Tabela zagnieżdżona w SQL

```
SELECT DATA_KUPNA, CENA, ZAWARTOSC_KOSZYKA
FROM ZAKUP;
```

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')

```
INSERT INTO ZAKUP VALUES(KUPNO(TO_DATE('28/04/2017',
'DD/MM/YYYY'), 30, ZAKUPY('dzem', 'miod' )));
```

```
SELECT VALUE(X) FROM TABLE(
SELECT ZAWARTOSC_KOSZYKA FROM ZAKUP
WHERE
DATA_KUPNA=TO_DATE('28/04/2017','DD/MM/YYYY')) X;
```

	VALUE(X)
1	dzem
2	miod

Operator Table

```
INSERT INTO TABLE(SELECT ZAWARTOSC_KOSZYKA FROM ZAKUP
WHERE DATA_KUPNA=TO_DATE('28/04/2017',
'DD/MM/YYYY') ) VALUES ('miod');
```

```
SELECT DATA_KUPNA, CENA, ZAWARTOSC_KOSZYKA
FROM ZAKUP;
```

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','miod','miod')

```
INSERT INTO TABLE(SELECT ZAWARTOSC_KOSZYKA
FROM ZAKUP WHERE DATA_KUPNA=TO_DATE('28/04/2017',
'DD/MM/YYYY') ) VALUES ('ziemniaki');
```

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','miod','miod','ziemniaki')

Tabela zagnieżdżona w SQL

```
UPDATE TABLE(SELECT ZAWARTOSC_KOSZYKA FROM ZAKUP
WHERE DATA_KUPNA=TO_DATE('28/04/2017','DD/MM/YYYY') ) X
SET VALUE(X)='marchewka' WHERE VALUE(X)='miod';
```

```
SELECT DATA_KUPNA, CENA, ZAWARTOSC_KOSZYKA
FROM ZAKUP;
```

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','marchewka','marchewka','ziemniaki')

było

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','miod','miod','ziemniaki')

Tabela zagnieżdżona w SQL

```
DELETE FROM TABLE(SELECT ZAWARTOSC_KOSZYKA
FROM ZAKUP
WHERE
DATA_KUPNA=TO_DATE('28/04/2017','DD/MM/YYYY')) X
WHERE VALUE(X)='ziemniaki';
```

```
SELECT DATA_KUPNA, CENA, ZAWARTOSC_KOSZYKA
FROM ZAKUP;
```

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','marchewka','marchewka')

było

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','marchewka','marchewka','ziemniaki')

Tabela zagnieżdżona w SQL

```
SELECT DATA_KUPNA,VALUE(X)
FROM ZAKUP CROSS JOIN
TABLE(ZAWARTOSC_KOSZYKA) X;
```

	DATA_KUPNA	VALUE(X)
2	17/05/04	maslo
3	17/05/04	mleko
4	17/05/04	kasza
5	17/05/04	ryz
6	17/04/28	dzem
7	17/04/28	marchewka
8	17/04/28	marchewka

Przykład kolekcji - tabela o zmiennym rozmiarze w SQL

Przykład

1. Utwórz typ zagnieżdżonej tabeli przechowującej listę tytułów wypożyczonych filmów
2. Utwórz typ obiektowy reprezentujący klienta wypożyczalni o atrybutach:
   - nazwisko (typu łańcuchowego)
   - Imię (typu łańcuchowego)
   - film (typu wcześniej zdefiniowanej kolekcji).
3. Utwórz tabelę obiektową KLIENCI przechowującą obiekty typu KLIENT
4. Wstaw przynajmniej dwóch klientów z wypożyczonymi filmami
5. Wykorzystując operator TABLE i polecenia INSERT, UPDATE i DELETE, wstaw, zmień i usuń jakieś filmy z tabeli zagnieżdżonej skojarzonej z jednym z klientów
6. Napisz zapytanie, które wypisze tabelę składającą się z trzech kolumn: nazwisko i imienia klienta oraz wypożyczonego filmu - wykorzystaj połączenie tabeli obiektowej z zagnieżdżoną tabelą

Tabela zagnieżdżona w PL/SQL

```
create type tytuly_filmow as table of
varchar2(50);
```

```
create or replace type klient as object (
nazwisko varchar2(50),
imie varchar2(50),
film tytuly_filmow
);
```

```
CREATE TABLE KLIENCI OF KLIENT NESTED
TABLE  FILM STORE AS
WYPOZYCZONE_FILMY;
```

Tabela zagnieżdżona w SQL

```
INSERT INTO KLIENCI VALUES(KLIENT('Kowalski','Jan',
TYTULY_FILMOW('Pan Tadeusz','Ranczo')));
```

```
INSERT INTO KLIENCI VALUES(KLIENT('Iksinski','Piotr',
TYTULY_FILMOW('Ogniem i mieczem','Kozioek Matolek')));
```

	NAZWISKO	IMIE	FILM
1	Kowalski	Jan	II372.TYTULY_FILMOW('Pan Tadeusz','Ranczo')
2	Iksinski	Piotr	II372.TYTULY_FILMOW('Ogniem i mieczem','Kozioek Matolek')

```
INSERT INTO TABLE(SELECT FILM FROM KLIENCI
WHERE NAZWISKO='Kowalski') VALUES ('Mis Uszatek');
```

	NAZWISKO	IMIE	FILM
1	Kowalski	Jan	II372.TYTULY_FILMOW('Pan Tadeusz','Ranczo','Mis Uszatek')
2	Iksinski	Piotr	II372.TYTULY_FILMOW('Ogniem i mieczem','Kozioek Matolek')

Tabela zagnieżdżona w SQL

```
UPDATE TABLE(SELECT FILMY FROM KLIENCI
WHERE NAZWISKO=Iksinski') X
SET VALUE(X)= 'Koziołek Matołek' WHERE
VALUE(X)= 'Kozioek Matolek';
```

```
SELECT Nazwisko, IMIE, Film
FROM KLIENCI;
```

	NAZWISKO	IMIE	FILM
1	Kowalski	Jan	II372.TYTULY_FILMOW('Pan Tadeusz','Ranczo','Mis Uszatek')
2	Iksinski	Piotr	II372.TYTULY_FILMOW('Ogniem i mieczem','Koziołek Matołek')

było

	NAZWISKO	IMIE	FILM
1	Kowalski	Jan	II372.TYTULY_FILMOW('Pan Tadeusz','Ranczo','Mis Uszatek')
2	Iksinski	Piotr	II372.TYTULY_FILMOW('Ogniem i mieczem','Kozioek Matolek')

Tabela zagnieżdżona w SQL

```
DELETE FROM TABLE(SELECT FILM FROM KLIENCI
WHERE NAZWISKO='Kowalski') X
WHERE VALUE(X)='Ranczo';
```

```
SELECT Nazwisko, IMIE, Film
FROM KLIENCI;
```

	NAZWISKO	IMIE	FILM
1	Kowalski	Jan	II372.TYTULY_FILMOW('Pan Tadeusz','Mis Uszatek')
2	Iksinski	Piotr	II372.TYTULY_FILMOW('Ogniem i mieczem','Koziołek Matołek')

było

	NAZWISKO	IMIE	FILM
1	Kowalski	Jan	II372.TYTULY_FILMOW('Pan Tadeusz','Ranczo','Mis Uszatek')
2	Iksinski	Piotr	II372.TYTULY_FILMOW('Ogniem i mieczem','Koziołek Matołek')

Tabela zagnieżdżona w SQL

```
SELECT NAZWISKO, IMIE, VALUE(X)
FROM KLIENCI CROSS JOIN TABLE(FILM) X;
```

	NAZWISKO	IMIE	VALUE(X)
1	Kowalski	Jan	Pan Tadeusz
2	Kowalski	Jan	Mis Uszatek
3	Iksinski	Piotr	Ogniem i mieczem
4	Iksinski	Piotr	Koziołek Matołek

Konstrukcja CAST

CAST(MULTISET(podzapytanie) AS typ_kolekcji)

- Podzapytanie zagnieżdżone w konstrukcji CAST(MULTISET(... to podzapytanie skorelowane
- Wynik podzapytania jest konwertowany na kolekcję, której typ podany jest w wyrażeniu
- Konstrukcja CAST pozwala na zamianę wyniku zapytania w kolekcję
- Konstrukcji „CAST(MULTISET(...„ można używać nie tylko w zapytaniach, ale również w innych miejscach - wszędzie tam, gdzie konieczne jest podanie wyrażenia, które zwraca całą kolekcję, np. w poleceniu INSERT, przy wstawianiu nowego obiektu zawierającego kolekcję do tabeli obiektowej

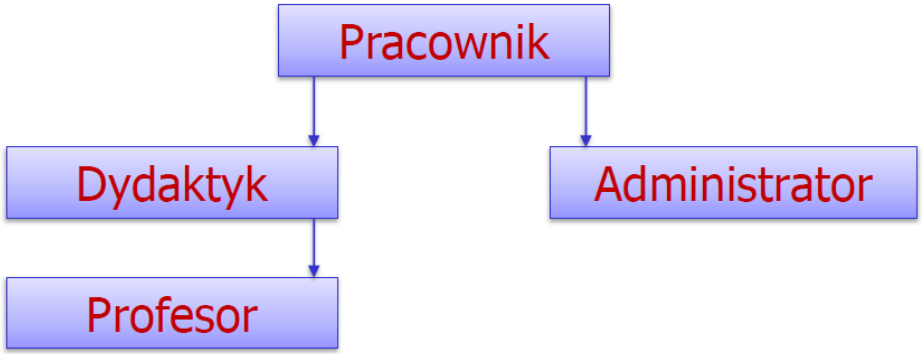
Konstrukcja CAST

```
create or replace type pracownicy as
table of varchar2(50);
```

```
SELECT NAZWA, CAST(MULTISET(
SELECT NAZWISKO||' '||imie
FROM WYKLADOWCA WHERE Id_tytul=id_tyt) AS
pracownicy)
AS KOLEKCJA FROM TYTULNAUKOWY;
```

