

**PL/SQL**

# Références

---

- Documentation Oracle en ligne :

<https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/index.html>

(exemples de ces diapos issus de la doc Oracle)

- Oracle 11g - SQL, PL/SQL, SQL Plus. Jérôme Gabillaud, Eni Eds, Mars 2009
- SQL pour Oracle - Applications avec Java, PHP et XML - Optimisation des requêtes et schémas - Avec 50 exercices corrigés. Christian Soutou. Collection Noire. Eyrolles. Février 2013

# Qu'est-ce que c'est ?

---

- Procedural Language / Structured Query Language
- Un langage procédural
- Combinaison
  - de SQL
  - de caractéristiques procédurales (gestion des variables, structures conditionnelles, itératives, ...)
  - de fonctionnalités supplémentaires (déclencheurs, gestion des curseurs, traitement d'erreurs, ...)
- Avec SQL il est possible de créer des scripts enchaînant des commandes, mais pas d'exécuter telle commande si...

# Langage procédural

---

- PL/SQL : spécifique à Oracle
- SQLServer => TransacSQL
- Postgresql => PL/pgSQL
- MySQL => SQL/PSM

Il y a des différences de syntaxe, attention aux recherches d'information !

# Tour d'horizon

---

- Les curseurs : pour parcourir et exploiter les résultats d'une requête
- Les sous-programmes stockés : procédures et fonctions stockées, pour regrouper et exécuter des traitements du côté du serveur au lieu de l'application (moins de charge réseau, utilisation optimale du SGBD – les sous-programmes sont précompilés, gain de temps sur l'interprétation par le moteur SQL)
- Les déclencheurs (triggers) : bloc PL/SQL associé à une table ou vue, dont l'exécution est déclenchée par une opération de mise à jour
- Les exceptions : pour la gestion des erreurs
- Les packages : regroupent un ensemble de types, d'objets liés (cohérents)

# Utilisation de code PL/SQL

---

3 formes :

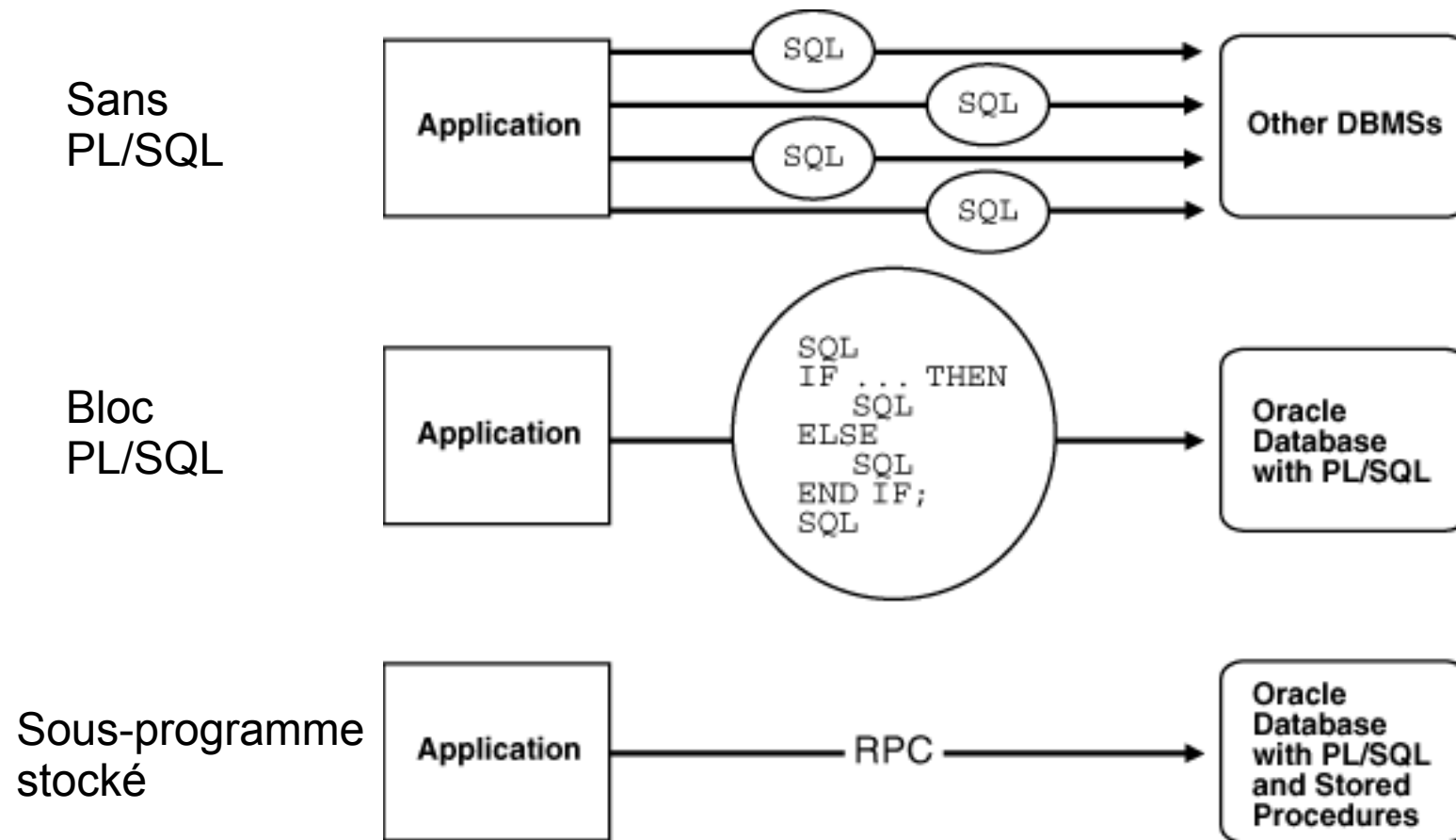
- Un bloc de code exécuté comme une commande SQL (sous SQL\*Plus par exemple)
- Un fichier de commandes PL/SQL
- Un programme stocké

# Avantages d'un programme stocké

- Le code est stocké dans la base de données et peut être **partagé** dans un contexte multi-utilisateurs
- Le code est exécuté par le SGBD
  - => **portabilité** des applications
  - => **applications légères**, chargement uniquement de l'interface graphique
  - => le code est **compilé une seule fois** (et lorsqu'un objet cité dans le code a été modifié), pas à chaque exécution
- **Performance** : réduction du nombre d'appels à la base

# Performances

- Plusieurs requêtes envoyées à la base en une fois → réduction du trafic entre la base et une application



[http://docs.oracle.com/cd/B19306\\_01/appdev.102/b14261/overview.htm](http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/overview.htm)



# Avantages PL/SQL

---

- Autre possibilité de **sécurisation des données** : on peut mettre des droits d'exécution sur les procédures/fonctions stockées, qui une fois lancées s'exécutent avec les droits de l'utilisateur qui les a définies, pas celui qui les a lancées (option par défaut)

Exemple. Un utilisateur peut modifier une table au travers d'une procédure stockée pour laquelle il a reçu les droits d'exécution sans avoir le droit de modification sur cette table

- Gestion des **exceptions**

# Utilisation de code PL/SQL

---

3 formes :

- Un bloc de code exécuté comme une commande SQL (sous SQL\*Plus par exemple)
- Un fichier de commandes PL/SQL
- Un programme stocké

# Bloc PL/SQL anonyme

---

- Interprétation d'un ensemble de commandes, compilé et exécuté par le moteur PL/SQL
- Syntaxe

```
[ DECLARE  
    -- déclaration de variables ]  
  
BEGIN  
    -- instructions SQL, PL/SQL, au  
    minimum NULL  
  
[ EXCEPTION  
    --traitement des erreurs ]  
  
END ;    - fin du bloc
```

# Les variables

---

- Déclaration :

`<nom_var> [constant] <type> [ <:= val> ] ;`

`DECLARE`

```
nouveau_sal number ;
ancien_sal emp.sal%TYPE ;
employe emp%ROWTYPE ;
```

Ou en dehors d'un bloc (et utilisable à l'intérieur d'un bloc précédée de :)

```
variable x NUMBER;
```

- Affectation avec l'opérateur `:=` ou directive `INTO` du `SELECT` (spécifique à PL/SQL)
- Constantes `C CONSTANT NUMBER := 10;`

# Exemple de bloc PL/SQL

**DECLARE**

```
salary NUMBER;
```

**BEGIN**

```
SELECT sal INTO salary  
FROM emp  
WHERE empno=8000;
```

```
INSERT INTO payincreases(numemp, dateinc,  
oldsal, amount) VALUES(8000, sysdate, salary, 100);
```

```
UPDATE emp SET sal=sal+100  
WHERE empno=8000;
```

**END;**



Nécessaire pour lancer la compilation et  
l'exécution sous SQL\*Plus

Remarque : Les blocs peuvent être imbriqués

# Exemple exécution de bloc PL/SQL

---

```
[SQL> SELECT empno, sal FROM emp WHERE empno=8000;
```

EMPNO	SAL
8000	1600

```
DECLARE
```

```
  2      salary NUMBER;
```

```
  3 BEGIN
```

```
    SELECT sal INTO salary
```

```
FROM emp
```

```
WHERE empno=8000;
```

```
  7
```

```
    INSERT INTO payincreases(numemp, dateinc, oldsal, amount) VALUES(8000, sysdate, salary, 100);
```

```
UPDATE emp SET sal=sal+100
```

```
WHERE empno=8000;
```

```
 12 END;
```

```
[ 13 /
```

Procédure PL/SQL terminée avec succès.

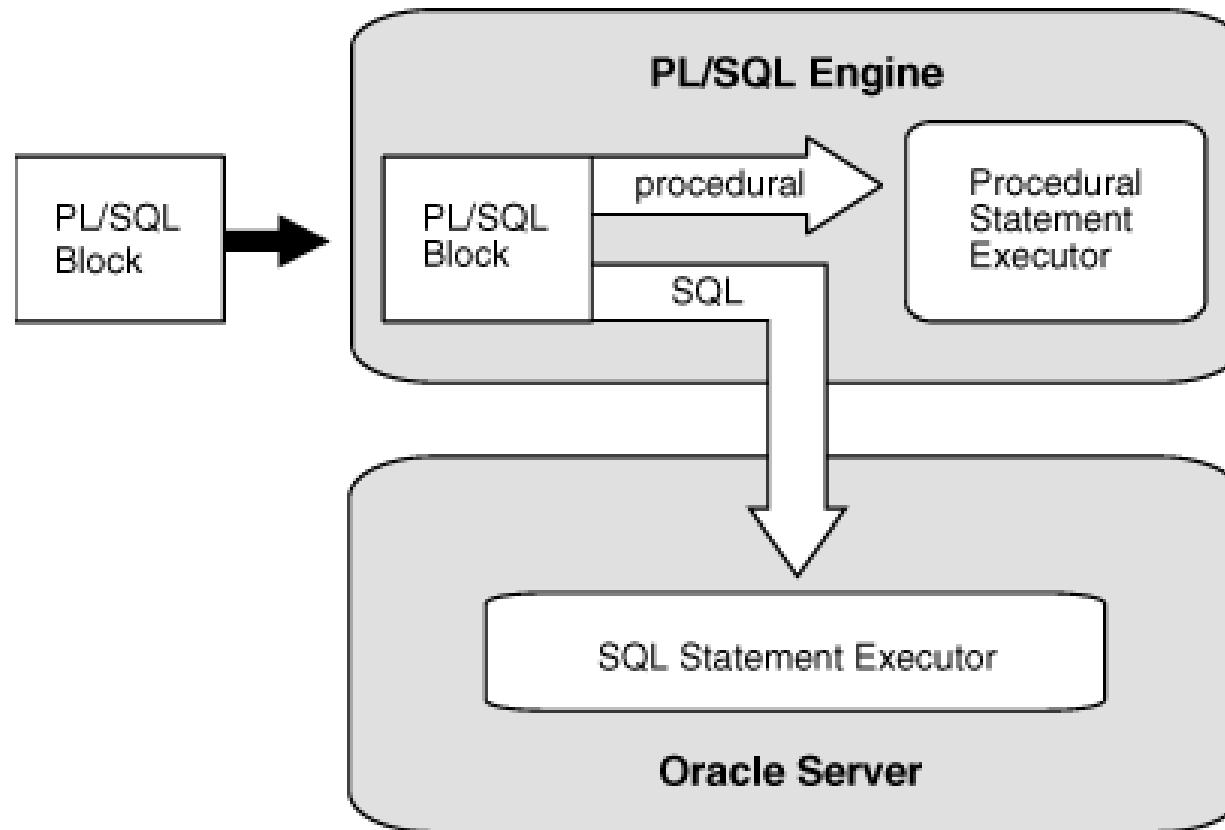
```
[SQL> SELECT empno, sal FROM emp WHERE empno=8000;
```

