------
   1 | a
   2 | b
   3 | c
```

```
t2
 num | value
-----+-------
   1 | xxx
   3 | yyy
   5 | zzz
```

```
SELECT * FROM t1 FULL JOIN t2 ON t1.num = t2.num;

 num | name | num | value
-----+------+-----+-------
   1 |    a |   1 | xxx
   2 |    b |     |
   3 |    c |   3 | yyy
     |      |   5 | zzz
(4 rows)
```

Przetwarzanie danych
Łączenie zapytań

PostgreSQL:

```
query1 UNION [ALL] query2
query1 INTERSECT [ALL] query2
query1 EXCEPT [ALL] query2
```

- **UNION** jest złączeniem wyników zapytań pierwszego i drugiego, eliminujące duplikaty chyba że użyto parametru **ALL**.
- **INTERSECT** zwraca wszystkie wiersze, które występują zarówno w wynikach pierwszego i drugiego zapytań. Eliminowane są duplikaty, chyba że użyto parametru **ALL**.
- **EXCEPT** zwraca wszystkie wiersze wyniku zapytania pierwszego, które nie występują w wyniku zapytania drugiego. Eliminowane są również duplikaty, chyba że użyto parametru **ALL**.

Przetwarzanie danych
Łączenie zapytań

Oracle:

```
query1 UNION [ALL] query2
query1 INTERSECT query2
query1 MINUS query2
```

- **UNION** jest złączeniem wyników zapytań pierwszego i drugiego, eliminujące duplikaty chyba że użyto parametru **ALL**.
- **INTERSECT** zwraca wszystkie wiersze, które występują zarówno w wynikach pierwszego i drugiego zapytań.
- **MINUS** zwraca wszystkie wiersze wyniku zapytania pierwszego, które nie występują w wyniku zapytania drugiego. Eliminowane są również duplikaty.

Aby można było skorzystać z łączenia zapytań, muszą one zwracać taką samą ilość kolumn oraz odpowiadające kolumny muszą mieć odpowiedni typ danych.

Przetwarzanie danych
LIMIT i OFFSET

PostgreSQL:

```
SELECT select_list
    FROM table_expression
    [LIMIT { number | ALL }] [OFFSET number]
```

- **LIMIT** służy do ograniczenia ilości wierszy wyniku zapytania. Wynik będzie zawierał *number* lub mniej wierszy. W przypadku użycia parametru **ALL** zwracane są wszystkie wiersze zapytania.
- **OFFSET** służy do pominięcia pierwszych *number* wierszy zapytania.

W przypadku użycia **LIMIT/OFFSET** konieczne jest użycie klauzuli **ORDER BY** w celu jednoznacznego posortowania wierszy wyniku.

Przetwarzanie danych Zapytania hierarchicznie

Oracle:

Gdy tablica zawiera dane hierarchiczne możliwe jest z wybór wierszy w sposób hierarchiczny wykorzystując przeznaczone do tego celu zapytania hierarchiczne.

Parametry zapytania:

- **START WITH** *warunek* – określa wiersz będący korzeniem drzewa hierarchii
- **CONNECT BY** *warunek* – określa relację pomiędzy wierszami nadrzędnymi i potomnymi

W zapytaniach hierarchicznych jeden z członów warunku musi być poprzedzony operatorem PRIOR odnoszącym się do wiersza nadrzędnego

Przetwarzanie danych
Zapytania hierarchicznie

Oracle:

```
... PRIOR expr = expr
lub
... expr = PRIOR expr
```

W przypadku gdy warunek jest złożony tylko jeden jego człon musi być poprzedzony operatorem **PRIOR** natomiast może ich być więcej:

```
CONNECT BY last_name != 'King' AND PRIOR employee_id = manager_id
...
CONNECT BY PRIOR employee_id = manager_id and PRIOR account_mgr_id
= customer_id ...
```

Przetwarzanie danych Zapytania hierarchicznie

Oracle:

```
SELECT employee_id, last_name, manager_id
FROM employees
CONNECT BY PRIOR employee_id = manager_id;

EMPLOYEE_ID LAST_NAME  MANAGER_ID
----------- ---------- ----------
        101 Kochhar           100
        108 Greenberg         101
        109 Faviet            108
        110 Chen              108
        111 Sciarra           108
        112 Urman             108
        113 Popp              108
        200 Whalen            101
```

Przetwarzanie danych
Tablica DUAL

Oracle:

Dual jest tablicą automatycznie tworzoną przez bazę Oracle. Znajduje się w schemacie użytkownika **SYS**, ale jest dostępna dla wszystkim użytkowników pod nazwą **DUAL**. Zbudowana jest z jednej kolumny **DUMMY** zdefiniowanej jako **VARCHAR(1)** i zawiera jeden wiersz o wartości *X*.
Tablica **DUAL** jest bardzo przydatna w przypadku obliczeń stałych wartości. Zawsze zwraca jeden wiersz.