	NAZWA	KOLEKCJA
1	mgr	II372.PRACOWNICY('Badina2 MILOSZ','Breitscheidel Przybymir','Drul Walenty','Krzępka Paschalis','Skrabas Marian','Montz Bartosz','Bukerzyński Naczesław')
2	mgr inż.	II372.PRACOWNICY('Swiergoń Saul','Biłokryła Dymitr','Igrzycki Juliusz','Klepato Cyryl')
3	dr	II372.PRACOWNICY('Hegier Sozant','Juka Iwon','Hoteit Wszemił','Paszylke Roland')
4	dr inż.	II372.PRACOWNICY('Zatwardzińska Damaris')
5	dr hab.	II372.PRACOWNICY('Dudel Dymitr','Prag Florencjusz','Krog Czcibor')
6	dr hab. inż.	II372.PRACOWNICY('Czerwiaków Kondrad','Caganos Maur')
7	Prof.	II372.PRACOWNICY('Wieleczek Tomasz','Cywentuch Wilhelm','Kiwaczycki Budziwoj','Simiak Jaropełk')
8	Prof. dr hab.	II372.PRACOWNICY('Wółczyńska Ludomił','Skarżuńska Germanik','Bialec Wiktoriusz')
9	Prof. dr hab. inż.	II372.PRACOWNICY('Badina Dolores','Aborowicz Sergiusz','Harwankowski Idzi')

Dziedziczenie



Dziedziczenie polega na definiowaniu nowego typu obiektowego w oparciu o istniejący typ obiektowy

- nowy typ stanowi podtyp (specjalizację) swojego nadtypu (przodka)
- podtyp dziedziczy wszystkie składowe i metody **MEMBER** i **STATIC**
- podtyp może dodawać nowe składowe i przesłaniać metody
- każdy podtyp może mieć tylko jeden nadtyp
- podtyp może dziedziczyć tylko z nadtypu, który został zadeklarowany jako **NOT FINAL** (uwaga: domyślnie każdy typ jest **FINAL**)
- metody porządkujące mogą się pojawić tylko w korzeniu hierarchii

Dziedziczenie

```
create or replace type figura as object (
kolor varchar2(20)
) not final;
```

- Polecenie tworzące typ FIGURA o atrybucie KOLOR
- Aby istniała możliwość tworzenia podtypów typu obiektowego należy to jawnie nakazać za pomocą słów kluczowych NOT FINAL

```
create or replace type kwadrat under figura (
dlugosc_boku numeric(5,2)
);
```

```
create or replace type kolo under figura (
promien numeric(5,2)
);
```

- Zamiast słowa kluczowego AS OBJECT słowem kluczowym UNDER po którym podano nazwę nadtypu
- Nowe podtypy reprezentują konkretne kształty figur, czyli kwadrat i koło

Kolejność podawania atrybutów w konstruktorze atrybutowym typu obiektowego

```
DECLARE
fi figura:=new figura('Czerwony');
ko kolo:=new kolo('Niebieski',6);
kw kwadrat:=new kwadrat('Zielony',10);
BEGIN
    Null;
END;
```

Przesłanianie metod

```
create or replace type figura as object (
kolor varchar2(30),
member function pole return numeric
) not final;
```

- Można modyfikować implementację metod odziedziczonych z nadtypu - przesłanianie
- W typie FIGURA zadeklarowana jest metodę POLE, której zadaniem jest obliczenie pola powierzchni figury
- Ponieważ nie znamy kształtu figury reprezentowanej przez obiekt typu FIGURA, metoda POLE jest zaimplementowano tak, aby zawsze zwracała wartość zero

```
create type body figura as
member function pole return numeric as
begin
return 0;
end;
end;
```

Przesłanianie metod

```
create or replace type kwadrat under figura (
dlugosc_boku numeric(5,2),
overriding member function pole return numeric
);
```

```
create type body kwadrat as
overriding member function pole return numeric as
begin
return dlugosc_boku*dlugosc_boku;
end;
end;
```

- Deklaracja metody POLE różni się tutaj od deklaracji tej metody w typie FIGURA tym, że deklaracja rozpoczyna się od słowa kluczowego OVERRIDING, które mówi, że deklarowana metoda będzie przesłaniać metodę z nadtypu

Przesłanianie metod

```
create or replace type kolo under figura (
promien numeric(5,2),
overriding final member function pole return numeric
);
```

```
create type body kolo as
overriding final member function pole return numeric
as
begin
return 3.14*promien*promien;
end;
end;
```

- Deklaracja metody POLE w typie KOLO ma dodatkowo słowo kluczowe FINAL - ogranicza możliwość przesłaniania implementacji metody w podtypach typu KOLO Jeżeli

Metody abstrakcyjne

- Metody abstrakcyjne - deklaracja stanowi jedynie zapowiedź, że w którymś z podtypów pojawi się ich implementacja
- Polecenie tworzy abstrakcyjny typ FIGURA, z abstrakcyjną metodą POLE
- Słowo kluczowe NOT INSTANTIABLE przed MEMBER oznacza, że metoda ta nie będzie implementowana
- Typ FIGURA zawiera abstrakcyjną metodę - musi być zdefiniowany jako typ abstrakcyjny - umieszczenie słów kluczowych NOT INSTANTIABLE na końcu polecenia tworzącego typ

```
create or replace type figura as object (
kolor varchar2(30),
not instantiable member function pole return numeric
) not instantiable not final;
```

Metody abstrakcyjne

```
DECLARE
fi figura:=new figura('Czerwony');
kw kwadrat:=new kwadrat('Zielony',10);
ko kolo:=new kolo('Niebieski',6);

BEGIN

fi:=kw;
fi :=ko;

END;
```

deklaracja zmiennych typów FIGURA, KWADRAT i KOLO, oraz utworzenie obiektów typów KWADRAT i KOLO

do zmiennej typu FIGURA można przypisać obiekty typu KWADRAT albo KOLO – dopuszczalne przypisania

Metody abstrakcyjne

```
DECLARE
fi figura:=new figura('Czerwony');
kw kwadrat:=new kwadrat('Zielony',10);
ko kolo:=new kolo('Niebieski',6);

BEGIN

kw:=fi;
ko:=fi;

END;
```

Error report -
ORA-06550: linia 7, kolumna 6:
PLS-00382: wyrażenie jest niewłaściwego typu
ORA-06550: linia 7, kolumna 1:
PL/SQL: Statement ignored
ORA-06550: linia 8, kolumna 6:
PLS-00382: wyrażenie jest niewłaściwego typu
ORA-06550: linia 8, kolumna 1:
PL/SQL: Statement ignored
06550. 00000 - "line %s, column %s:\n%s"
*Cause: Usually a PL/SQL compilation error.
*Action:

przypisanie takie nie zawsze jest możliwe

Metody abstrakcyjne

```
DECLARE
fi figura:=new figura('Czerwony');
kw kwadrat:=new kwadrat('Zielony',10);
ko kolo:=new kolo('Niebieski',6);

BEGIN

kw:= ko;
ko:= kw;

END;
```

Error report -
ORA-06550: linia 7, kolumna 6:
PLS-00382: wyrażenie jest niewłaściwego typu
ORA-06550: linia 7, kolumna 1:
PL/SQL: Statement ignored
ORA-06550: linia 8, kolumna 6:
PLS-00382: wyrażenie jest niewłaściwego typu
ORA-06550: linia 8, kolumna 1:
PL/SQL: Statement ignored
06550. 00000 - "line %s, column %s:\n%s"
*Cause: Usually a PL/SQL compilation error.
*Action:

przypisania takie są niedozwolone

Polimorfizm

```
CREATE TABLE FIGURY OF FIGURA;
```

⇧

tworzy tabelę obiektową typu FIGURA - tabelę można traktować jako zbiór zmiennych typu FIGURA

```
INSERT INTO FIGURY VALUES (NEW KWADRAT('Czarny',10));

INSERT INTO FIGURY VALUES (NEW KOLO('Zolty',6));
```

⇧

wstawia to tabeli FIGURY obiekty typów KWADRAT i KOLO

Polimorfizm

zapytanie odczytuje wartość wszystkich dostępnych atrybutów z tabeli FIGURY

⇩

```
SELECT * FROM FIGURY;
```

	KOLOR
1	Czarny
2	Zolty

Dynamiczne wiązanie metod

```
declare
kw figura:=new kwadrat('Zielony',10);
ko figura:=new kolo('Niebieski',6);
begin
dbms_output.put_line(kw.pole());
dbms_output.put_line(ko.pole());
end;
```

- do zmiennych Kw i Ko typu FIGURA przypisywane są obiekty typu KWADRAT i KOLO
- przez te zmienne, aktywowane są metody POLE
  - zmienna przechowującą kwadrat o boku 10, powoduje wykonanie kodu obliczającego pole kwadratu
  - zmienna przechowującą koło o promieniu 6, powoduje wykonanie kodu obliczającego pole koła

Dynamiczne wiązanie metod

```
SELECT VALUE(X).POLE() FROM FIGURY X;
```

	VALUE(X).POLE()
1	100
2	113

- zapytanie odczytuje wszystkie obiekty z tabeli obiektowej FIGURY i dla każdego z tych obiektów aktywowana jest metoda POLE i zwracane są wyniki