EMPNO	SAL
8000	1700

```
SQL> █
```

# Interprétation d'un bloc PL/SQL

---



Source : <http://docs.oracle.com/database/121/LNPLS/overview.htm#LNPLS001>

# Exemple de bloc PL/SQL

---

```
DECLARE
    nbDeptWithoutEmp NUMBER(2);

BEGIN
    SELECT COUNT(*) INTO nbDeptWithoutEmp
    FROM dept
    WHERE deptno NOT IN (SELECT deptno FROM emp);
    DBMS_OUTPUT.PUT_LINE('département(s) sans
employés : ' || nbDeptWithoutEmp);
END;
/
```

**NB : DBMS\_OUTPUT nécessite le positionnement du paramètre  
SERVEROUTPUT : SET SERVEROUTPUT ON**

**ATTENTION :** c'est un outil SQL\*Plus, pas PL/SQL

Pas de procédure standard pour afficher à l'écran, PL/SQL est exécuté côté serveur et côté serveur il n'y a pas de terminal associé



# Exemple de bloc PL/SQL

```
[SQL>
DECLARE
  nbDeptWithoutEmp NUMBER(2);
3 BEGIN
  SELECT COUNT(*) INTO nbDeptWithoutEmp
5     FROM dept
  WHERE deptno NOT IN (SELECT deptno FROM emp);
7
  DBMS_OUTPUT.PUT_LINE('département(s) sans employés : ' || nbDeptWithoutEmp);
9 END;
[ 10 /
```

Procédure PL/SQL terminée avec succès.

```
[SQL> set serveroutput on
```

```
DECLARE
  nbDeptWithoutEmp NUMBER(2);
3 BEGIN
  SELECT COUNT(*) INTO nbDeptWithoutEmp
5     FROM dept
  WHERE deptno NOT IN (SELECT deptno FROM emp);
7
  DBMS_OUTPUT.PUT_LINE('département(s) sans employés : ' || nbDeptWithoutEmp);
9 END;
[ 10 /
```

département(s) sans employés : 1

Procédure PL/SQL terminée avec succès.

```
SQL> █
```

# Quelques éléments de syntaxe

---

- Les identificateurs comportent des lettres, chiffres, \$, # ou \_ et 30 caractères maximum

- Les commentaires

-- ceci est un commentaire monoligne

/\* et ceci

est un commentaire multilignes \*/

# Types de données

---

- `char(n)` : chaîne fixe
- `varchar2(n)` : chaîne de longueur variable
- `number` : numérique
- `number(p)` : entier d'au plus  $p$  chiffres
- `number(p,d)` : décimal sur  $p$  positions en tout (séparateur compris), dont  $d$  chiffres décimaux
- `date`

Des types sont ajoutés par rapport à SQL

Liste :

[http://docs.oracle.com/cd/B28359\\_01/appdev.111/b28370/datatypes.htm#CJAEDAEA](http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/datatypes.htm#CJAEDAEA)

# Types dérivés

---

Référence à (au type d') une entité existante

- `%TYPE` permet de référencer le type soit de la colonne d'une table (1), soit d'une variable définie précédemment (2)

`salaires emp.sal%TYPE; (1)`

`variable2 variable1%TYPE; (2)`

- `%ROWTYPE` permet de référencer des structures entières de tables ou de curseurs

`mon_emp emp%ROWTYPE;`

# Affectation avec SELECT INTO

---

```
DECLARE
    bonus    NUMBER (8, 2) ;
BEGIN
    SELECT sal * 0.10 INTO bonus
    FROM emp
    WHERE empno = 100;
    DBMS_OUTPUT.PUT_LINE (bonus) ;
END;
/
```

=> le SELECT ne doit retourner qu'une ligne (sinon utilisation d'un curseur)

=> pas de SELECT sans INTO dans un bloc PL/SQL (et pas de INTO en SQL simple)

Ou encore affectation via les paramètres d'un sous-programme (plus tard)

# SELECT INTO

---

Permet d'extraire des données à partir d'un programme  
PL/SQL

```
SELECT liste INTO {nomVarPLSQL [,nomVarPLSQL] }  
FROM nomTable ... ;
```

# Conditionnelle

---

```
IF <condition> THEN
    <code>
[ELSIF <condition> THEN
    <code> ]
[ELSE
    <code> ]
END IF;
```

# Conditionnelle : exemple

---

```
DECLARE
    sales    NUMBER(8,2)  := 12100;
    quota    NUMBER(8,2)  := 10000;
    bonus     NUMBER(6,2);
    emp_id    NUMBER(6)    := 120;
BEGIN
    IF sales > (quota + 200) THEN
        bonus := (sales - quota)/4;
    ELSE
        bonus := 50;
    END IF;
    UPDATE emp
    SET sal = sal + bonus
    WHERE empno = emp_id;
END;
```

/



# CASE

---

```
CASE <variable>
  WHEN <valeur_1> THEN
    <code>
  WHEN <valeur_2> THEN
    <code>

  [...]
  WHEN <valeur_n> THEN
    <code>
  ELSE
    <code>

END CASE;
```

# Exemple CASE 1

---

```
DECLARE
    grade CHAR(1);
BEGIN
    grade := 'G';
    CASE grade
        WHEN 'A' THEN
            DBMS_OUTPUT.PUT_LINE('Excellent');
        WHEN 'B' THEN
            DBMS_OUTPUT.PUT_LINE('Très bon');
        WHEN 'C' THEN
            DBMS_OUTPUT.PUT_LINE('Bon');
        WHEN 'D' THEN
            DBMS_OUTPUT.PUT_LINE('Passable');
        WHEN 'F' THEN
            DBMS_OUTPUT.PUT_LINE('Faible');
        ELSE DBMS_OUTPUT.PUT_LINE('Cette note n''existe pas');
    END CASE;
END;
/
```

# Exemple CASE 2

---

```
DECLARE
    grade CHAR(1);
    appreciation VARCHAR2(24);

BEGIN
    grade := 'B';
    appreciation := CASE grade
        WHEN 'A' THEN 'Excellent'
        WHEN 'B' THEN 'Très bon'
        WHEN 'C' THEN 'Bon'
        WHEN 'D' THEN 'Passable'
        WHEN 'F' THEN 'Faible'
        ELSE 'Cette note n''existe pas'
    END ;

    DBMS_OUTPUT.PUT_LINE(appreciation);
END;
/
```

# Boucles LOOP

---

LOOP

    <code>

EXIT [WHEN <condition>];

END LOOP;

# Exemple boucle

---

```
DECLARE
    x NUMBER := 0;

BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE('Inside loop:  x = ' ||
TO_CHAR(x));

        x := x + 1;

        EXIT WHEN x > 3;
    END LOOP;

    -- Après EXIT, on reprend ici

    DBMS_OUTPUT.PUT_LINE(' After loop:  x = ' ||
TO_CHAR(x));

END;
```

/

# Boucle WHILE

---

```
WHILE <condition>  
LOOP  
    <code>  
    [EXIT [WHEN <condition>]]  
END LOOP;
```

# Exemple boucle WHILE

---

```
DECLARE
    x NUMBER := 0;

BEGIN
    WHILE x <= 3 LOOP
        DBMS_OUTPUT.PUT_LINE('Inside loop:  x = ' ||
TO_CHAR(x));

        x := x + 1;
    END LOOP;

    -- Après EXIT, on reprend ici

    DBMS_OUTPUT.PUT_LINE(' After loop:  x = ' ||
TO_CHAR(x));
END;

/
```

# Boucle FOR sur intervalle de valeurs

---

```
FOR <variable> in [REVERSE] <borne_inf> ..<  
borne_sup>
```

```
LOOP
```

```
    <code>
```

```
    [EXIT [WHEN <condition>]]
```

```
END LOOP;
```

REVERSE : le compteur est décrémenté



# Exemple boucle FOR

---

```
BEGIN
```

```
    FOR i IN 1..3 LOOP
```

```
        DBMS_OUTPUT.PUT_LINE (TO_CHAR(i) ) ;
```

```
    END LOOP;
```

```
END;
```

```
/
```

# Boucle FOR sur résultat d'une requête

---

Ou

```
FOR <curseur> in (requete)
```

```
LOOP
```

```
    <code>
```

```
END LOOP;
```

ou

```
FOR <variable> in curseur
```

```
LOOP
```

```
    <code>
```

```
END LOOP;
```

# Exemple boucle FOR

---

```
BEGIN
```

```
    FOR curs IN (select empno, job from emp)  
LOOP
```

```
    DBMS_OUTPUT.PUT_LINE(curs.empno || ' :  
' || curs.job);
```

```
END LOOP;
```

```
END;
```

```
/
```

# Exemple boucle FOR

---

```
[SQL>
BEGIN
  FOR curs IN (select empno, job from emp) LOOP
    DBMS_OUTPUT.PUT_LINE(curs.empno || ' : ' || curs.job);
  4   END LOOP;
  5   END;
[ 6 /
9999 :
7839 : PRESIDENT
7698 : MANAGER
7782 : MANAGER
7566 : MANAGER
7654 : SALESMAN
7499 : SALESMAN
7844 : SALESMAN
7900 : CLERK
7521 : SALESMAN
7902 : ANALYST
7369 : CLERK
7788 : ANALYST
7876 : CLERK
7934 : CLERK
8000 : ANALYST
```

Procédure PL/SQL terminée avec succès.