```
SELECT TO_CHAR(TO_DATE('27-OCT-98', 'DD-MON-RR') ,'YYYY') "Year"
FROM DUAL;
Year
----
1998
SELECT CHR(67)||CHR(65)||CHR(84) "Dog" FROM DUAL;
Dog
---
CAT
```

Przetwarzanie danych
Schematy

PostgreSQL:

Klaster bazy PostgreSQL może zawierać jedną lub więcej baz.

Użytkownicy lub grupy użytkowników są dzielone pomiędzy cały klaster, natomiast nie są dzielone dane z poszczególnych baz danych.

Baza danych może zawierać jeden lub więcej nazwanych schematów, w których znajdują się tablice. Schematy mogą zawierać inne obiekty np. typy danych, funkcje, operatory.

Tak samo nazwane obiekty mogą być używane w różnych schematach bez żadnych ograniczeń.

Zalety stosowania schematów:

- Pozwalają wielu użytkownikom na korzystanie z jednej bazy danych bez żadnych konfliktów
- Pozwalają organizować obiekty bazodanowe w łatwiejsze w zarządzaniu grupy

Przetwarzanie danych
Schematy

PostgreSQL:

Tworzenie schematu

```
CREATE SCHEMA schemaname AUTHORIZATION username;
```

Dostęp do obiektów:

```
database.schema.table
```

Tworzenie tablic w schemacie

```
CREATE TABLE myschema.mytable (
...
);
```

Domyślny schemat publiczny:

```
CREATE TABLE products ( ... );
lub
CREATE TABLE public.products ( ... );
```

Przetwarzanie danych
Schematy

Oracle:

Oracle rozpoznaje obiekty, które są powiązane z poszczególnymi schematami lub obiekty, które nie należą wyłącznie do danego schematu.

Schemat jest własnością użytkownika o tej samej nazwie. Do każdego użytkownika należy tylko jeden schemat.

Dostęp do obiektów baz:

```
schema.object.part@dblink
```

Przetwarzanie danych
Schematy

Oracle:

```
CREATE SCHEMA AUTHORIZATION oe
  CREATE TABLE new_product
   (color VARCHAR2(10) PRIMARY KEY, quantity NUMBER)
  CREATE VIEW new_product_view
   AS SELECT color, quantity FROM new_product WHERE color = 'RED'
  GRANT select ON new_product_view TO hr;
```

Polecenie **CREATE SCHEMA** właściwie nie tworzy schematu, gdyż jest on domyślnie tworzony podczas tworzenia użytkownika (**CREATE USER**).
Za pomocą tego polecenia możemy stworzyć wiele tablic oraz widoków oraz nadać uprawnienia dla tych obiektów innym użytkownikom.

Wyzwalacze

Wyzwalacze bazodanowe to zazwyczaj krótkie programy,
Które reagują na operacje na tabelach tej bazy.

PostgreSQL:

```
CREATE TRIGGER name { BEFORE | AFTER } { event [ OR ... ] }
ON table [ FOR [ EACH ] { ROW | STATEMENT } ]
EXECUTE PROCEDURE funcname ( arguments )
```

CREATE TRIGGER tworzy nowy trigger dla danej tablicy, który wywołuje procedurę określoną przez nazwę funcname w przypadku wystąpienia zdarzenia event.

Procedury wywoływane przez trigger'y można tworzyć w następujących językach: PL/pgSQL, PL/Tcl, PL/Perl, PL/Pyton oraz C.

Wyzwalacze

PostgreSQL:

Parametry:

- **name** – nazwa nowego trigger'a unikalna dla danej tablicy. W przypadku wielu trigger'ów zdefiniowanych dla tego samego zdarzenia, są one wywoływane według kolejności alfabetycznej.
- **BEFORE | AFTER** – określają czy funkcja będzie wywoływana przed zdarzeniem go wywołującym czy po.
- **BEFORE** oznacza, że wywołanie następuje przed sprawdzeniem ograniczeń oraz wykonaniem INSERT, UPDATE lub DELETE. Możliwe jest pominięcie lub zmiana aktualnie wstawianego wiersza dla operacji INSERT lub UPDATE.
- **AFTER** oznacza wywołanie funkcji po sprawdzeniu ograniczeń oraz zakończeniu INSERT, UPDATE lub DELETE. Zmiany wywołane przez ostatnią instrukcję INSERT, UPDATE, DELETE są dostępne dla funkcji trigger'a.

Wyzwalacze

PostgreSQL:

Parametry:

- **event** – jedno z INSERT, UPDATE, DELETE lub wiele oddzielonych operatorem OR.
- **table** – tablica dla której tworzony jest trigger.
- **FOR EACH ROW** – wykonywany dla każdego wiersza objętego danym zdarzeniem.
- **FOR EACH STATEMENT** – wykonywany tyko raz dla danego zdarzenia (domyślnie). Wywoływany również w przypadku, gdy nastąpiła modyfikacja zerowej ilości wierszy.
- **funcname** – nazwa procedury, która jest wywoływana przez trigger nie przyjmująca domyślnie żadnych parametrów oraz zwracająca typ trigger.

Opcjonalna lista argumentów oddzielona przecinkami jest udostępniania do funkcji gdy trigger jest wywoływany. Parametry są konwertowane na stałe znakowe.

Wyzwalacze

PostgreSQL:

Zgodność ze standardem SQL:

- PostgreSQL nie pozwala na wywoływanie trigger'ów w związku z aktualizacją konkretnej kolumny