Przykład

- Zbuduj hierarchię typów obiektowych reprezentujących zwierzęta:

  typ DRAPIEZNIK

  i dziedziczące z niego TYGRYS i PANTERA
- W typie obiektowym DRAPIEZNIK zdefiniuj atrybut LICZBA_OFIAR i abstrakcyjną metodę ILE_UPOLOWAL
- Metodę zaimplementuj w odpowiednich podtypach
- Metoda powinna być funkcją, która zwraca komunikat zależny od aktualnego typu obiektowego i liczby ofiar zapisanej w obiekcie.
- Działanie metod przetestuj

Przykład

```
create or replace type drapieznik as object (
liczba_ofiar numeric,
not instantiable member function ile_upolowal
return varchar2
) not final not instantiable;
```

```
create or replace type tygrys under drapieznik (
overriding member function ile_upolowal
return varchar2
);
```

```
create or replace type pantera under drapieznik (
overriding member function ile_upolowal
return varchar2
);
```

Przykład

```
create or replace type body tygrys as
overriding member function ile_upolowal
return varchar2 as
begin
return 'Tygrys upolowal '||liczba_ofiar||' zwierzat';
end;
end;
```

```
create or replace type body pantera as
overriding member function ile_upolowal
return varchar2 as
begin
return 'Pantera upolowala '||liczba_ofiar||' zwierzat';
end;
end;
```

Przykład

```
CREATE TABLE ZWIERZETA OF DRAPIEZNIK;
```

⇧

Utworzenie tabeli obiektowej ZWIERZETA, która przechowuje obiekty typu DRAPIEZNIK.

```
INSERT INTO ZWIERZETA VALUES (TYGRYS(5));
INSERT INTO ZWIERZETA VALUES (TYGRYS(2));
INSERT INTO ZWIERZETA VALUES (TYGRYS(8));
INSERT INTO ZWIERZETA VALUES (PANTERA(1));
INSERT INTO ZWIERZETA VALUES (PANTERA(7));
INSERT INTO ZWIERZETA VALUES (PANTERA(3));
```

⇧

Wpisanie do tabeli kilka obiektów typu TYGRYS oraz PANTERA z różnymi liczbami upolowanych zwierząt

Przykład

```
SELECT VALUE(D).ILE_UPOLOWAL()
FROM ZWIERZETA D;
```

	VALUE(D).ILE_UPOLOWAL()
1	Tygrys upolowal 5 zwierzat
2	Tygrys upolowal 2 zwierzat
3	Tygrys upolowal 8 zwierzat
4	Pantera upolowala 1 zwierzat
5	Pantera upolowala 7 zwierzat
6	Pantera upolowala 3 zwierzat

Operator TREAT w PL/SQL

```
declare
f1 figura:=new kwadrat('Zielony',10);
f2 figura:=new kolo('Niebieski',6);
kw kwadrat;
ko kolo;
dlugosc_boku numeric(5,2);
promien numeric(5,2);
begin
kw:=treat(f1 as kwadrat);
ko:=treat(f2 as kolo);
```

```
dlugosc_boku:=treat(
f1 as kwadrat).dlugosc_boku;
promien:=treat(
f2 as kolo).promien;
end;
```

- poprzez zmienne nadtypu dostępne są atrybuty, które zadeklarowe są w nadtypie
- atrybuty dla podtypów nie są dostępne bezpośrednio.
- Dostęp do atrybutów podtypu umożliwia operator TREAT - zmienia typ zmiennej na typ podany po słowie kluczowym AS

Operator TREAT w PL/SQL

```
declare
f1 figura:=new kwadrat('Zielony',10);
f2 figura:=new kolo('Niebieski',6);
kw kwadrat;
ko kolo;
dlugosc_boku numeric(5,2);
promien numeric(5,2);
begin
kw:=treat(f1 as kwadrat);
ko:=treat(f2 as kolo);
```

```
dlugosc_boku:=treat(
f1 as kwadrat).dlugosc_boku;
promien:=treat(
f2 as kolo).promien;
--kw:=treat(f2 as kwadrat); --blad
--ko:=treat(f1 as kolo); --blad
end;
```

Operator TREAT w SQL

```
SELECT KOLOR,
TREAT(VALUE(X) AS KWADRAT).DLUGOSC_BOKU AS BOK,
TREAT(VALUE(X) AS KOLO).PROMIEN AS PROMIEN
FROM FIGURY X;
```

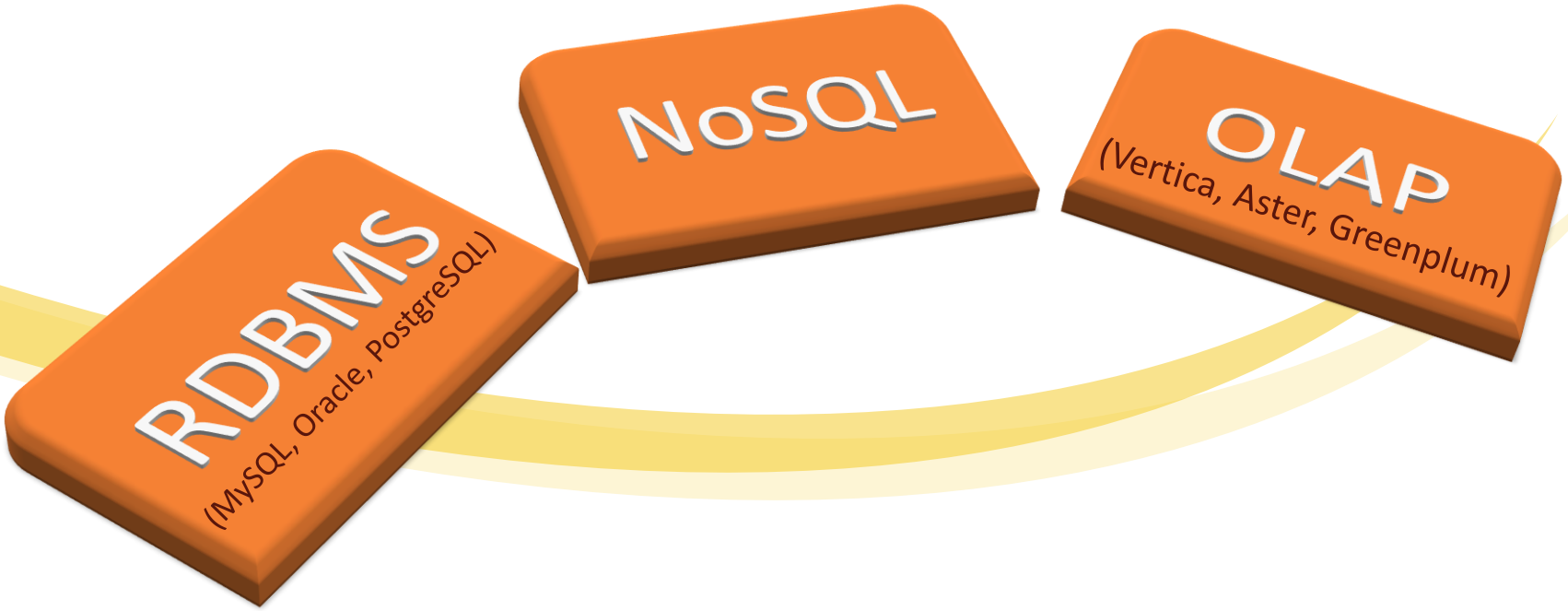
	KOLOR	BOK	PROMIEN
1	Czarny	10	(null)
2	Zolty	(null)	6

Operator IS OF

zmienna IS OF (typ obiektowy) - sprawdza, czy obiekt zapisany w zmiennej jest typu podanego jako parametr albo dowolnego podtypu tego typu i jeżeli tak jest, to zwraca wartość logiczną TRUE, a w przeciwnym wypadku zwraca FALSE

zmienna IS OF (ONLY typ obiektowy) - sprawdza, czy obiekt zapisany w zmiennej jest dokładnie tego typu co podany jako parametr

Rodzaje baz danych



SQL

- relacyjny model danych
- język SQL
- transakcje
- ACID
  - Atomicity (atomowość)
  - Consistency (spójność)
  - Isolation (izolacja)
  - Durability (trwałość)