SQL> █

# Boucles avec étiquettes

---

Il est possible d'étiqueter les boucles

**<<etiquette>>**

LOOP

instructions ;

END LOOP ***etiquette***;

Permet la sortie de plusieurs boucles imbriquées : la courante et celle(s) qui l'inclue(nt)

# Directive CONTINUE

---

Interrompt l'itération et revient au début de la structure

CONTINUE [etiquette] [WHEN <condition>] ;

Va

- À la condition pour un WHILE
- À l'itération suivante pour un FOR
- À l'instruction qui suit le LOOP

EXIT interrompt à la fois l'itération et la structure répétitive

# Curseurs explicites

---

- Pointeur vers une zone mémoire privée
- Contiennent le résultat d'une requête et permettent de parcourir les lignes de ce résultat afin de les traiter une à une

- Déclaration

```
CURSOR mon_curseur IS requete;
```

- Ouverture : `OPEN mon_curseur;`

- Traitement (parcours) : ramène l'enregistrement courant

```
FETCH mon_curseur INTO liste_var;
```

- Fermeture : `CLOSE mon_curseur;`

# Attributs des curseurs

---

- `%FOUND` : vaut true si le dernier FETCH a ramené une ligne
- `%NOTFOUND` : vaut true si le dernier fetch n'a pas ramené de ligne
- `%ISOPEN` : vaut true si le curseur est ouvert
- `%ROWCOUNT` : nombre de lignes parcourues par le fetch jusqu'à présent (avant le premier fetch, égal à 0)



# Exemple curseur

---

```
DECLARE
    CURSOR c1 IS
        SELECT ename
        FROM emp
        WHERE empno = 120;

    emp_lig c1%ROWTYPE;

BEGIN

    OPEN c1;
    FETCH c1 INTO emp_lig;
    DBMS_OUTPUT.PUT_LINE('Nom employe : ' ||
emp_lig.ename);
    CLOSE c1;

END;

/
```

# Exemple curseur

---

```
DECLARE
    CURSOR c1 IS
        SELECT ename
        FROM emp;

    emp_lig c1%ROWTYPE;

BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO emp_lig;
        EXIT WHEN c1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Nom employe : ' ||
emp_lig.ename);
    END LOOP;
    CLOSE c1;
END;
/
```

# Exemple curseur

---

```
SQL>
DECLARE
  2   CURSOR c1 IS
  3     SELECT ename
  4     FROM emp;
  5   emp_lig c1%ROWTYPE;
  6 BEGIN
OPEN c1;
LOOP
FETCH c1 INTO emp_lig;
EXIT WHEN c1%NOTFOUND;
  DBMS_OUTPUT.PUT_LINE('Nom employe : ' || emp_lig.ename);
END LOOP;
CLOSE c1;
  14 END;
[ 15 /
Nom employe : ADAMS
Nom employe : ALLEN
Nom employe : BLAKE
Nom employe : CLARK
Nom employe : DUPONT
Nom employe : FORD
Nom employe : GAG
Nom employe : JAMES
Nom employe : JONES
Nom employe : KING
Nom employe : MARTIN
Nom employe : MILLER
Nom employe : SCOTT
Nom employe : SMITH
Nom employe : TURNER
Nom employe : WARD

Procédure PL/SQL terminée avec succès.

SQL> █
```

# La boucle FOR sur un curseur

---

```
FOR rec_emp IN curs_emp LOOP  
    ...  
END LOOP;
```

- Remarques
  - OPEN, FETCH et CLOSE sont faits implicitement
  - le record (ici rec\_emp) est déclaré implicitement

# Curseur implicite

---

- Retourne des informations sur des commandes telles que INSERT, UPDATE, DELETE, SELECT INTO, COMMIT, ou ROLLBACK
- Utilisation de SQL%FOUND, SQL%ISOPEN, SQL%NOTFOUND, et SQL%ROWCOUNT (référence à la dernière commande exécutée)

# Exemple FETCH - Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17/12/80	800		20
7499	ALLEN	SALESMAN	7698	20/02/81	1600	300	30
7521	WARD	SALESMAN	7698	22/02/81	1250	500	30
7566	JONES	PLEADER	7839	02/04/81	2975		20
7654	MARTIN	SALESMAN	7698	28/09/81	1250	1400	30
7698	BLAKE	PLEADER	7839	01/05/81	2850		30
7782	CLARK	PLEADER	7839	09/06/81	2450		10
7788	SCOTT	ANALYST	7566	19/04/87	3000		20
7839	KING	PRESIDENT		17/11/81	5000		10
7844	TURNER	SALESMAN	7698	08/09/81	1500	0	30
7876	ADAMS	CLERK	7788	23/05/87	1100		20
7900	JAMES	CLERK	7698	03/12/81	950		30
7902	FORD	ANALYST	7566	03/12/81	3000		20
7934	MILLER	CLERK	7782	23/01/82	1300		10

# Exemple FETCH – bloc test ROWCOUNT

```
DECLARE
    CURSOR curs_emp IS SELECT * FROM emp;
    rec_emp emp%ROWTYPE;
BEGIN
    -- avant OPEN, ROWCOUNT donne une erreur
    OPEN curs_emp;
    DBMS_OUTPUT.PUT_LINE('Après OPEN - %ROWCOUNT: ' ||
curs_emp%ROWCOUNT);
    LOOP
        FETCH curs_emp INTO rec_emp;
        EXIT WHEN curs_emp%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Après EXIT - %ROWCOUNT: ' ||
curs_emp%ROWCOUNT);
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Après END LOOP - %ROWCOUNT: ' ||
curs_emp%ROWCOUNT);
    CLOSE curs_emp; -- après CLOSE curseur non valide

END;
/
```

# Exemple FETCH – Résultat test ROWCOUNT

---

```
Après OPEN - %ROWCOUNT: 0
Après EXIT - %ROWCOUNT: 1
Après EXIT - %ROWCOUNT: 2
Après EXIT - %ROWCOUNT: 3
Après EXIT - %ROWCOUNT: 4
Après EXIT - %ROWCOUNT: 5
Après EXIT - %ROWCOUNT: 6
Après EXIT - %ROWCOUNT: 7
Après EXIT - %ROWCOUNT: 8
Après EXIT - %ROWCOUNT: 9
Après EXIT - %ROWCOUNT: 10
Après EXIT - %ROWCOUNT: 11
Après EXIT - %ROWCOUNT: 12
Après EXIT - %ROWCOUNT: 13
Après EXIT - %ROWCOUNT: 14
Après END LOOP - %ROWCOUNT: 14
```

Procédure PL/SQL terminée avec succès.



# Exemple FETCH – test FOUND

```
DECLARE
    CURSOR curs_emp IS SELECT * FROM emp;
    rec_emp emp%ROWTYPE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Avant OPEN : curseur non valide, %FOUND
echoue');
    OPEN curs_emp;
    DBMS_OUTPUT.PUT_LINE('Après OPEN - %FOUND');
    IF curs_emp%FOUND THEN DBMS_OUTPUT.PUT_LINE('ok');
    ELSE DBMS_OUTPUT.PUT_LINE('ko'); END IF;
    LOOP
        FETCH curs_emp INTO rec_emp;
        EXIT WHEN curs_emp%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Après FETCH - %FOUND');
        IF curs_emp%FOUND THEN DBMS_OUTPUT.PUT_LINE('ok');
        ELSE DBMS_OUTPUT.PUT_LINE('ko'); END IF;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Après END LOOP - %FOUND');
    IF curs_emp%FOUND THEN DBMS_OUTPUT.PUT_LINE('ok');
    ELSE DBMS_OUTPUT.PUT_LINE('ko'); END IF;
    CLOSE curs_emp;
    DBMS_OUTPUT.PUT_LINE('Après CLOSE : curseur non valide, %FOUND
echoue');
```

# Exemple FETCH – résultat test FOUND

---

Avant OPEN : curseur non valide, %FOUND echoue

Après OPEN - %FOUND

ko

Après FETCH - %FOUND

ok

Après FETCH - %FOUND

ok

...

Après FETCH - %FOUND

ok

Après END LOOP - %FOUND

ko

Après CLOSE : curseur non valide, %FOUND echoue

Procédure PL/SQL terminée avec succès.

# Création de procédures stockées

---

Stocker dans la base de données un bloc PL/SQL muni d'un nom et (si besoin) d'une liste de paramètres

```
CREATE [OR REPLACE] PROCEDURE nom [(params)] [AUTHID  
{CURRENT_USER | DEFINER}] AS  
    declarations  
  
BEGIN  
    instructions  
[EXCEPTION  
    gestionnaire]  
END nom;
```

avec params de la forme

```
nom [IN | OUT | IN OUT] type [, nom [IN | OUT | IN  
OUT] type]...
```

**AUTHID** détermine si la procédure s'exécute avec les privilèges de son propriétaire (par défaut) ou de l'utilisateur courant.

# Exemple procédure stockée

---

```
CREATE OR REPLACE PROCEDURE award_bonus
(emp_id NUMBER, bonus NUMBER) AS
    commission      REAL;
BEGIN
    SELECT comm / 100 INTO commission
    FROM emp
    WHERE empno = emp_id;

    UPDATE emp
    SET sal = sal+bonus*nvl(commission,0)
    WHERE empno = emp_id;

END award_bonus;
/
```

**Remarque : SHOW ERRORS sous SQL\*Plus donne des informations en cas d'erreurs à la compilation**

# Exemple de procédure stockée

---

```
SQL>
CREATE OR REPLACE PROCEDURE award_bonus (
  emp_id NUMBER, bonus NUMBER) AS
  3   commission    REAL;
  4 BEGIN
  5   SELECT comm / 100 INTO commission
  6   FROM emp
  7   WHERE empno = emp_id;
  8   UPDATE emp
  9   SET sal = sal+bonus*nvl(commission,0)
 10   WHERE empno = emp_id;
 11 END award_bonus;
 12 /
```

Procédure créée.

```
SQL> select object_name from user_procedures where lower(object_name)='award_bonus';
```

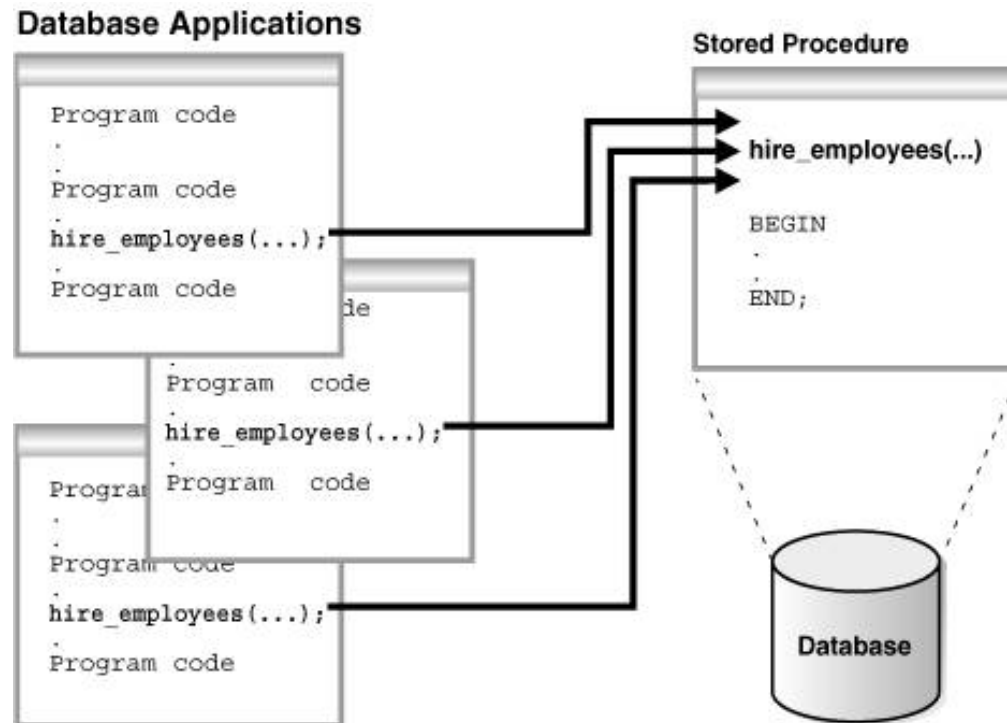
OBJECT_NAME
-------------

AWARD_BONUS
-------------

```
SQL> █
```

# Procédure stockée

---



Source : [http://docs.oracle.com/cd/E25178\\_01/server.11111/e25789/srvrside.htm](http://docs.oracle.com/cd/E25178_01/server.11111/e25789/srvrside.htm)