```
AFTER UPDATE OF col1, col2
```

- SQL powala na definiowanie aliasów dla starego i nowego wiersza tabeli używanych w definicji akcji triggera.

```
CREATE TRIGGER ... ON tablename REFERENCING OLD ROW AS
somename NEW ROW AS othername ...).
```

Ze względu na fakt, iż procedury triggerów w PostgreSQL mogą być pisane w różnych językach, dostęp do danych jest specyficzny dla danego języka.

Wyzwalacze

PostgreSQL:

Zgodność ze standardem SQL:

- PostgreSQL pozwala jedynie na wykonanie zdefiniowanej w akcji trigger'a funkcji. Standard natomiast zezwala na wykonanie licznych instrukcji SQL np. CREATE TABLE.

- Standard SQL określa kolejność wykonywania wielu trigger'ów w kolejności ich utworzenia, natomiast PostgreSQL przyjmuje kolejność alfabetyczną.

- PostgreSQL umożliwia wykonywanie wielu akcji w jednym trigger'ze przy użyciu operatora OR, co jest rozszerzeniem standardu SQL.

Wyzwalacze

PostgreSQL:

```
ALTER TRIGGER name ON table RENAME TO newname
```

RENAME - służy do zmiany nazwy istniejącego trigger'a. Aby możliwa była ta operacja użytkownik musi być właścicielem tablicy, do której przypisany jest trigger'a. **ALTER TRIGGER** jest rozszerzeniem standardu SQL.

```
DROP TRIGGER name ON table [ CASCADE | RESTRICT ]
```

CASCADE – automatycznie usuwa obiekty zależne od trigger'a
RESTRICT – odmawia usunięcia trigger'a w przypadku istnienia obiektu zależnego od trigger'a (domyślnie).

Wyzwalacze

Oracle:

```
CREATE [OR REPLACE] TRIGGER [schema.]trigger
        {BEFORE | AFTER | INSTEAD OF}
   {DELETE | INSERT | UPDATE [OF column [, column] ...]}
[OR {DELETE | INSERT | UPDATE [OF column [, column] ...]}] ...
        ON [schema.]{table | view}
   [ [REFERENCING { OLD [AS] old [NEW [AS] new][PARENT [AS] parent]
                  | NEW [AS] new [OLD [AS] old][PARENT [AS] parent]
                  | PARENT [AS] parent [OLD [AS] old][NEW [AS] new]} ]
FOR EACH ROW
   [WHEN (condition)] ]
  pl/sql_block
  call_procedure_statement
```

Wewnątrz wyzwalacza typu ROW mamy dostęp do starej i nowej wartości wiersza. Tzw. *correlation names* istnieją dla każdej modyfikowanej kolumny. Kwantyfikatory pozwalające się odnosić do tych wartości to NEW i OLD.

Wyzwalacze

Oracle:

- **BEFORE** - Nie można używać BEFORE trigger na widoku
  Można zapisywać do :NEW, ale nie można do :OLD
- **AFTER** - Nie można używać AFTER trigger na widoku
  Nie można zapisywać ani do :OLD ani do :NEW
- **INSTEAD OF** - opcja ta umożliwia transparentną metodę modyfikacji danych w widoku, które nie mogą być modyfikowane bezpośrednio. Te wyzwalacze uruchamiane są zamiast zwykłych operacji INSERT lub UPDATE, które nie są możliwe do wykonania na widoku. Domyślnie wyzwalacze INSTEAD OF są uruchamiane dla każdego wiersza FOR EACH ROW.
  Możliwy jest odczyt zarówno :OLD oraz :NEW, ale niemożliwy jest ich zapis.
- **REFERENCING** - opcja ta została wprowadzona, aby uniknąć konfliktów pomiędzy ciałem wyzwalacza, a kolumnami, które mogą nazywać się OLD lub NEW.

Wyzwalacze

Oracle:

```
CREATE VIEW order_info AS
  SELECT c.customer_id,
         c.cust_last_name,
         c.cust_first_name,
         o.order_id,
         o.order_date,
         o.order_status
  FROM customers c,
       orders o
  WHERE c.customer_id =
        o.customer_id;
```