ACID

ATOMICITY CONSISTENCY ISOLATION DURABILITY

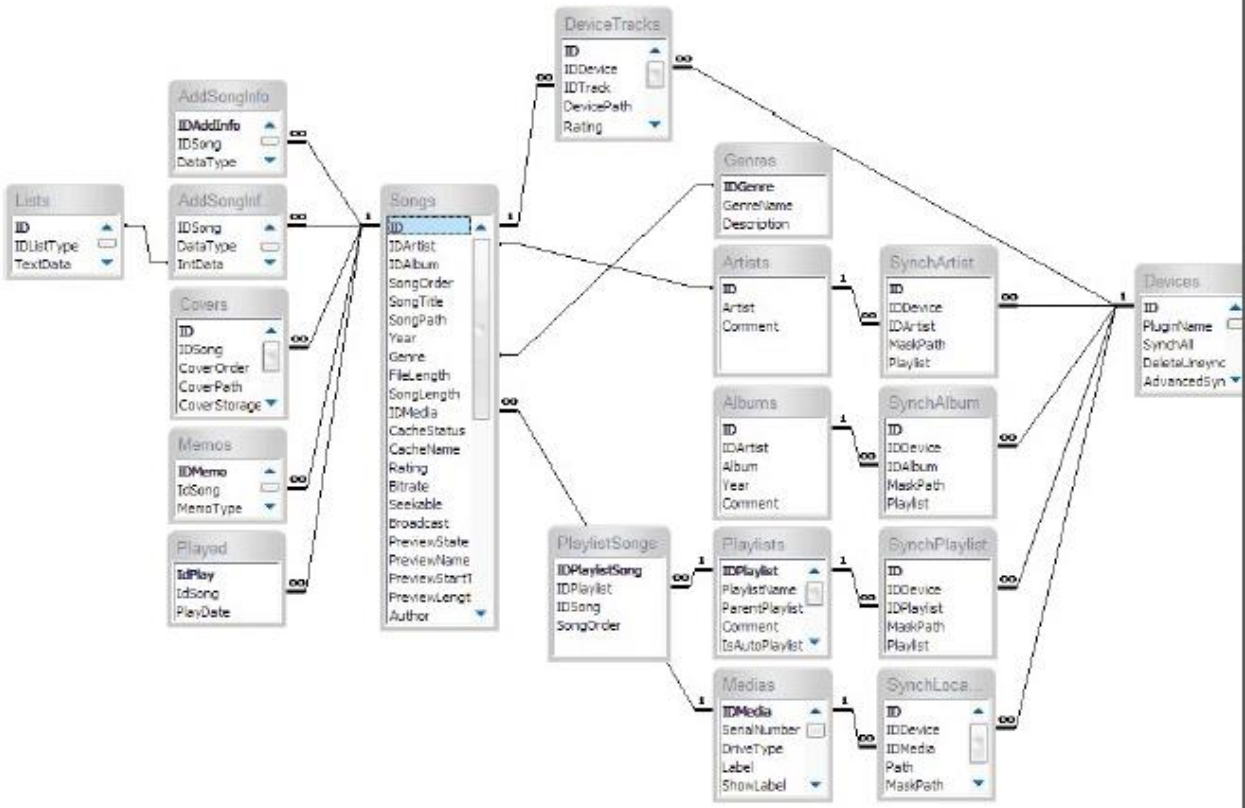
salowalność

- skalowanie wertykalne
  (ang. scale up)

- skalowanie horyzontalne
  (ang. scale out)

SQL

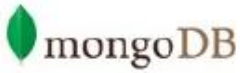
wady



- Rozbudowa
- Normalizacja
- Zmiany
- Efektywność

NoSQL

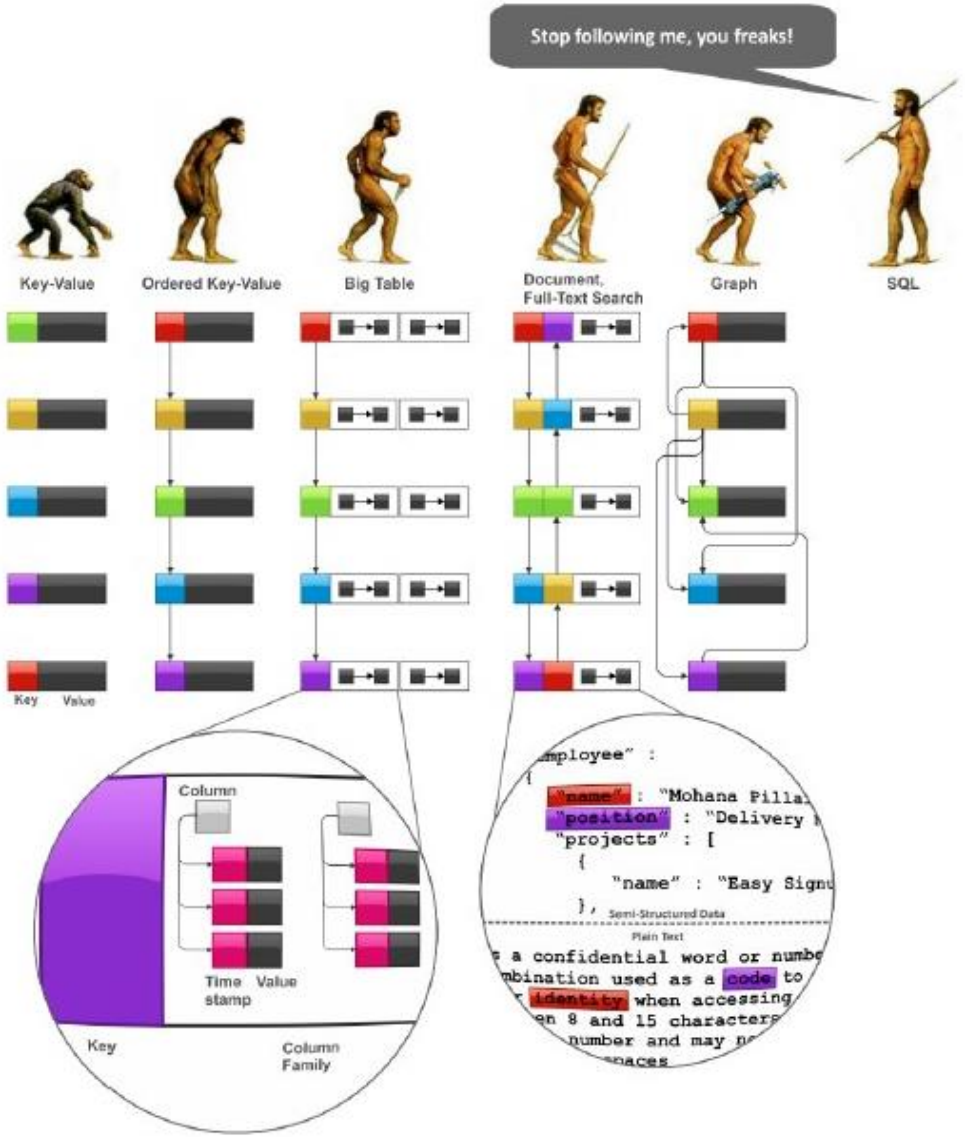
Not Only SQL



- rezygnacja z ACID
- skalowność
- schemat danych
- odporna na awarie
- rozproszona
- brak języka SQL
- nierelacyjny model danych
- łatwa skalowalność pozioma

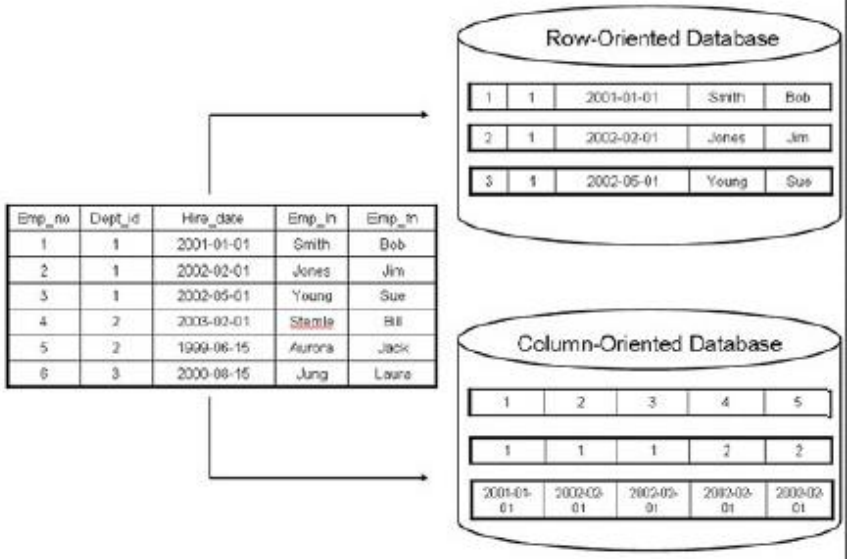
NoSQL Modele

- Bazy klucz-wartość
- Bazy kolumnowe
- Bazy dokumentowe
- Bazy grafowe
- Bazy XML
- Bazy obiektowe



NoSQL Bazy kolumnowe

- wzorowane na BigTable (Google)
- każdy wiersz może mieć przyporządkowany inny zestaw kolumn
- częściowo ustrukturalizowane
- Cassandra, SimpleDB, HBase



NoSQL Bazy klucz-wartość

- przechowują pary klucz-wartość
- dostęp do danych jedynie po kluczu
- Redis, Dynamo, Riak, Berkeley DB, OpenLDAP

NoSQL
Not Only

```
{ imie:        "Jan",
  nazwisko:    "Kowalski",
  nr_indeksu: 98765,
  oceny:       [5, 4.5, 3, 4]
  dzienny:     true }
```