# Fonctions stockées

---

- Similaire, mais renvoie une valeur

```
CREATE [OR REPLACE] FUNCTION nom (param)
RETURN type AS
    declarations

BEGIN
    instructions
[EXCEPTION
    gestionnaire]

END nom;
```

Une fonction doit contenir l'instruction

```
RETURN expression;
```

# Exemple de fonction stockée

---

```
CREATE FUNCTION countEmployees (department
dept.deptno%TYPE) RETURN NUMBER AS
    total NUMBER;

BEGIN
    SELECT COUNT(*) INTO total
    FROM emp
    WHERE deptno=department;

RETURN total;

END countEmployees;
/
```



# Appel des sous-programmes

---

## Procédures

- Depuis un bloc PL/SQL `nom_proc (params) ;`
- Sous SQL\*Plus : `execute nom_proc (params) ;`

## Fonctions : on utilise la valeur

- `SELECT nom_fonc (params) FROM dual;`
- `execute  
DBMS_OUTPUT.PUT_LINE (nom_fonc (params) )`
- `variable var type_retour_fs  
execute :var := nom_fonc (params)  
print :var`

# Appel d'une procédure stockée

---

```
[SQL> select empno, sal, comm from emp where empno=7521;
```

EMPNO	SAL	COMM
7521	1750	500

```
[SQL> execute award_bonus(7521, 200);
```

Procédure PL/SQL terminée avec succès.

```
[SQL> select empno, sal, comm from emp where empno=7521;
```

EMPNO	SAL	COMM
7521	2750	500

```
SQL> █
```

# Appel d'une fonction stockée

```
CREATE FUNCTION countEmployees(department dept.deptno%TYPE) RETURN NUMBER AS
  2      total NUMBER;
  3 BEGIN
  SELECT COUNT(*) INTO total
  FROM emp
  WHERE deptno=department;
  7
  RETURN total;
  9 END countEmployees;
[ 10 /
```

Fonction créée.

```
[SQL> SELECT countEmployees(10) FROM dual;
```

```
COUNTEMPLOYEES(10)
-----
                    5
```

```
[SQL> execute DBMS_OUTPUT.PUT_LINE(countEmployees(10));
5
```

Procédure PL/SQL terminée avec succès.

```
[SQL> variable compte NUMBER
```

```
[SQL> execute :compte:=countEmployees(10)
```

Procédure PL/SQL terminée avec succès.

```
[SQL> print :compte
```

```
      COMPTE
-----
          5
```

```
SQL> █
```

# Suppression des sous-programmes

---

- DROP PROCEDURE nom\_procedure;
- DROP FUNCTION nom\_fonction;

# Tableaux

---

- Type TABLE
- Ensemble de paires clé-valeur
- Chaque clé est unique, c'est un entier ou une chaîne
- Exemple :

```
TYPE population IS TABLE OF NUMBER  
INDEX BY VARCHAR2(64);
```

```
city_population population;
```

```
city_population('Smallville') := 2000;
```

# Exceptions

---

- Permettent de traiter les erreurs rencontrées à l'exécution
- Déclenchées implicitement (exceptions Oracle prédéfinies ou non) ou explicitement (définies par l'utilisateur)
- Syntaxe :

```
EXCEPTION
    WHEN exception1 THEN
        sequence_of_statements1
    [ WHEN exception2 THEN
        sequence_of_statements2
    ... ]
    [ WHEN OTHERS THEN
        sequence_of_statements3]
END;
```

# Exceptions : exemple

---

```
DECLARE
```

```
    . . .
```

```
BEGIN
```

```
    . . .
```

```
    EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN
```

```
        DBMS_OUTPUT.PUT_LINE
```

```
            ('No Data found for SELECT on ' ||  
temp_var);
```

```
END;
```

```
/
```

# Exceptions

---

- Exemples d'exceptions prédéfinies :  
NO\_DATA\_FOUND, TOO\_MANY\_ROWS,  
INVALID\_CURSOR, CASE\_NOT\_FOUND, ...
- Exception déclenchées explicitement (définies par l'utilisateur) :
  - Déclaration dans le DECLARE :  
`monexcep EXCEPTION ;`
  - Lancement avec `RAISE`



# Package

---

- Permet de stocker des objets PL/SQL (procédures, fonctions, curseurs, variables, ...)
- Deux parties : spécification (déclaration) et corps
- Syntaxe spécification :

```
CREATE [ OR REPLACE ] PACKAGE nompack
    IS [ declaration variables ]
        [ declaration curseurs ]
        [ declaration sous-programmes ]
        [ declaration exceptions ]
END nompack;

/
```

# Package

---

- Syntaxe corps du package

```
CREATE [ OR REPLACE ] PACKAGE BODY nompack IS  
    ...  
END nompack;  
  
/
```

- Appel aux éléments définis dans un package :

```
nompack.objet
```

# Exemple package

```
CREATE PACKAGE package_emp AS  
    CURSOR c1 RETURN emp%ROWTYPE;
```

Spécification

```
    PROCEDURE calc_bonus (date_hired emp.hiredate%TYPE);  
END package_emp;  
/
```

```
CREATE PACKAGE BODY package_emp AS  
    CURSOR c1 RETURN emp%ROWTYPE IS  
        SELECT * FROM emp WHERE sal > 2500;
```

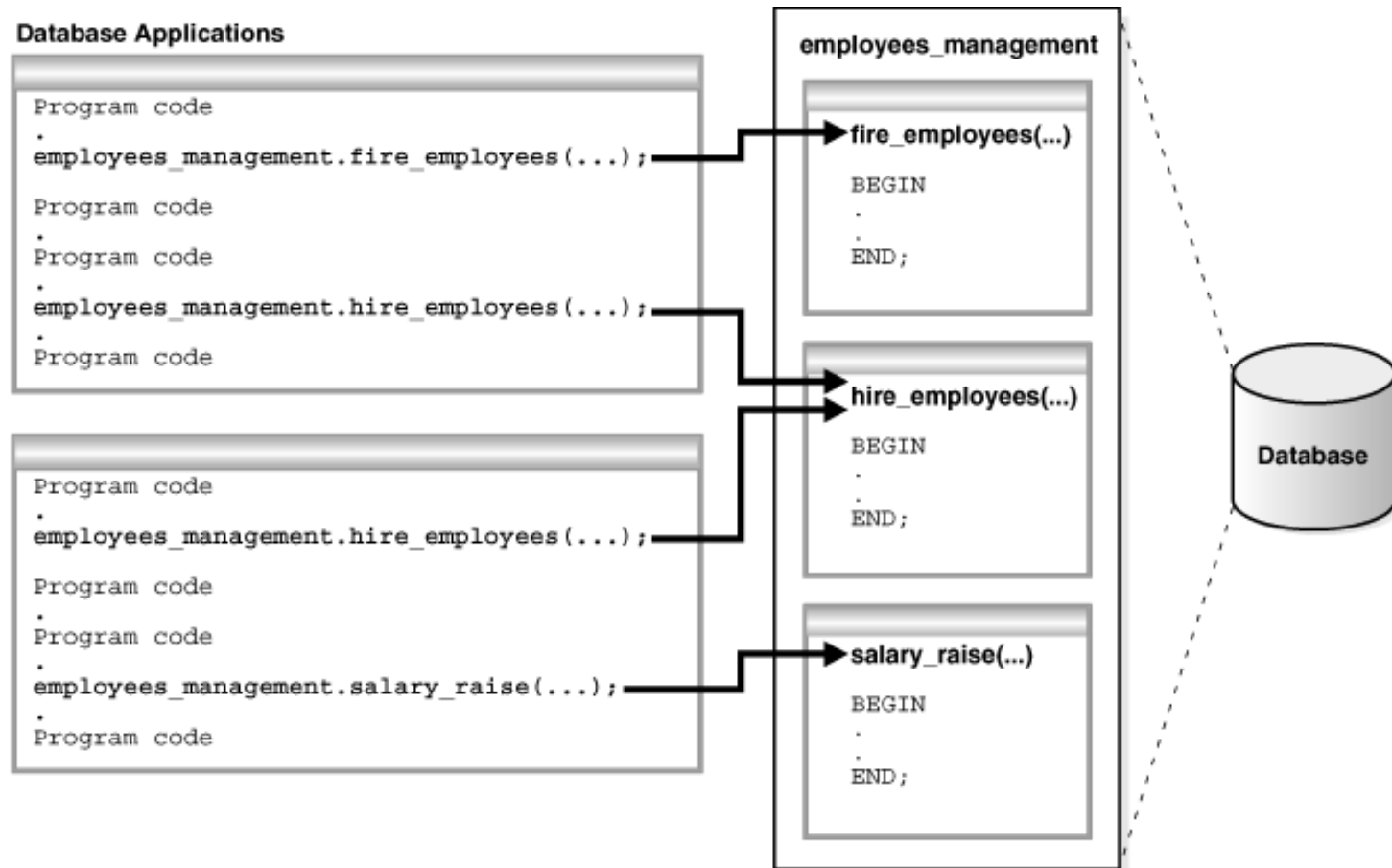
Corps du package

```
    PROCEDURE calc_bonus(date_hired emp.hiredate%TYPE) IS  
    BEGIN  
        DBMS_OUTPUT.PUT_LINE('Employees hired on ' ||  
date_hired || ' get bonus.');
```

```
    END;  
END package_emp;
```

/

# Package



Source : [http://docs.oracle.com/cd/E25178\\_01/server.1111/e25789/srvrside.htm](http://docs.oracle.com/cd/E25178_01/server.1111/e25789/srvrside.htm)

# Déclencheur (Trigger)

---

- Programme PL/SQL stocké dans la base de données et exécuté **automatiquement** en réponse à un événement
  - une opération de mise à jour (INSERT, UPDATE, DELETE)
  - une opération du langage de définition des données (CREATE, ALTER ou DROP)
  - une opération de la base de données (connexion, arrêt, ...)
- Exécuté avant ou après l'événement déclencheur, avant ou après qu'une ligne soit affectée par le trigger
- Forme : Règle Evénement - [Condition] - Action

# Syntaxe TRIGGER

---

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
    [ BEFORE | AFTER ] { INSERT | UPDATE | DELETE }
    ON tbl_name [ FOR EACH ROW ] [ WHEN Condition]
DECLARE
    ...
BEGIN
    ...
END;
/
```

# Syntaxe

---

- BEFORE : le bloc PL/SQL est exécuté AVANT la vérification et la mise à jour des données dans la table
- AFTER : le bloc est exécuté après la mise à jour des données dans la table
- INSERT/UPDATE/DELETE : instruction associée au déclenchement du trigger. Il peut y en avoir plusieurs, elles sont alors séparées par OR

# Syntaxe

---

- FOR EACH ROW : le bloc s'exécute pour chaque ligne traitée par l'instruction associée (c'est un **déclencheur de lignes** : exécuté pour chacune des lignes modifiées, par opposition à un **déclencheur de table** exécuté une seule fois lorsque des modifications interviennent dans une table)



# Les attributs :OLD et :NEW

---

- Permettent de gérer l'ancienne et la nouvelle ligne manipulée
- :OLD non défini pour INSERT
- :NEW non défini pour DELETE

# Exemple TRIGGER

---

```
CREATE OR REPLACE TRIGGER Print_salary_changes
  BEFORE UPDATE ON emp
  FOR EACH ROW WHEN (NEW.EMPNO > 0)

DECLARE

  sal_diff number;