```
CREATE OR REPLACE TRIGGER order_info_insert
INSTEAD OF INSERT ON order_info
DECLARE
  duplicate_info EXCEPTION;
PRAGMA EXCEPTION_INIT (duplicate_info, -00001);
BEGIN
INSERT INTO customers
  (customer_id, cust_last_name, cust_first_name)
VALUES (
  :new.customer_id,
  :new.cust_last_name,
  :new.cust_first_name);
INSERT INTO orders (order_id, order_date,
customer_id)
VALUES (
  :new.order_id,
  :new.order_date,
  :new.customer_id);
EXCEPTION
WHEN duplicate_info THEN
  RAISE_APPLICATION_ERROR (
  num=> -20107,
  msg=> 'Duplicate customer or order ID');
END order_info_insert;
```

Wyzwalacze

Oracle:

```
ALTER TRIGGER [schema.]trigger
        {DISABLE | ENABLE}
        |{RENAME to name }
        |{COMPILE [DEBUG] [COMPILER_PARAMETERS] [REUSE SETTINGS]};
```

- **COMPILE** - pozwala na jawną kompilację ciała trigger'a po wcześniejszej kompilacji obiektów, od których zależy. Kompilacja ta pozwala na wcześniejsze wykrycie błędów niż podczas kompilacji uruchomieniowej.
- **REUSE SETTINGS** - pozwala zachować wyspecyfikowane parametry kompilatora sprzed uruchomienia kompilacji dla danego trigger'a z jego własnymi parametrami.

```
DROP TRIGGER [schema.]trigger;
```

Perspektywy / Widoki

PostgreSQL:

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name [ (
column_name [, ...] ) ] AS query
```

- **CREATE VIEW** definiuje perspektywę z zapytania. Zapytanie na podstawie, którego tworzona jest perspektywa jest wykonywane każdorazowo, gdy następuje odwołanie do niej w jakimkolwiek zapytaniu.
- **TEMPORARY lub TEMP -** tworzy perspektywę tymczasową, która jest usuwana wraz z zakończeniem aktualnej sesji.
  W przypadku, gdy tworzona jest perspektywa do tablic tymczasowych, domyślnie jest ona tymczasowa.
- **column_name –** opcjonalna lista nazw kolumn perspektywy.
- **query –** zapytanie, które udostępnia wybrane kolumny i wiersze perspektywie.

Perspektywy / Widoki

PostgreSQL:

Zgodność ze standardem SQL

Standard SQL specyfikuje dodatkowe opcje dla CREATE VIEW

```
CREATE VIEW name [ ( column_name [, ...] ) ]
AS query
[ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

- **CHECK OPTION** używany przy perspektywach aktualizowalnych. Powoduje, że w przypadku użycia komend INSERT lub UPDATE wprowadzane dane są sprawdzane z warunkami tworzącymi perspektywę.
- **LOCAL** sprawdza integralność danej perspektywy.
- **CASCADED** sprawdza integralność perspektywy oraz perspektyw zależnych (domyślnie).

Perspektywy / Widoki

PostgreSQL:

```
DROP VIEW name [, ...] [ CASCADE | RESTRICT ]
```

- **CASCADE** automatycznie usuwa obiekty zależne od perspektywy (inne perspektywy)
- **RESTRICT** zapobiega usunięciu perspektywy w przypadku, gdy istnieją inne obiekty zależne (domyślnie).

Standard SQL zezwala jedynie na usunięcie tylko jednej perspektywy przy pomocy pojedynczej komendy.

Perspektywy / Widoki

Oracle:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW
     [schema.]view
       [(alias,...) inline_constraint(s)]
       [out_of_line_constraint(s)]
       [XMLType_view_clause]
          AS subquery options
options:
   WITH READ ONLY
   WITH CHECK OPTION [CONSTRAINT constraint]

XMLType_view_clause:
   OF XMLTYPE [XMLSCHEMA XMLSchema_URL] ELEMENT element
     WITH OBJECT IDENTIFIER {DEFAULT | (expr,...)}
   OF XMLTYPE [XMLSCHEMA XMLSchema_URL] ELEMENT
XMLSchema_URL#element
     WITH OBJECT IDENTIFIER {DEFAULT | (expr,...)}
```

Perspektywy / Widoki

Oracle:

- **REPLACE** – trigger'y INSTEAD OF zdefiniowane dla widoku są usuwane gdy widok jest tworzony ponownie.
  W przypadku, gdy zmaterializowane widoki są zależne od widoku, wtedy zostają oznaczone jako UNUSABLE
  i wymagają pełnego odświeżenia danych. UNUSABLE nie mogą być używane ani odświeżane, aż nie zostaną ponownie skompilowane.
- **FORCE** – ustawienie, które specyfikuje widok tworzony bez względu czy tablice bazowe istnieją oraz czy właściciel schematu zawierającego widok posiada do nich uprawnienia.
  W przypadku gdy definicja widoku zawiera ograniczenia odnoszące się do nieistniejących kolumn tabeli, widok nie zostanie stworzony.
- **NO FORCE** – (domyślne) ustawienie, które specyfikuje widok tworzony wyłącznie na istniejących tablicach bazowych oraz gdy właściciel schematu zawierającego widok posiada do nich uprawnienia.