Bazy dokumentowe

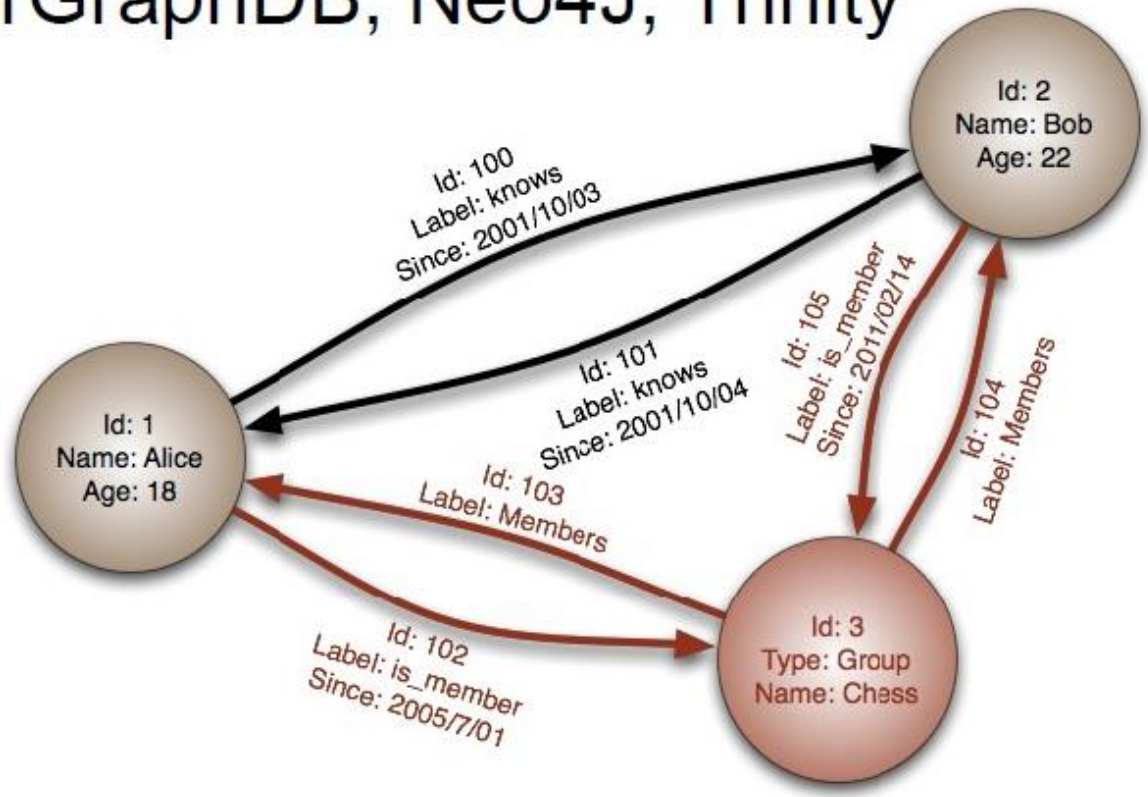
```
db.students.find({nazwisko:
                  "Kowalski"})
```

- przechowuje dokumenty zamiast wierszy/rekordów
- dokument: wpis w bazie składający się z pól (nazwa-wartość)
- możliwość odwoływania się po polach nie będących kluczem podstawowym
- CouchDB, MongoDB, ThruDB, Lotus Notes

NoSQL Not Only

Bazy grafowe

- węzły, krawędzie (łuki), własności
- szybki dostęp do powiązanych danych
- HyperGraphDB, Neo4J, Trinity



NoSQL Bazy XML

- język XML
- wyszukiwanie przez XQL (XmlQueryLanguage)
- eXist, Sedna, Qizx

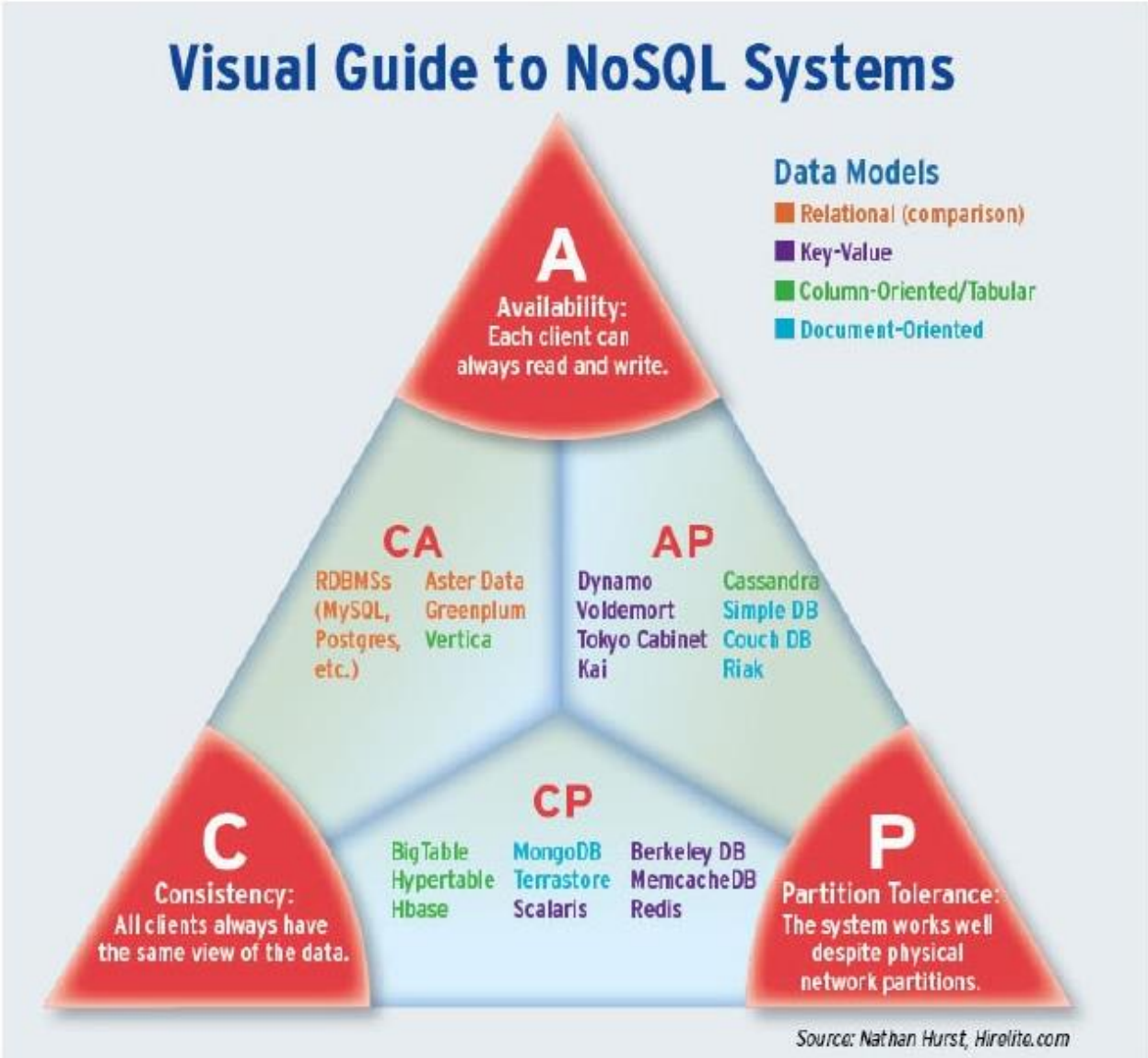
```
<Sensor>
 <name>Sensor 193</name>
 <attributes>
  <Attribute>
   <name>Alpha</name>
   <x>101</x>
   <y>20031</y>
  </Attribute>
  <Attribute>
   <name>Beta</name>
   <x>243</x>
   <y>3037</y>
  </Attribute>
 </attributes>
 <scale>1</scale>
</Sensor>
```

NoSQL Not Only

Bazy obiektowe

- obiektowa
- mała wydajność
- brak optymalizacji zapytań
- dostosowane do jednego języka programowania
- Versant, db4o, Magma, NEO, Datastore

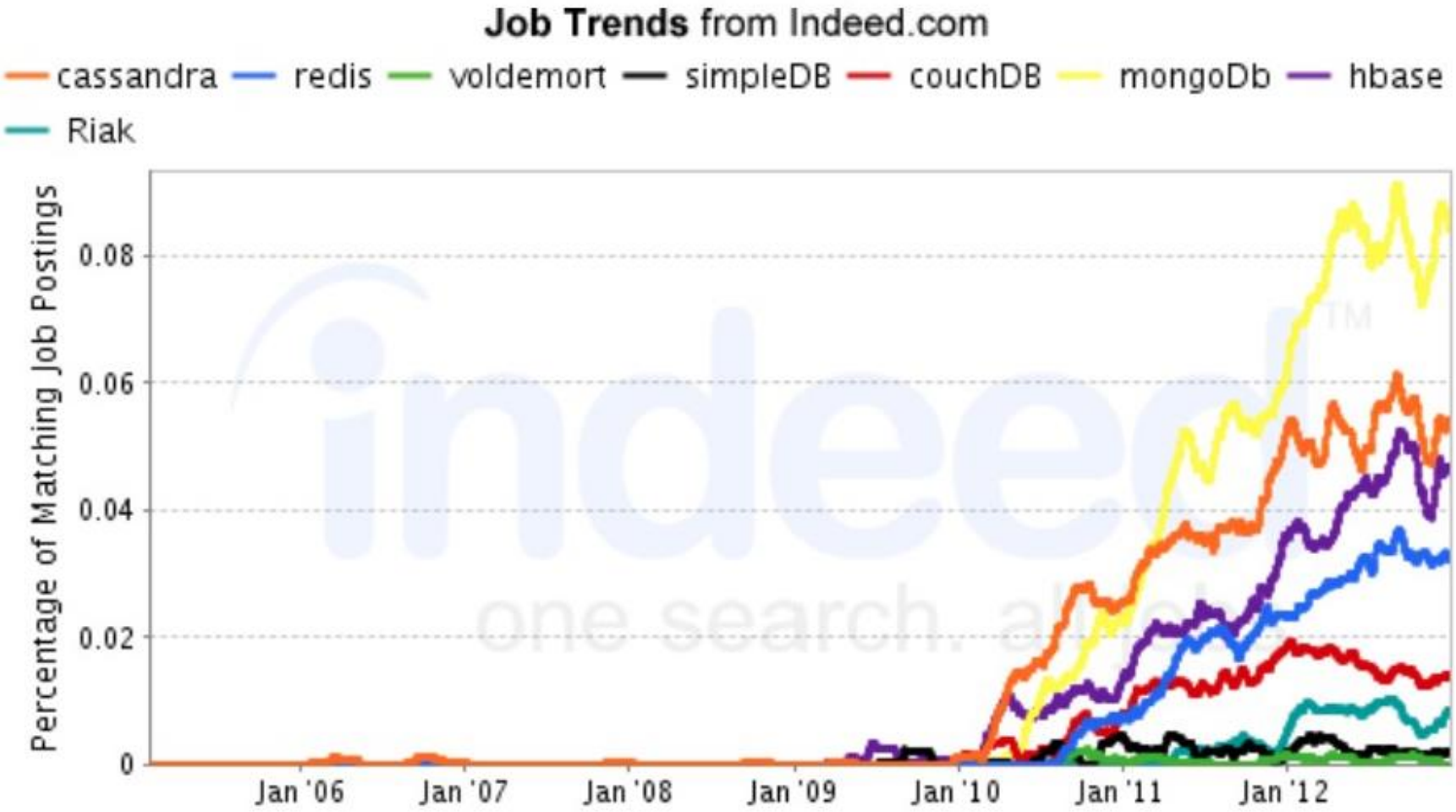
NoSQL Not Only SQL — Które wybrać?