BEGIN
  sal_diff := :NEW.SAL - :OLD.SAL;
  dbms_output.put('Old salary: ' || :OLD.sal);
  dbms_output.put('  New salary: '
|| :NEW.sal);
  dbms_output.put_line('  Difference ' ||
sal_diff);

END;
```

# Trigger sur plusieurs instructions

---

- Lorsqu'un trigger s'applique sur plusieurs types d'instructions (insert/update/delete), possibilité de personnaliser avec les prédicats `inserting`, `updating`, `deleting`  
→ valeur booléenne qui peut être utilisée dans une condition `if (deleting) ...`

# Remarques sur l'utilisation

---

- Dans les triggers BEFORE et FOR EACH ROW (uniquement), possibilité de :
  - Modifier les données qui vont être insérées pour qu'elles respectent les contraintes d'intégrité (impossible dans un trigger AFTER car les contraintes d'intégrité ont déjà été vérifiées, donc plus de modification des données de la ligne)
- Dans un trigger BEFORE INSERT, possibilité de :
  - Faire des requêtes de type SELECT sur la table sur laquelle porte la mise à jour (impossible dans un trigger AFTER car la modification de la ligne n'est pas terminée)

# Modification/Suppression de trigger

---

- ALTER TRIGGER
- DROP TRIGGER

# Utile

---

En cas d'erreurs de compilation `SHOW ERRORS`  
donne des indications sur ces erreurs

# **Extensions du relationnel et autres paradigmes**

# Les bases de données géographiques

---

Extension des bases de données relationnelles :

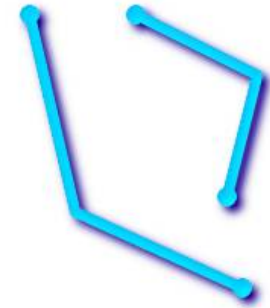
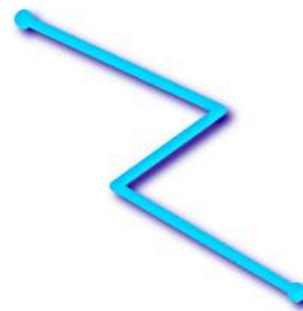
- Ajout d'un type pour stocker des formes géométriques
- Fonctions spatiales
- Index spatiaux



# Géométries

---

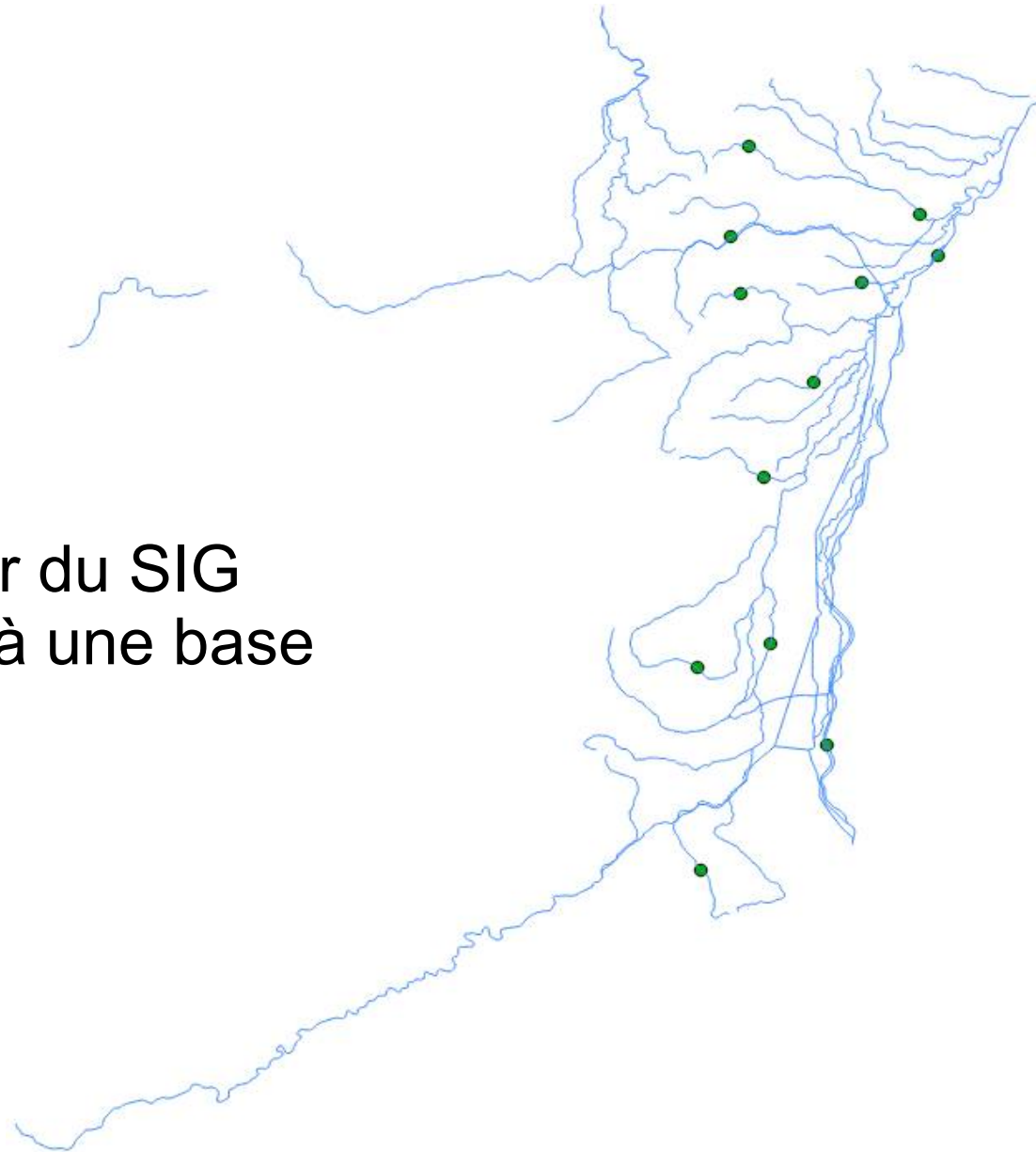
- Points, multipoints
- Lignes, multilignes
- Polygones, multipolygones
- ...



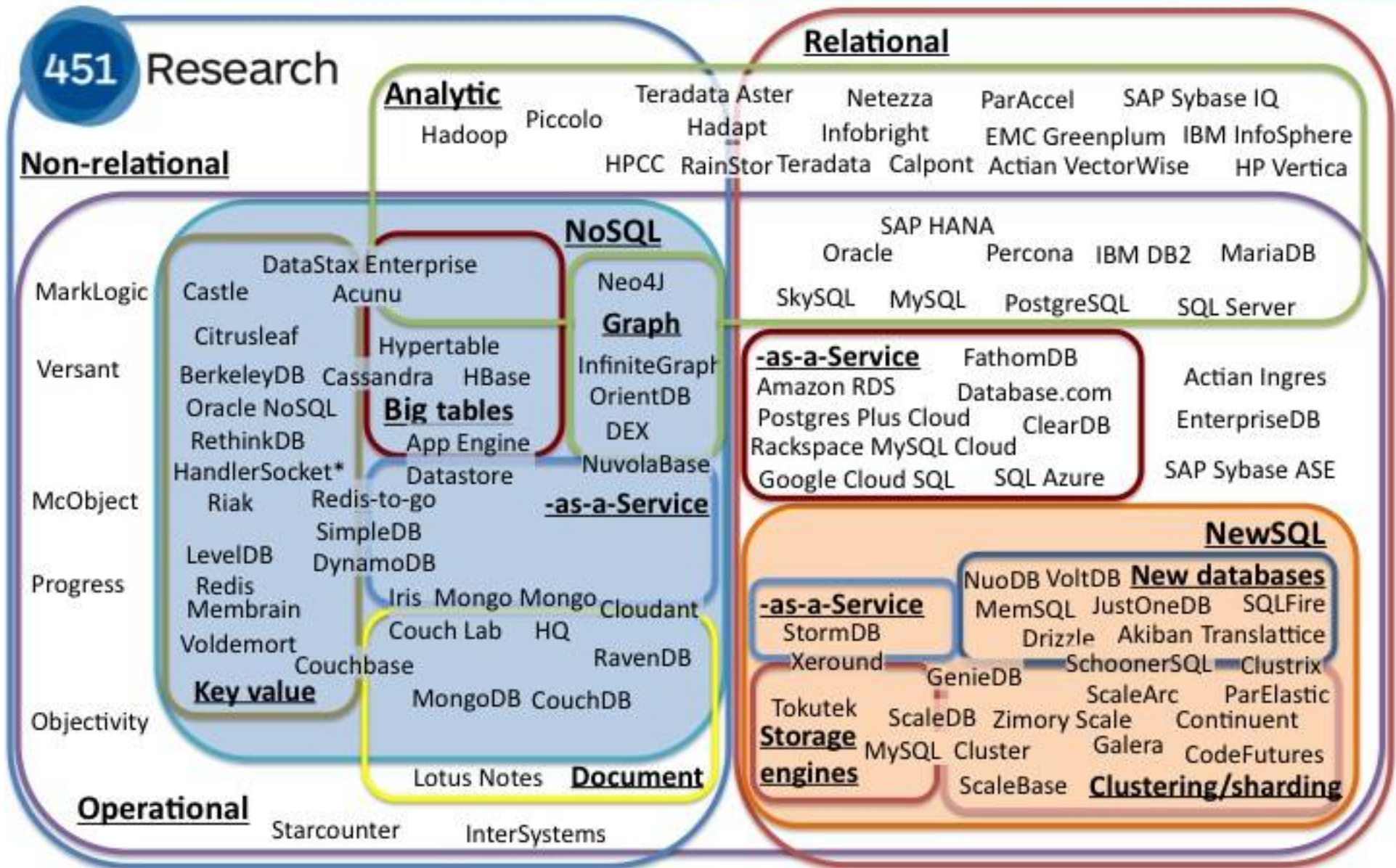
# Exemple données géographiques

---

Affichage à partir du SIG  
QGIS connecté à une base  
Postgis



## The evolving database landscape






# NoSQL

---

- Not Only SQL
- Pallier certaines limites du modèle relationnel
  - modèles de données différents plus adaptés à certains contextes
  - passage à l'échelle
  - performances extrêmes
  - abandon des propriétés ACID pour éviter les surcoûts en latence, accès disque, CPU (verrous, journalisation, ...)

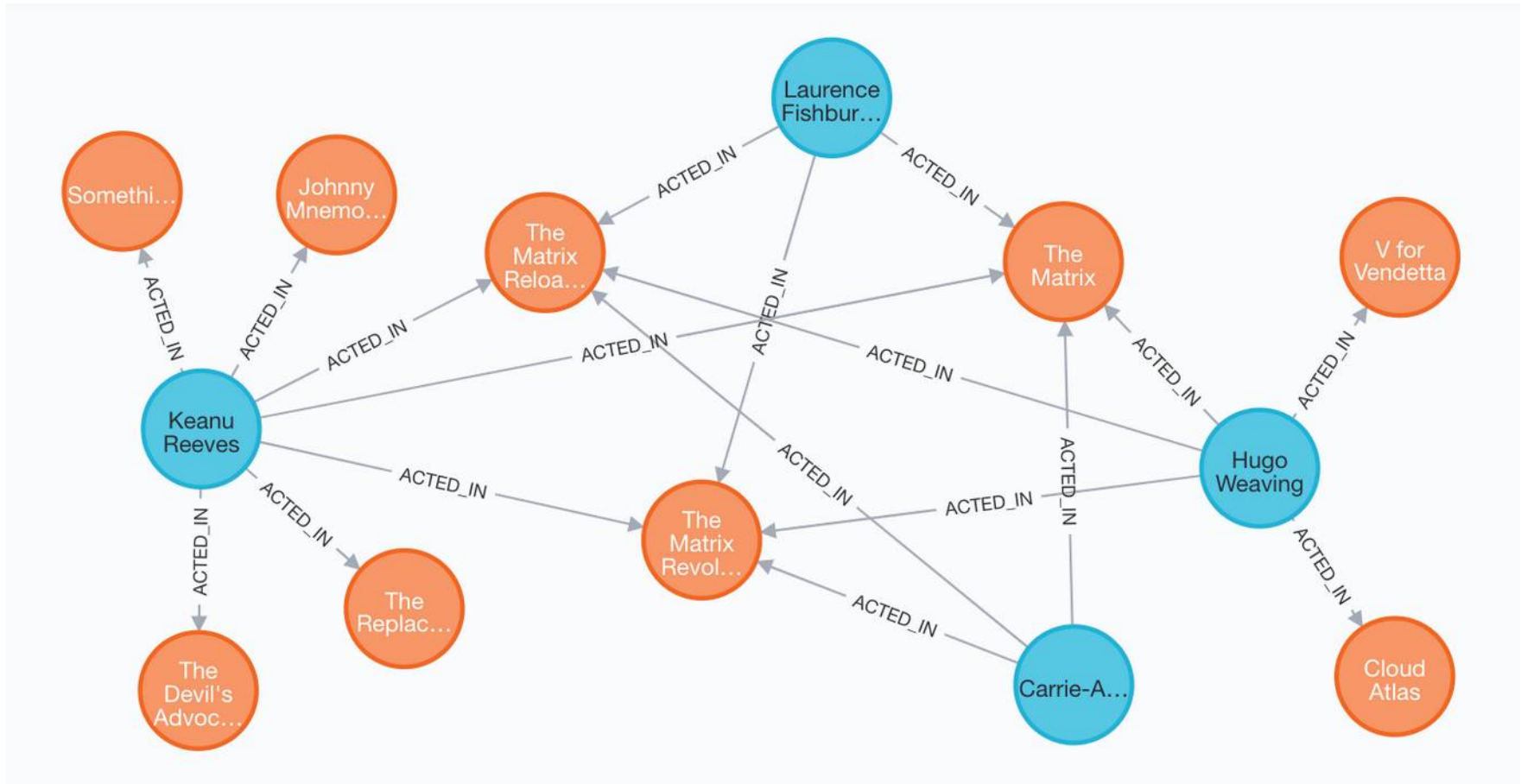
# Différents types de SGBD NoSQL

---

- Graphe  neo4j
- Triplets (web sémantique)
- Clé-valeur
- Orientés document : documents organisés en collections, utilisation d'index, facilité d'utilisation dans un langage de programmation  mongoDB
- Orientés colonnes  cassandra
- ...



# Exemple Neo4j



<https://neo4j.com/developer/example-project/>

# Exemple MongoDB

---

Un document :