Perspektywy / Widoki

Oracle:

- XMLType_view_clause - klauzuli tej używamy w przypadku tworzenia widoków XMLType, które wyświetlają dane z tablic bazujących na typie XMLType.

subquery options:

- **WITH READ ONLY** - specyfikuje, że widok lub tablica jest tylko do odczytu.
- **WITH CHECK OPTION** - specyfikuje, że baza zabrania jakichkolwiek zmian w tablicach lub widokach, które mogą generować nowe wiersze a nie są zawarte w podzapytaniu.
- **CONSTRAINT** constraint - specyfikuje nazwę ograniczenia CHECK OPTION.

```
DROP VIEW [schema.]view [CASCADE CONSTRAINTS]
```

Widoki zmaterializowane

Oracle:

Widok zmaterializowany jest obiektem bazy danych zawierającym rezultat zapytania.

Aby stworzyć widok zmaterializowany na własnym schemacie należy:

- posiadać uprawnienia tworzenia zmaterializowanego widoku oraz tablic.
- posiadać dostęp do macierzystych tablic, których nie jesteśmy właścicielami, a które są wykorzystywane do tworzenia tego widoku.

```
CREATE MATERIALIZED VIEW [schema.]materialized_view
  [BUILD {IMMEDIATE | DEFFERED }]
  [REFRESH { FAST | COMPLETE | FORCE }]
  [ON COMMIT | ON DEMAND]
  [WITH {PRIMARY KEY | ROWID}]
  [START WITH | NEXT]
  AS SELECT …;
```

Widoki zmaterializowane

Oracle:

- **BUILD** – określa kiedy ma zostać wypełniony widok zmaterializowany:

  IMMEDIATE - parametr domyślny, określający natychmiastowe wypełnienie widoku

  DEFERRED - określa wypełnienie widoku podczas pierwszej operacji odświeżania
- **REFRESH:**

  FAST – odświeżanie przyrostowe, uaktualnienie o zmiany w tablicy macierzystej, dodatkowo musi istnieć **MATERIALIZED VIEW LOG** dla tablicy macierzystej

  COMPLETE – pełne odświeżanie, wykonując zapytanie zmaterializowanego widoku

  FORCE – domyślny parametr, baza wykonuje odświeżanie **FAST** jeżeli jest możliwe, a jeśli nie to **COMPLETE**.

Widoki zmaterializowane

Oracle:

- **ON COMMIT** – określa wykonanie odświeżania **FAST** w momencie zatwierdzenia transakcji na tablicy macierzystej widoku.
- **ON DEMAND** – domyślny parametr, określa wykonanie odświeżania na żądanie poprzez wywołanie jednej z trzech procedur odświeżających DBMS_MVIEW

  Jeśli użyta jest opcja **ON COMMIT** lub **ON DEMAND** wtedy nie można użyć ani **START WITH** ani **NEXT**.
- **START WITH** – określa datę pierwszego odświeżania
- **NEXT** – określa interwał pomiędzy kolejnymi automatycznymi odświeżaniami

  Parametry **START WITH** i **NEXT** muszą odnosić się do przyszłości. Jeśli zostanie pominięty parametr **START WITH** wtedy baza określi datę pierwszego odświeżania na podstawie parametru NEXT.

  W przypadku gdy określony jest wyłącznie parametr **START WITH** wtedy widok zostanie odświeżony tylko raz.

  W przypadku gdy pominięto oba parametry widok nie jest odświeżany automatycznie.

Widoki zmaterializowane

Oracle:

Sposób identyfikowania rekordów:

- **PRIMARY KEY**

  - tabela master musi posiadać ograniczenie PRIMARY KEY

  - klauzula SELECT musi zawierać wszystkie atrybuty wchodzące w skład klucza podstawowego tabeli macierzystej

- **ROWID**

  - specyfikowane, gdy widok nie posiada wszystkich kolumn klucza podstawowego tablicy macierzystej

  - musi bazować na pojedynczej tablicy oraz nie może zawierać DISTINCT oraz funkcji agregujących, GROUP BY, CONNECT BY, podzapytań, złączeń.