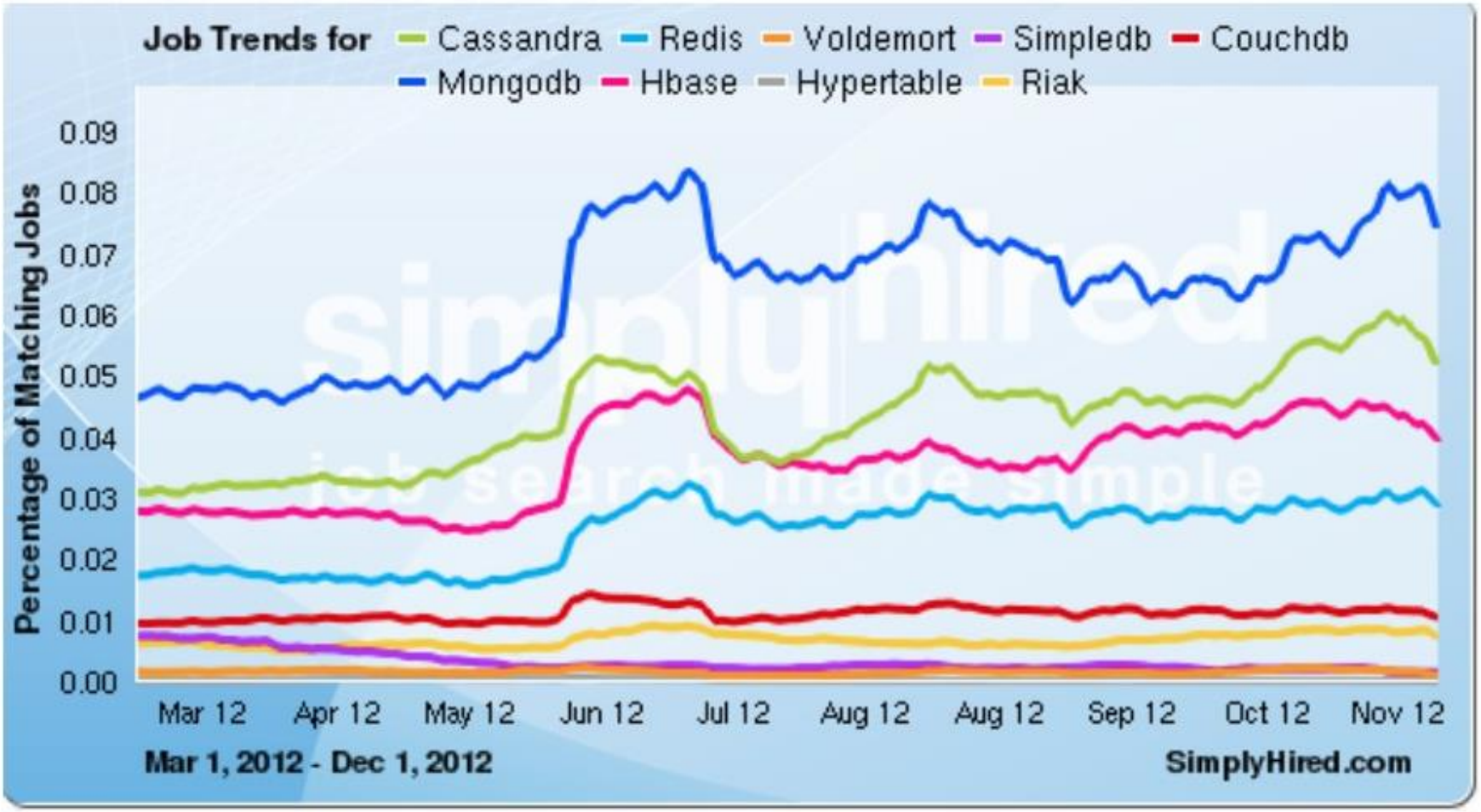
NoSQL Gdzie stosowane?

Who is using them?