```
{
  "_id": "movie:1",
  "title": "Vertigo",
  "year": "1958",
  "director": {
    "_id": "artist:3",
    "last_name": "Hitchcock",
    "first_name": "Alfred",
    "birth_date": "1899"
  },
  "actors": [
    {
      "_id": "artist:15",
      "first_name": "James",
      "last_name": "Stewart",
    },
    {
      "_id": "artist:16",
      "first_name": "Kim",
      "last_name": "Novak",
    }
  ]
}
```

Une requête :

```
db.movies.find ({"director.last_name":
"Hitchcock"})
```

Source : <http://b3d.bdpedia.fr/mongodb.html>

# Un classement des SGBD

415 systems in ranking, October 2023

Rank			DBMS	Database Model	Score		
Oct 2023	Sep 2023	Oct 2022			Oct 2023	Sep 2023	Oct 2022
1.	1.	1.	Oracle +	Relational, Multi-model ⓘ	1261.42	+20.54	+25.05
2.	2.	2.	MySQL +	Relational, Multi-model ⓘ	1133.32	+21.83	-72.06
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model ⓘ	896.88	-5.34	-27.80
4.	4.	4.	PostgreSQL +	Relational, Multi-model ⓘ	638.82	+18.06	+16.10
5.	5.	5.	MongoDB +	Document, Multi-model ⓘ	431.42	-8.00	-54.81
6.	6.	6.	Redis +	Key-value, Multi-model ⓘ	162.96	-0.72	-20.41
7.	7.	7.	Elasticsearch	Search engine, Multi-model ⓘ	137.15	-1.84	-13.92
8.	8.	8.	IBM Db2	Relational, Multi-model ⓘ	134.87	-1.85	-14.79
9.	9.	↑ 10.	SQLite +	Relational	125.14	-4.06	-12.66
10.	10.	↓ 9.	Microsoft Access	Relational	124.31	-4.25	-13.85

<https://db-engines.com/en/ranking>

Classement basé sur

- le nombre de mentions du système sur des sites web (nombre de résultats dans Google, Bing et Yandex)
- l'intérêt général du système (fréquence de recherche du système donnée par Google Trends)
- la fréquence des discussions techniques sur le système (nombre de questions liées et nombre d'utilisateurs intéressés sur Stack Overflow et DBA Stack Exchange)