Widoki zmaterializowane

Oracle:

```
CREATE MATERIALIZED VIEW mv_suma_sprzedazy
BUILD DEFFERED
REFRESH FAST
START WITH TRUNC(SYSDATE+1) + 9/24
NEXT sysdate+7
AS
  SELECT sklep_id, produkt_id,
  SUM(l_sztuk), SUM(l_sztuk*cena_jedn),
  COUNT(l_sztuk), COUNT(l_sztuk*cena_jedn), COUNT(*)
  FROM sprzedaz
  GROUP BY sklep_id, produkt_id;
```

ALTER MATERIALIZED VIEW – wykorzystywane do zmiany struktury i parametrów.

```
DROP MATERIALIZED VIEW [schema.]materialized view;
```

Usunięcie tablicy macierzystej, nie usuwa automatycznie widoku zmaterializowanego bazującego na niej. W przypadku próby odświeżania widoku zgłaszany jest błąd.

Transakcje

PostgreSQL i **Oracle** traktuje każdą komendę SQL jako oddzielną transakcję otoczoną komendami BEGIN i COMMIT (w przypadku, gdy zakończy się powodzeniem).

- W bloku transakcji możliwe jest użycie komendy ROLLBACK powodującej cofnięcie wszystkich wykonanych komend w bloku.
- **SAVEPOINT** i **ROLLBACK TO** *savepoint_name*

  Instrukcje te pozwalają na głębszą kontrolę transakcji. W wyniku ich zastosowania możliwe jest wycofanie pewnej części bloku transakcji i zatwierdzenie pozostałej jego części.

```
BEGIN;
UPDATE accounts SET balance = balance-100.00 WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance+100.00 WHERE name = 'Bob';
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance+100.00 WHERE name = 'Wally';
COMMIT;
```

Transakcje - Poziomy izolacji

Standard SQL definiuje 4 poziomy izolacji transakcji w celu zapobiegania trzem niepożądanym zjawiskom jakie mogą zajść podczas równoległych transakcji.

- **dirty read -** odczyt danych z niezatwierdzonej innej transakcji.
- **nonrepeatable read -** transakcja odczytuje ponownie dane, które pobrała wcześniej i okazują się one danymi zmodyfikowanymi przez inną transakcję.
- **phantom read -** transakcja wykonuje ponownie zapytanie zwracające zbiór wierszy spełniających określone warunki i okazują się one innym zbiorem danych niż w zapytaniu wcześniejszym.

Poziomy izolacji	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Możliwy	Możliwy	Możliwy
Read committed	Niemożliwy	Możliwy	Możliwy
Repeatable read	Niemożliwy	Niemożliwy	Możliwy
Serializable	Niemożliwy	Niemożliwy	Niemożliwy

Transakcje - Poziomy izolacji

Read Committed Isolation Level

Zapytanie SELECT transakcji objętej tym poziomem izolacji ma dostęp do danych, które zostały zatwierdzone przed rozpoczęciem zapytania, natomiast nie ma dostępu do danych niezatwierdzonych lub zatwierdzonych przez inną transakcję w trakcie wykonywania bieżącego zapytania. Są dostępne natomiast dane, które zostały zmienione w obrębię danej transakcji mimo, że nie zostały zatwierdzone.

Poziom Read Committed jest odpowiedni dla zapytań, z prostą klauzulą WHERE.
Zastosowanie tego poziomu izolacji skutkuje szybką i efektywną realizacją zapytań do bazy danych.

Transakcje - Poziomy izolacji

Serializable Isolation Level

Poziom Serializable jest najbardziej restrykcyjnym poziomem izolacji transakcji. Poziom ten emuluje szeregowe wykonanie transakcji tzn. tak jakby były wykonywane jedna po drugiej. Aplikacja używająca tego poziomu musi być przygotowana do ponownej próby wykonania transakcji w przypadku, gdy serializacja nie zostanie utrzymana.

Poziom Serializable jest używany wyłącznie wtedy, gdy wymagane jest zapewnienie identycznych danych wielu operacjom podczas jednej transakcji, a także:

- w dużych bazach danych z krótkimi transakcjami, które aktualizują mała ilość wierszy
- w przypadku, gdy prawdopodobieństwo, że dwie transakcje będą modyfikować te same wiersze jest relatywnie małe
- gdy względnie długie transakcje wymagają głównie odczytu danych

Transakcje - Poziomy izolacji

PostgreSQL:

Można wybrać każdy z czterech standardowych poziomów izolacji, jednak system rozróżnia tylko dwa różne poziomy izolacji, **Read Committed** oraz **Serializable**. W przypadku wybrania **Read Uncommitted** otrzymujemy **Read Committed (poziom domyślny)** natomiast w przypadku wybrania **Repeatable Read** otrzymujemy **Serializable**. Jest to zgodne ze standardem SQL, który mówi, że cztery poziomy izolacji wyłącznie definiują, które zjawiska nie mogą zajść natomiast nie definiują, które z nich muszą zajść.