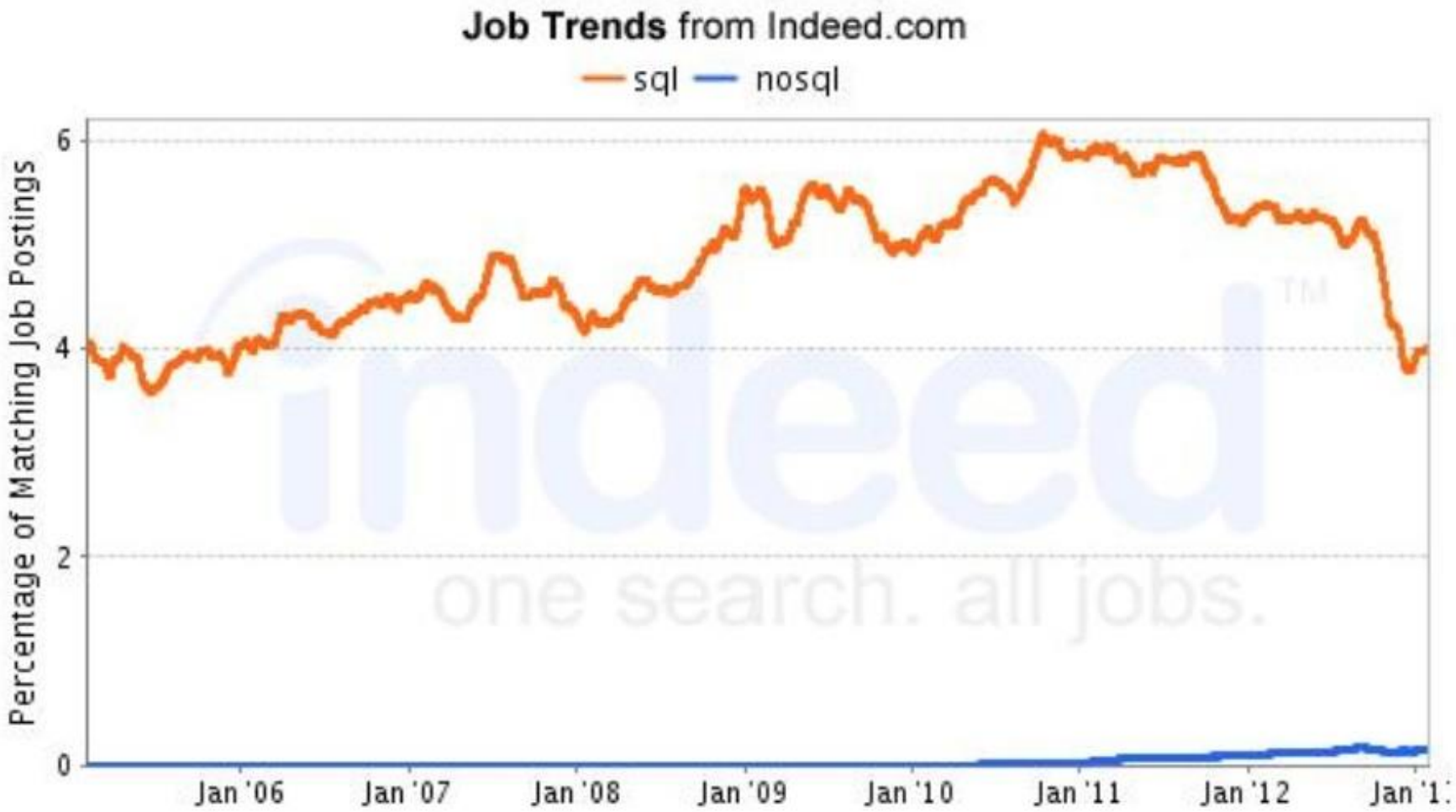
NoSQL Rynek pracy USA



NoSQL
Not Only



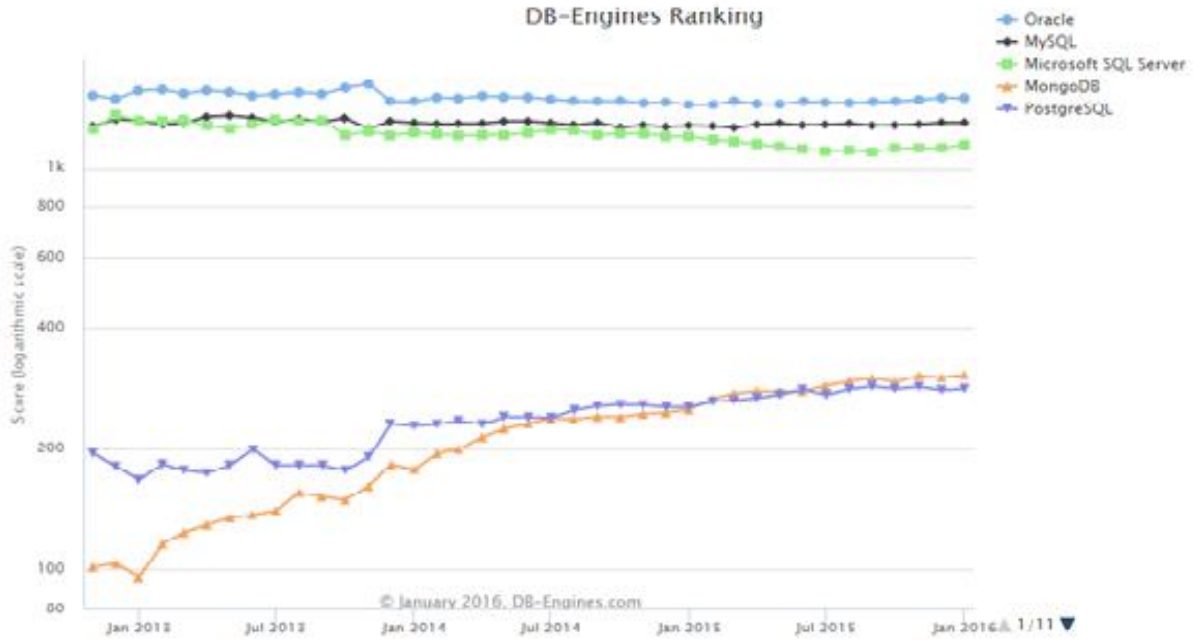
NoSQL
Not Only



Ranking systemów zarządzania operacyjną bazą danych

NoSQL

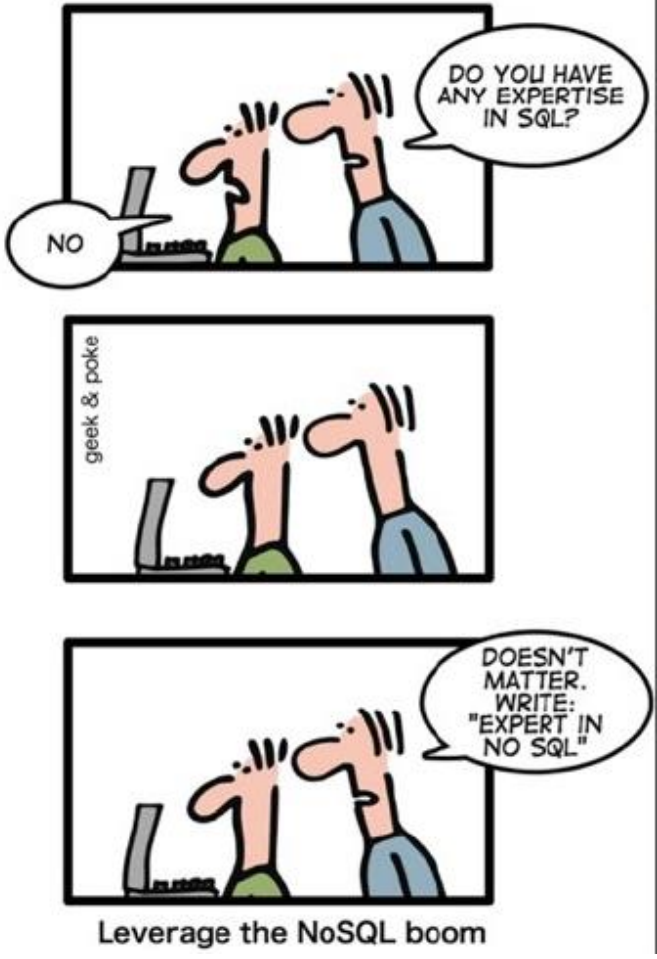
Not Only SQL



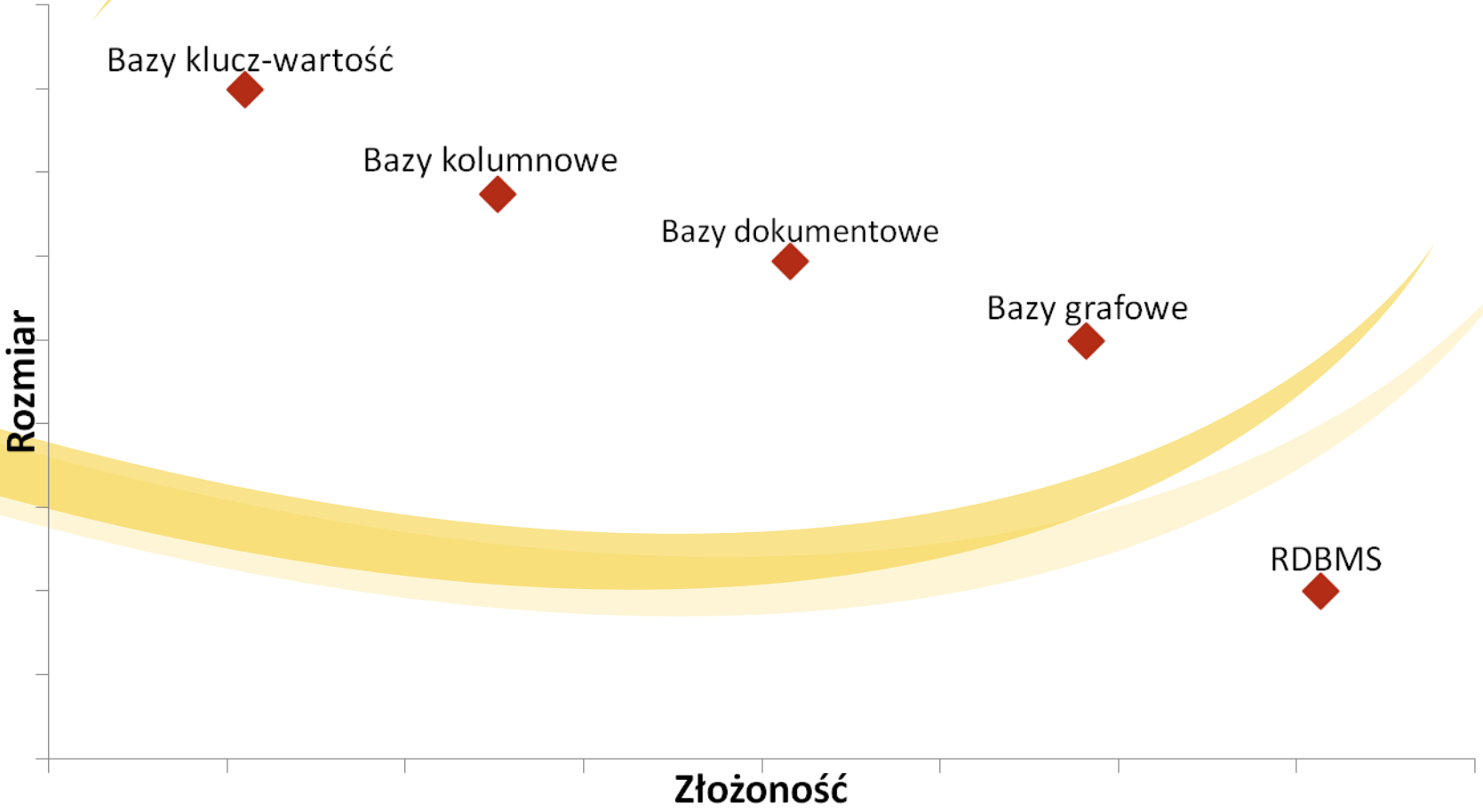
NoSQL (Not Only SQL)

Dlaczego warto się go nauczyć?

- Poziom osobistej satysfakcji
- Poziom technicznej satysfakcji
- Poziom finansowej satysfakcji



Skalowalność baz



Relacyjna vs. Dokumentowa BD

Relacyjne bazy danych	Dokumentowe bazy danych
Predefiniowana struktura danych	Dynamiczna struktura danych
Jednakowe tabele	Kolekcje dokumentów o tej samej lub różnej strukturze
Dane znormalizowane Ograniczona nadmiarowość	Dane zdenormalizowane Występuje redundancja
Wymagana znajomość schematu dla operacji read/write	Wymagana nazwa dokumentu
Dynamiczne zapytania dla statycznej struktury	Statyczne zapytania dla dynamicznej struktury

Składowanie danych

- Binary JSON
- Binarnie zakodowany zserializowany JSON
- MongoDB JavaScript shell
- Biblioteki językowe MongoDB

Przykład:

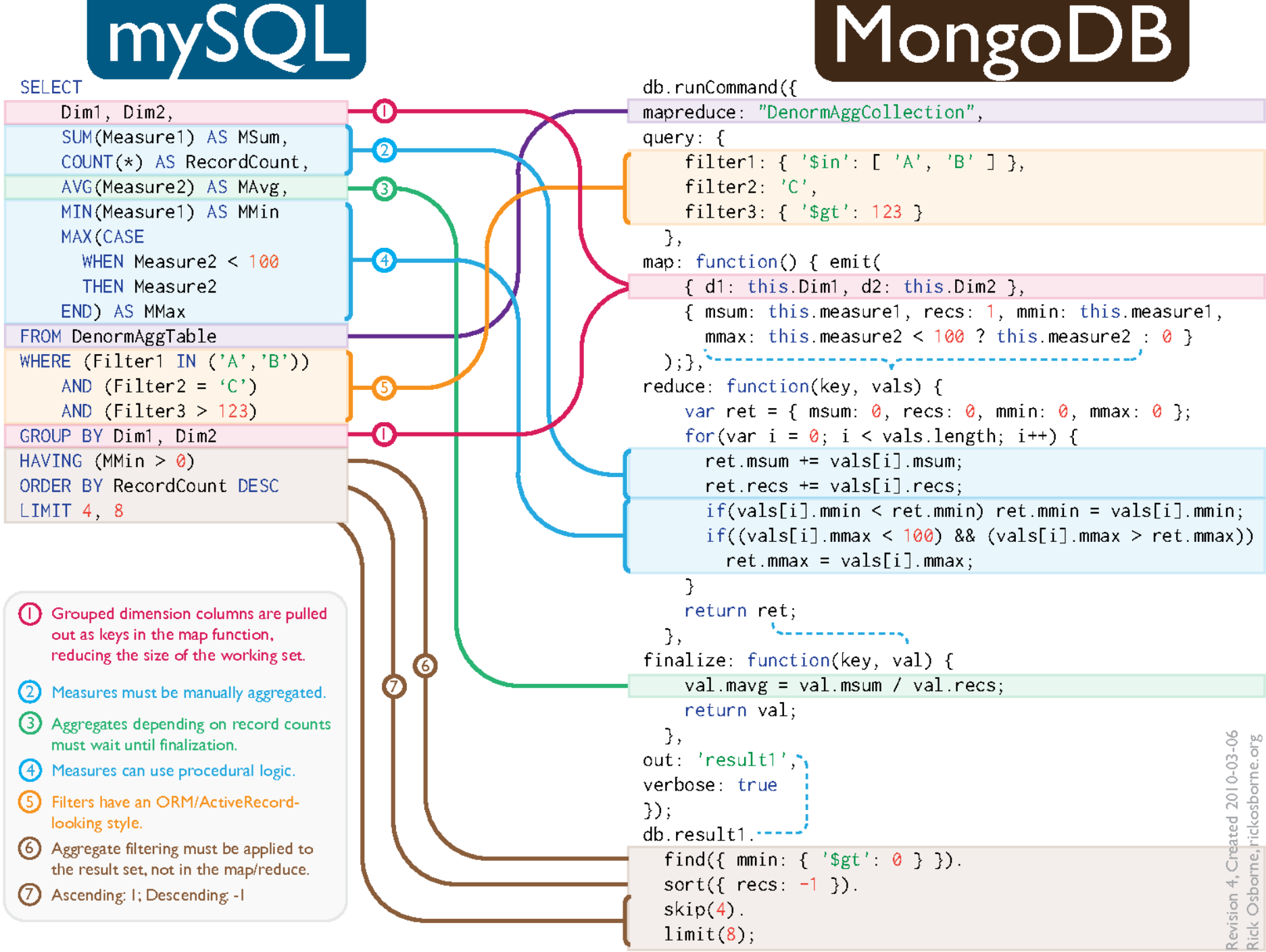
```
{"hello": "world"}
```

→

```
"\x16\x00\x00\x00\x02hello\x00
 \x06\x00\x00\x00world\x00\x00"
```

GridFS

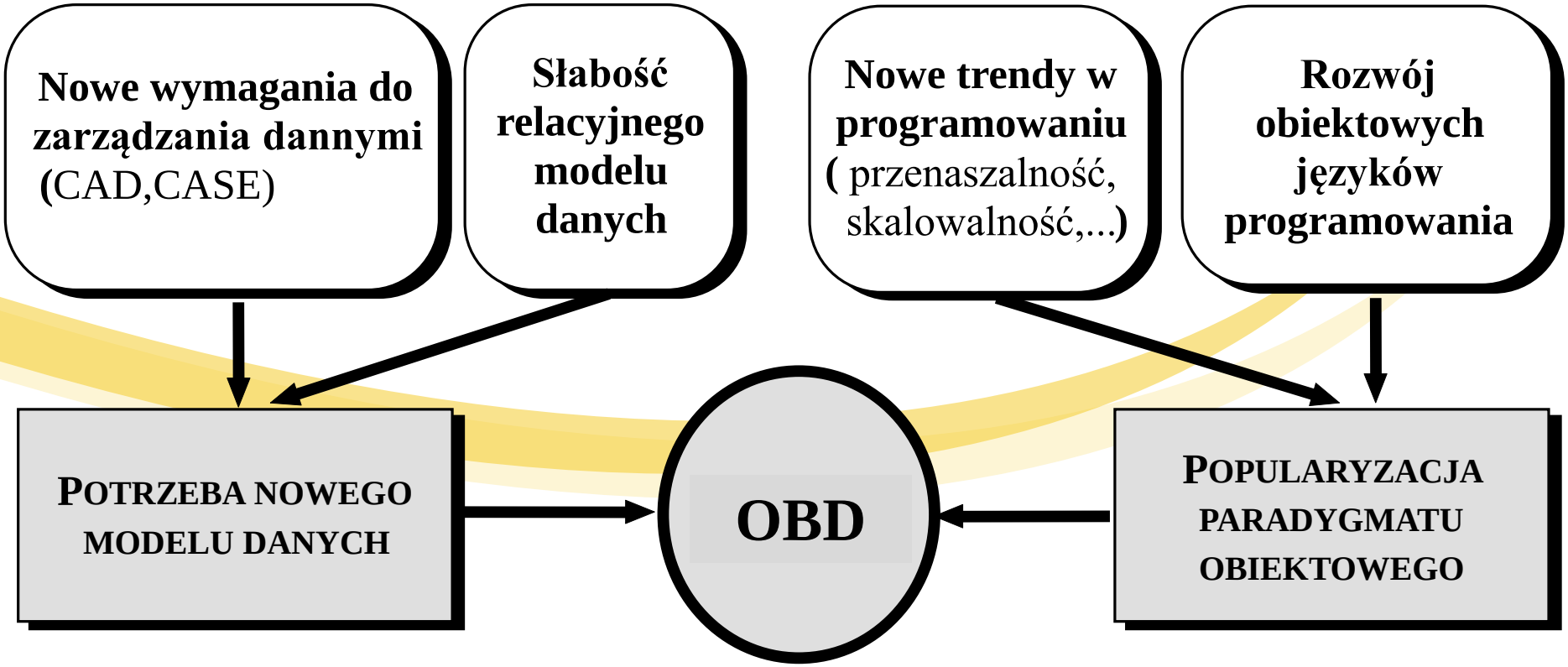
- Maks. rozmiar dokumentu BSON 16MB
- Większe dokumenty przechowywane są za pomocą GridFS
- GridFS dzieli dokument na części
- Każda z nich traktowana jest jako osobny plik
- GridFS wykorzystuje dwie kolekcje:
  - Fragmenty plików
  - Metadane
- Stanowi również API do przechowywania innych plików np. PDF


mySQL
MongoDB
SELECT
Dim1, Dim2,
SUM(Measure1) AS MSum,
COUNT(*) AS RecordCount,
AVG(Measure2) AS MAvg,
MIN(Measure1) AS MMin
MAX(CASE
WHEN Measure2 < 100
THEN Measure2
END) AS MMax
FROM DenormAggTable
WHERE (Filter1 IN ('A','B'))
AND (Filter2 = 'C')
AND (Filter3 > 123)
GROUP BY Dim1, Dim2
HAVING (MMin > 0)
ORDER BY RecordCount DESC
LIMIT 4, 8
db.runCommand({
mapreduce: "DenormAggCollection",
query: {
filter1: { '$in': ['A', 'B'] },
filter2: 'C',
filter3: { '$gt': 123 }
},
map: function() { emit(
{ d1: this.Dim1, d2: this.Dim2 },
{ msum: this.measure1, recs: 1, mmin: this.measure1,
mmax: this.measure2 < 100 ? this.measure2 : 0 }
);},
reduce: function(key, vals) {
var ret = { msum: 0, recs: 0, mmin: 0, mmax: 0 };
for(var i = 0; i < vals.length; i++) {
ret.msum += vals[i].msum;
ret.recs += vals[i].recs;
if(vals[i].mmin < ret.mmin) ret.mmin = vals[i].mmin;
if((vals[i].mmax < 100) && (vals[i].mmax > ret.mmax))
ret.mmax = vals[i].mmax;
}
return ret;
},
finalize: function(key, val) {
val.mavg = val.msum / val.recs;
return val;
},
out: 'result1',
verbose: true
});
db.result1.
find({ mmin: { '$gt': 0 } }).
sort({ recs: -1 }).
skip(4).
limit(8);
1 Grouped dimension columns are pulled out as keys in the map function, reducing the size of the working set.
2 Measures must be manually aggregated.
3 Aggregates depending on record counts must wait until finalization.
4 Measures can use procedural logic.
5 Filters have an ORM/ActiveRecord-looking style.
6 Aggregate filtering must be applied to the result set, not in the map/reduce.
7 Ascending: 1; Descending: -1
Revision 4, Created 2010-03-06
Rick Osborne, rickosborne.org

Obiektowy model danych

- Identyfikatory obiektów
- Powiązania referencyjne między obiektami
- Atrybuty wielowartościowe
- Wyrażenia ścieżkowe
- Hierarchie rozszerzeń klas
- Duże obiekty
- Nowy model przetwarzania danych

Jak doszło do powstania modelu obiektowego?



Do klasycznych funkcji obsługiwanych przez OSZBD można zaliczyć:

- zarządzanie pamięcią zewnętrzną,
- zarządzanie schematem,
- sterowanie współbieżnością,
- zarządzanie transakcjami,
- odtwarzalność,
- przetwarzanie zapytań,
- kontrolę dostępu.

OSZBD dokładają dodatkowo:

- złożone obiekty,
- typy definiowane przez użytkownika,
- tożsamość obiektów,
- hermetyzację,
- typy i/lub klasy oraz ich hierarchię,
- przesłanianie, przeciążanie, późne wiązanie,
- kompletność obliczeniową (pragmatyczną).

Trwałość obiektu

- Trwałe obiekty
- Ulotne obiekty