- nombre d'offres d'emploi dans lesquelles le système est mentionné sur Indeed et Simply Hired
- nombre de profils professionnels où le système est mentionné sur LinkedIn et Upwork
- nombre de tweets où le système est mentionné



# Un classement des SGBD

378 systems in ranking, September 2021

Rank			DBMS	Database Model	Score		
Sep 2021	Aug 2021	Sep 2020			Sep 2021	Aug 2021	Sep 2020
1.	1.	1.	Oracle +	Relational, Multi-model ⓘ	1271.55	+2.29	-97.82
2.	2.	2.	MySQL +	Relational, Multi-model ⓘ	1212.52	-25.69	-51.72
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model ⓘ	970.85	-2.50	-91.91
4.	4.	4.	PostgreSQL + ⓘ	Relational, Multi-model ⓘ	577.50	+0.45	+35.22
5.	5.	5.	MongoDB +	Document, Multi-model ⓘ	496.50	-0.04	+50.02
6.	6.	↑ 7.	Redis +	Key-value, Multi-model ⓘ	171.94	+2.05	+20.08
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model ⓘ	166.56	+1.09	+5.32
8.	8.	8.	Elasticsearch	Search engine, Multi-model ⓘ	160.24	+3.16	+9.74
9.	9.	9.	SQLite +	Relational	128.65	-1.16	+1.98
10.	↑ 11.	10.	Cassandra +	Wide column	118.99	+5.33	-0.18

<https://db-engines.com/en/ranking>

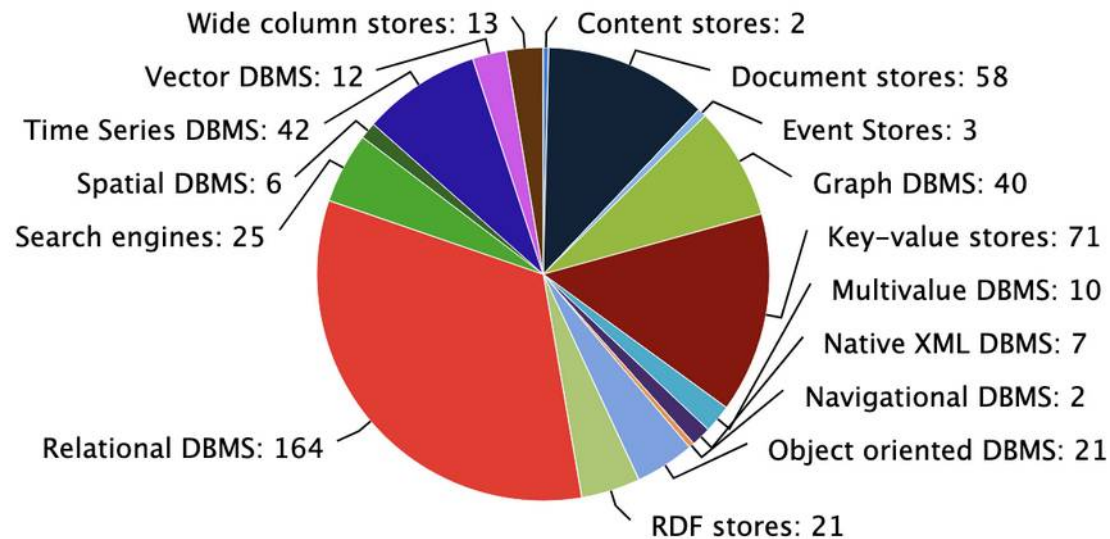
Classement basé sur

- le nombre de mentions du système sur des sites web (nombre de résultats dans Google, Bing et Yandex)
- l'intérêt général du système (fréquence de recherche du système donnée par Google Trends)
- la fréquence des discussions techniques sur le système (nombre de questions liées et nombre d'utilisateurs intéressés sur Stack Overflow et DBA Stack Exchange)
- nombre d'offres d'emploi dans lesquelles le système est mentionné sur Indeed et Simply Hired