Oracle:

Oracle oferuje dwa poziomy izolacji: **Read Committed** i **Serializable** zgodne ze standardem SQL92 oraz **Read-only**, który nie jest częścią standardu. **Read Committed** jest poziomem domyślnym.

Read-only level

Transakcje **Read-only** operują na danych, które zostały zatwierdzone w czasie jej rozpoczęcia, jednakże nie pozwalają na użycie operacji INSERT, UPDATE oraz DELETE.

Rozszerzenia

PostgreSQL:

Interfejsy programistyczne

- libpq C library – podstawowa biblioteka stanowiąca interfejs programistyczny dla aplikacji C; stanowi podstawę do innych interfejsów programistycznych takich języków jak: C++, Perl, Python, Tcl and ECPG.
- Large Objects – wsparcie dla obsługi dużych obiektów przechowywanych w bazie. Umożliwia strumieniowy dostęp do danych.
- ECPG - Embedded SQL in C – wbudowany pakiet SQL; przeznaczony do współpracy z językiem C. Program wykorzystujący ten interfejs składa się z kodu napisanego w C wraz z poleceniami SQL. Kod programu jest analizowany przez wbudowany preprocesor SQL, a następnie konwertowany do języka C, który może być skompilowany za pomocą kompilatora C.

Rozszerzenia

PostgreSQL:

Języki proceduralne:

- PL/pgSQL – język proceduralny będący rozszerzeniem SQL

```
CREATE FUNCTION populate() RETURNS integer AS $$
DECLARE
-- declarations
BEGIN
PERFORM my_function();
END;
$$ LANGUAGE plpgsql;
```

- PL/Tcl (tool command language) – wykorzystywany do pisania funkcji oraz procedur wyzwalaczy w Tcl.
- PL/Perl
- PL/Python

Rozszerzenia

PostgreSQL:

Server Programming Interface (SPI) – zbiór funkcji interfejsu umożliwiający używanie komend SQL wewnątrz funkcji napisanych w języku C.

Inne interfejsy:

- psqlODBC – najbardziej popularny interfejs dla aplikacji Windows
- pgjdbc – interfejs JDBC
- npgsql – interfejs .Net dla aplikacji Windows
- libpqxx – nowszy interfejs C++
- libpq++ - starszy interfejs C++
- pgperl – interfejs Perl
- pgtclng – nowsza wersja interfejsu Tcl
- pgtcl – oryginalna wersja interfejsu Tcl
- pgin – interfejs Tcl napisany w Tcl
- PyGreSQL – biblioteka interfejsu Python
- Psycopg – interfejs Python

Rozszerzenia

Oracle:

- **PL/SQL** – rozszerzenie SQL o języki proceduralny stanowiący standardowy język obsługi bazy Oracle .

```
PROCEDURE debit_account (p_acct_id INTEGER, p_debit_amount REAL)
IS
v_old_balance REAL;
v_new_balance REAL;
e_overdrawn EXCEPTION;
BEGIN
SELECT bal
  INTO v_old_balance
  FROM accts
  WHERE acct_no = p_acct_id;
v_new_balance := v_old_balance - p_debit_amount;
IF v_new_balance < 0 THEN
  RAISE e_overdrawn;
ELSE
  UPDATE accts SET bal = v_new_balance
  WHERE acct_no = p_acct_id;
END IF;
COMMIT;
EXCEPTION
WHEN e_overdrawn THEN
-- handle the error
END debit_account;
```

Rozszerzenia

Oracle:

- **Java** – wbudowane biblioteki jądra JDK: java.lang, java.io,... Wsparcie dla JDBC i SQLJ po stronie klienta i serwera.
- **Pro*C/C++** - prekompilator pozwalający na użycie kodu SQL w plikach źródłowych C/C++. Prekompilator zamienia użyty kod SQL na odpowiednie wywołania biblioteki uruchomieniowej Oracle, a następnie kod jest kompilowany przez kompilator C/C++
- **Pro*COBOL** - prekompilator pozwalający na użycie kodu SQL w plikach źródłowych COBOL. Prekompilator zamienia użyty kod SQL na odpowiednie wywołania biblioteki uruchomieniowej Oracle, a następnie kod jest kompilowany przez kompilator COBOL.

Rozszerzenia

Oracle:

- **OCI** (Oracle Call Interface) i **OCCI** (Oracle C++ Call Interface) – interfejsy programistyczne umożliwiające tworzenie aplikacji używających wywołań natywnych procedur lub funkcji języków programowania do dostępu do bazy danych Oracle.
- **Oracle Data Provider for .NET** (ODP.NET) – interfejs umożliwiający dostęp do bazy danych aplikacjom .NET. Używa oraz dziedziczy klasy oraz interfejsy dostępne w Microsoft .NET Framework Class Library.
- **Oracle Objects for OLE** (OO4O) – interfejs pozwalający na łatwy dostęp do bazy Oracle językom programowania oraz językom skryptowym, które wspierają technologie Microsoft COM Automation i ActiveX. Wspiera więc języki takie jak: Visual Basic, Visual C++, Visual Basic For Applications (VBA), IIS Active Server Pages (VBScript, JavaScript) itp.

Narzędzia

PostgreSQL:

Produkty Open Source
Klienty graficzne

- pgAdmin III – wieloplatformowe narzędzie do administrowania bazą danych
- PhpPgAdmin – webowe narzędzie do administrowania bazą danych

Języki proceduralne (Server side)

- plPHP – język proceduralny bazujący na PHP
- plJava – język proceduralny bazujący na Java
- plR - język proceduralny bazujący na języku statystycznym R
- pl-j - język proceduralny Java dla PostgreSQL

Projekty

- **PostgreSQL** http://www.postgresq.org
  Oficjalna strona projektu PostgreSQL
- **PgFoundry** http://pgfoundry.org/
  strona PostgreSQL Development Group's, która zajmuje się publikacją i rozwojem oprogramowania dla PostgreSQL.

Narzędzia

PostgreSQL:

Produkty komercyjne
Narzędzia programistyczne i administracyjne

dbDeveloper http://www.dbdeveloper.prominentus.com
Wizualne narzędzie do zarządzania wieloma bazami danych: Oracle, MS SQL Server, Borland Interbase, Firebird, MySQL, PostgreSQL, Sybase, DB2

EMS SQL Manager™ for PostgreSQL http://www.sqlmanager.net
Wizualny pakiet narzędzi do projektowania, programowania oraz administrowania bazą danych. Zawiera między innymi: Visual Database Designer, Visual Query Builder, BLOB editor.

Interactive SQL for PostgreSQL http://www.microolap.com
PgISQL (Interactive SQL for PostgreSQL) jest narzędziem do tworzenie, wykonywania oraz przetwarzania zapytań dla bazy PostgreSQL pozwalające wykonywać wszystkie czynności przy pomocy interfejsu graficznego bez jakiejkolwiek wiedzy na temat SQL.

Narzędzia

PostgreSQL:

Navicat 2005 http://pgsql.navicat.com/
Narzędzie wizualne do administrowania oraz zarządzania bazą danych.

pgEdit http://pgedit.com
pgEdit to edytor SQL dla PostgreSQL, który koloruje składnie, pozwala wykonywać kod, posiada wsparcie dla PHP jak również pozwala na zarządzanie bazą danych.

PG Lightning Admin http://www.amsoftwaredesign.com
To graficzne narzędzie do administrowania bazą PostgreSQL z zaawansowanym edytorem kodu (uzupełnianie kodu, podświetlanie składni SQL, podpowiadanie parametrów funkcji)

PostgresDAC http://www.microolap.com
Komponent do bezpośredniego dostępu do serwera PostgreSQL z poziomu aplikacji budowanych w Borland Delphi oraz C++ Builder.

Narzędzia

Oracle:

Oracle Designer 10g http://www.oracle.com
Dostarcza pełny zestaw narzędzi do modelowania i projektowania baz danych. Umożliwia modelowanie procesów, analizę systemową, tworzenie wstępnego projektu systemu, projektowanie aplikacji oraz generację kodu aplikacji.

TOAD http://www.quest.com
Narzędzie do administrowania bazą danych Oracle. Zawiera interfejs graficzny, zaawansowany edytor dla PL/SQL z możliwościami kompilacji oraz wsparciem dla SQL*Plus.

SQL Navigator http://www.quest.com
Posiada wszystkie potrzebne narzędzie do tworzenia, edycji oraz zarządzania obiektami bazy danych Oracle z pełnym wsparciem dla PL/SQL.

Narzędzia

Oracle:

QDesigner http://www.quest.com
Zbiór narzędzi służących do wielopoziomowego modelowania konceptualnego, logicznego oraz fizycznego danych wspierających nowoczesne technologie Java, XML oraz webowe w bazach danych. Umożliwia obiektowe projektowanie przy pomocy UML, a także generację dokumentacji w HTML i RTF.

Quest Central® for Databases http://www.quest.com
Kompletne narzędzie z interfejsem graficznym zawierające moduły do: administrowania bazą danych, diagnozowania wydajności, zarządzania przestrzenią, tuningu SQL.

Źródła

- www.oracle.com
- www.postgresql.org
- PostgreSQL 8.1.0 Documentation
- Oracle® Database Application Developer's Guide – Fundamentals 10g Release 2 (10.2)
- Oracle® DatabaseSQL Reference 10*g* Release 2 (10.2)
- Oracle® Database Concepts 10*g* Release 2 (10.2)