- nombre de profils professionnels où le système est mentionné sur LinkedIn et Upwork
- nombre de tweets où le système est mentionné

# Nombre de systèmes par type

## DBMS popularity broken down by database model

Number of systems per category, October 2023

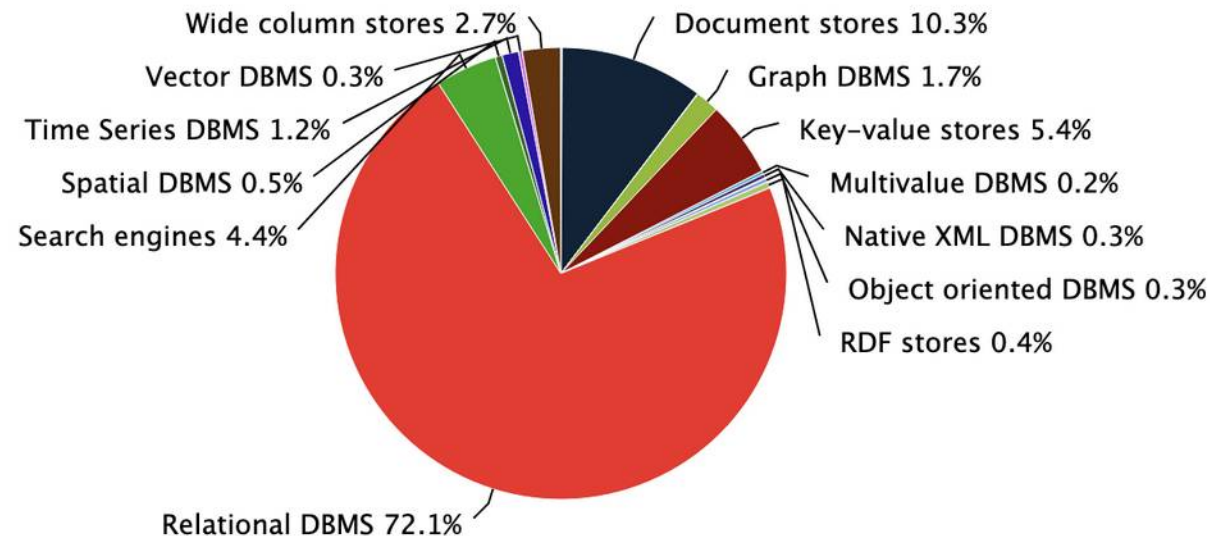


© 2023, DB-Engines.com

[http://db-engines.com/en/ranking\\_categories](http://db-engines.com/en/ranking_categories)

# Popularité en % par type de SGBD

**Ranking scores per category in percent, October 2023**

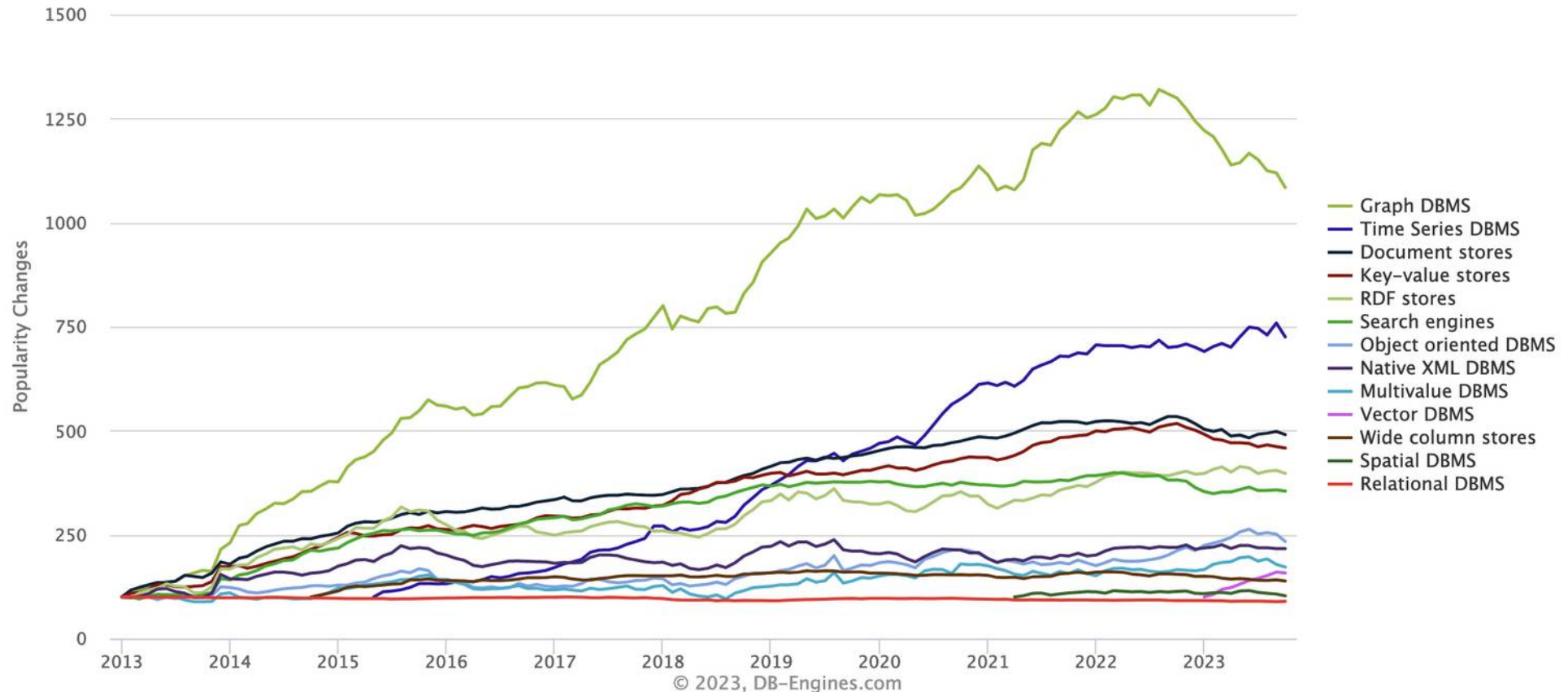


© 2023, DB-Engines.com

[http://db-engines.com/en/ranking\\_categories](http://db-engines.com/en/ranking_categories)

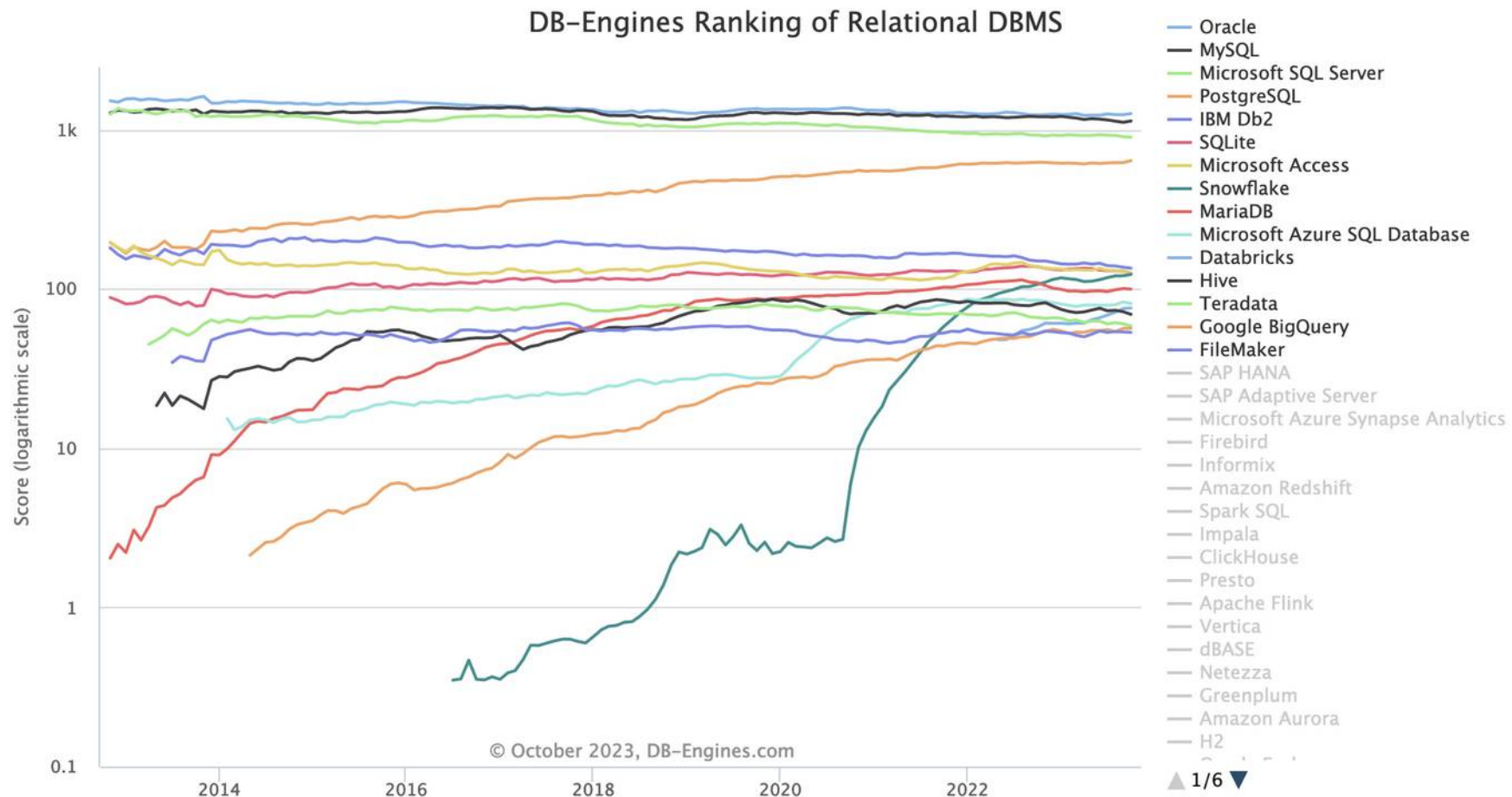
# Evolution de la popularité par type

Complete trend, starting with January 2013



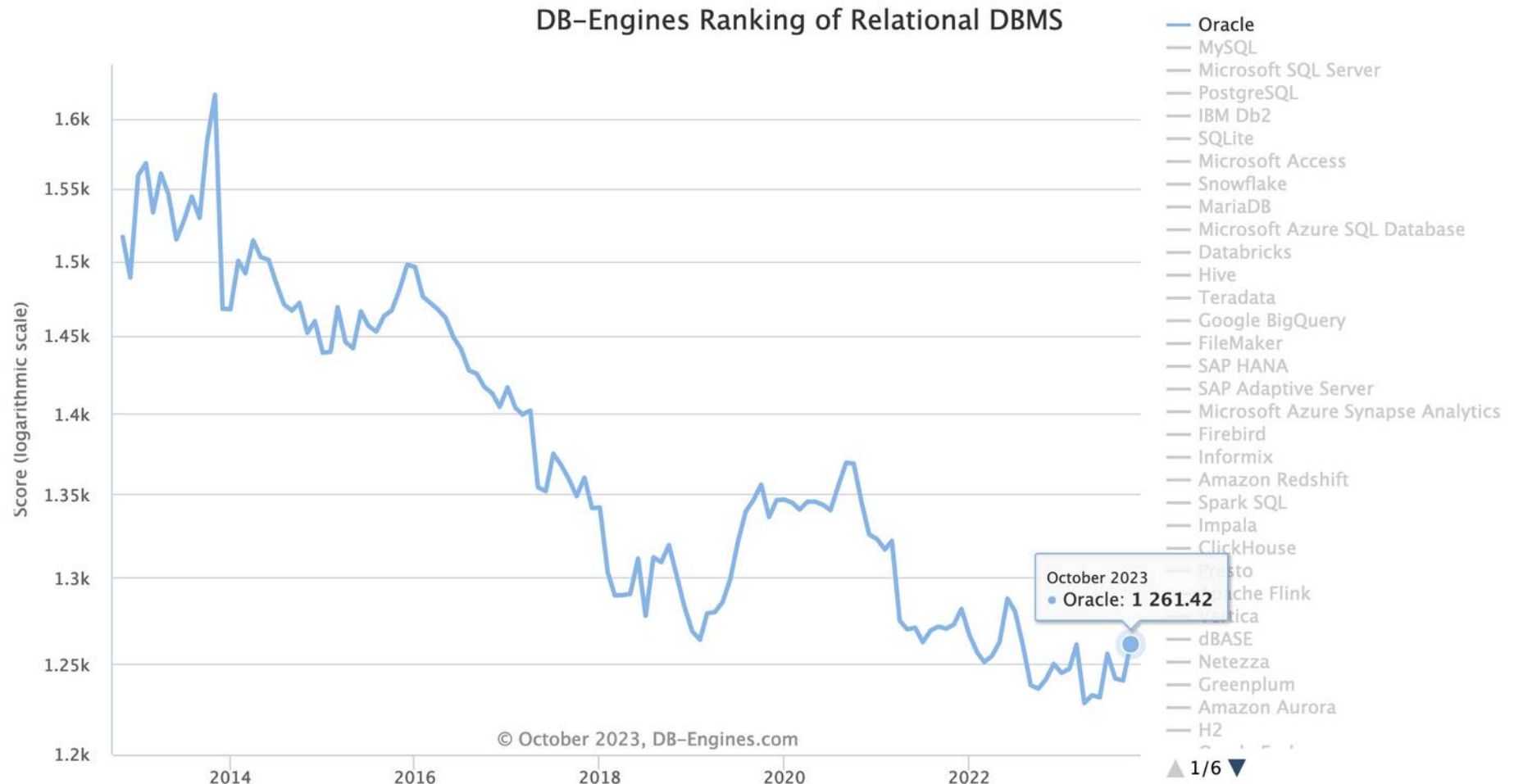
[http://db-engines.com/en/ranking\\_categories](http://db-engines.com/en/ranking_categories)

# Evolution de la popularité des SGBD relationnels



[https://db-engines.com/en/ranking\\_trend/relational+dbms](https://db-engines.com/en/ranking_trend/relational+dbms)

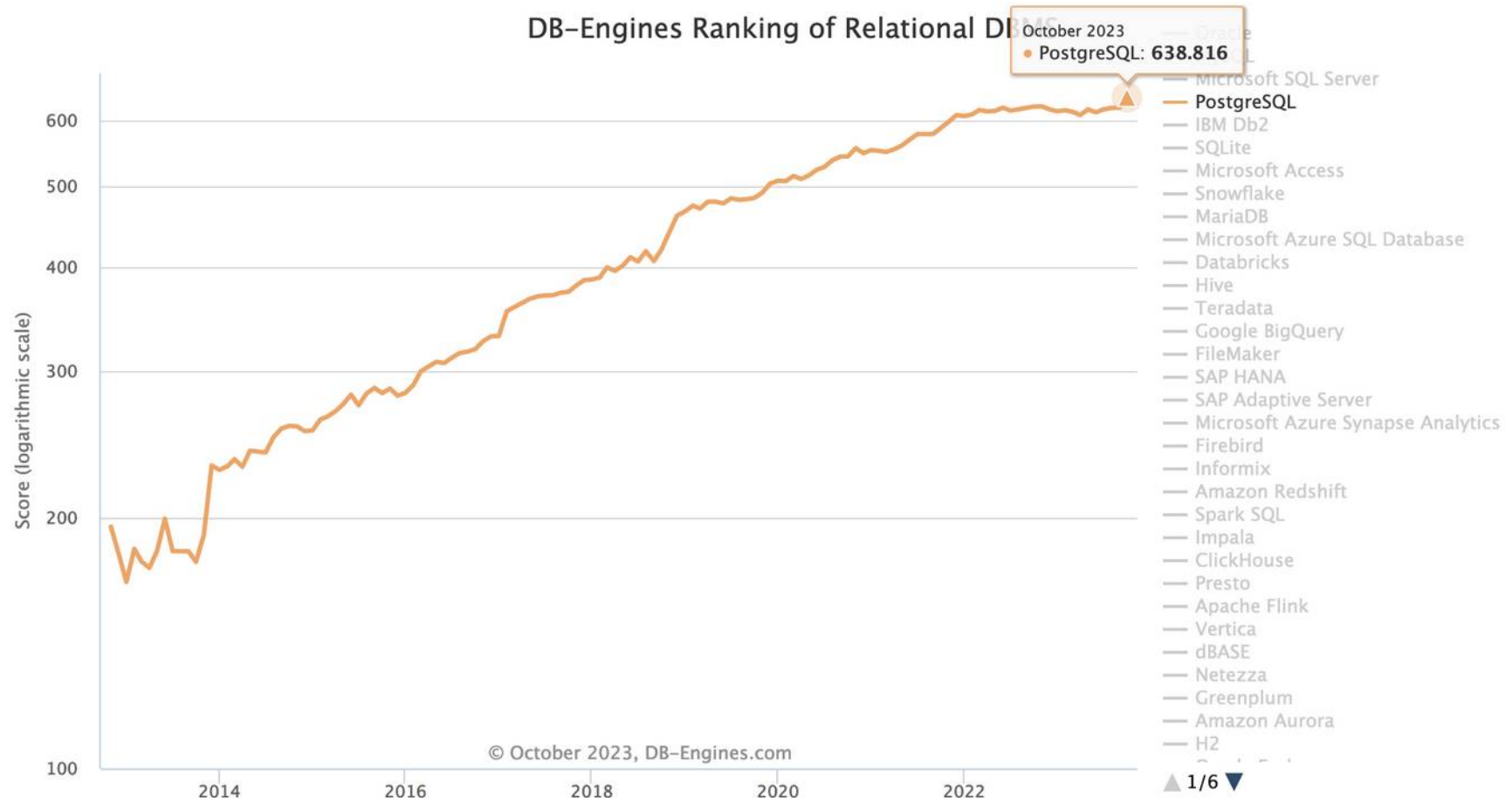
# Evolution de la popularité d'Oracle



[https://db-engines.com/en/ranking\\_trend/system/Oracle](https://db-engines.com/en/ranking_trend/system/Oracle)

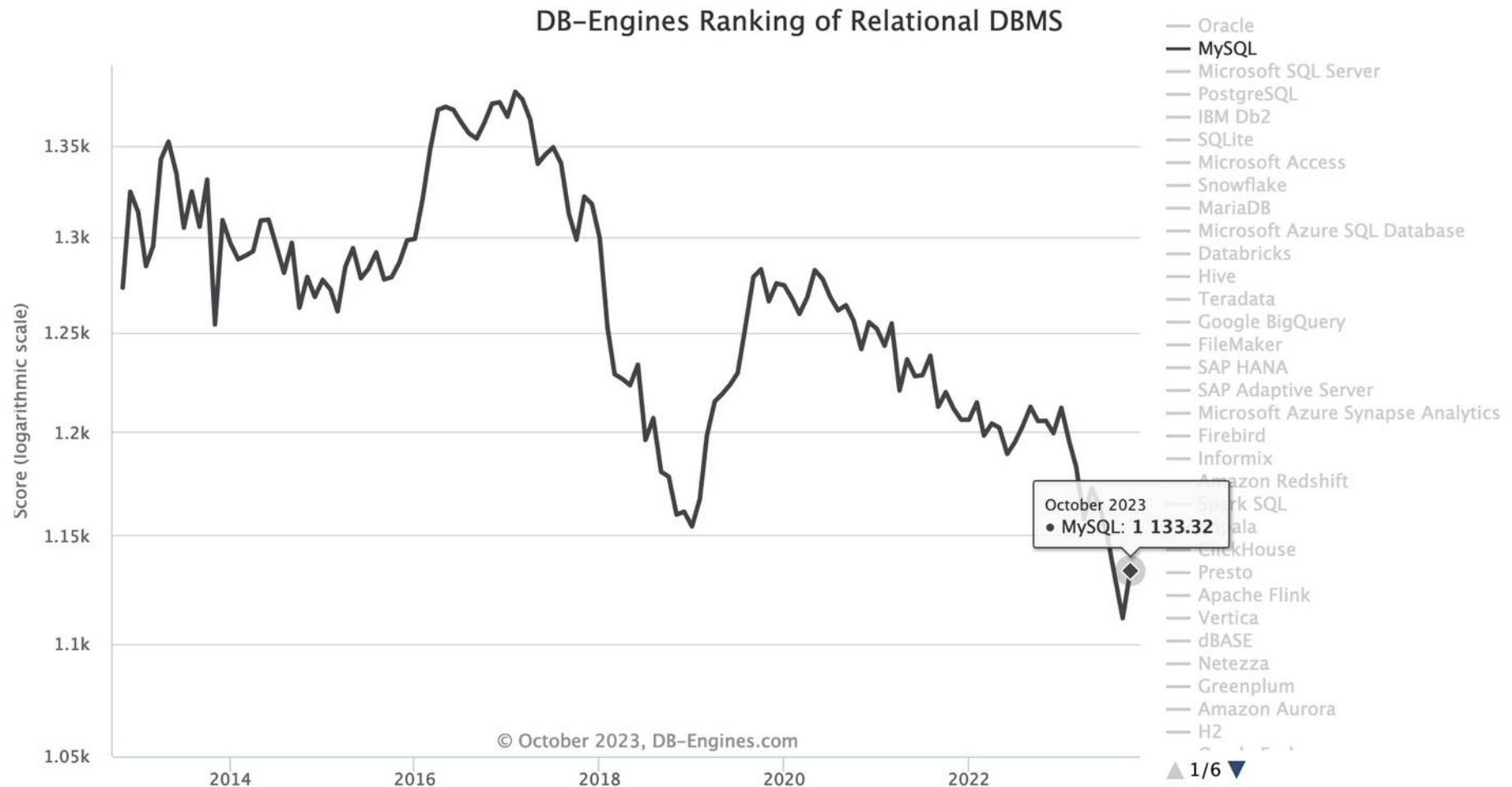


# Evolution de la popularité de Postgres



[https://db-engines.com/en/ranking\\_trend/system/PostgreSQL](https://db-engines.com/en/ranking_trend/system/PostgreSQL)

# Evolution de la popularité de MySQL

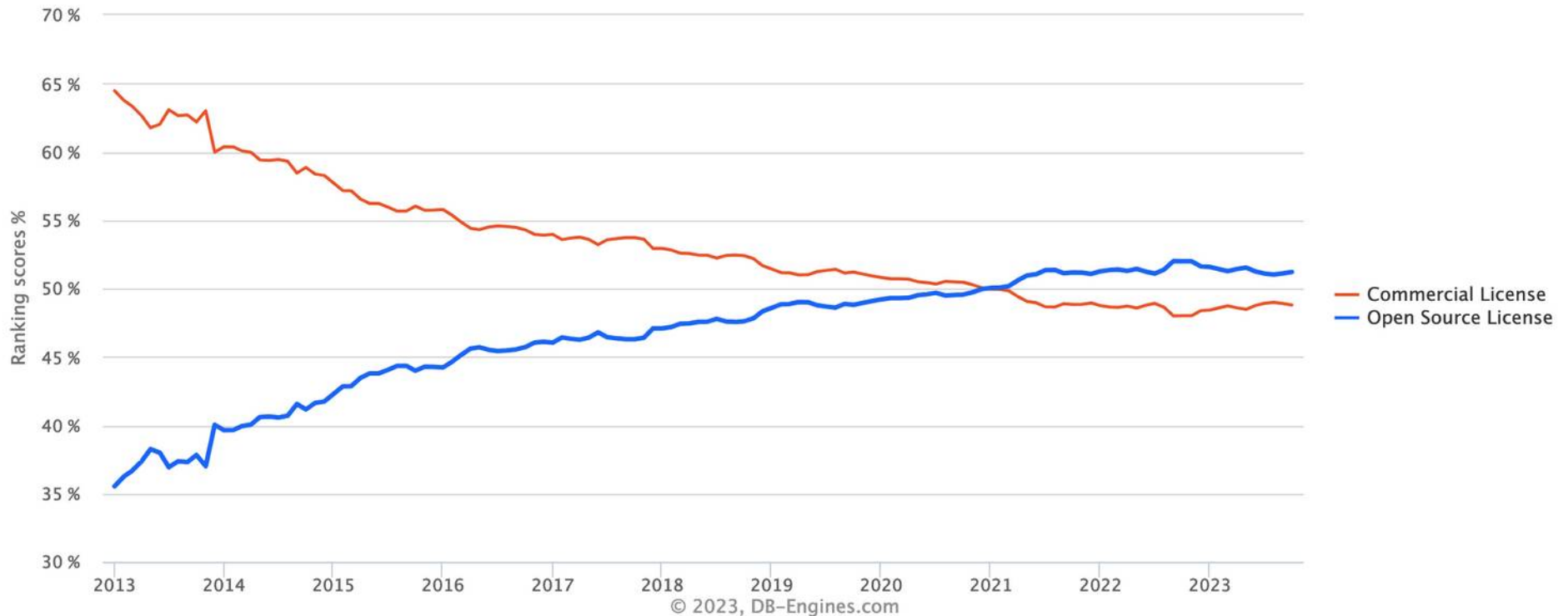


[https://db-engines.com/en/ranking\\_trend/system/MySQL](https://db-engines.com/en/ranking_trend/system/MySQL)



# Popularité SGBD commerciaux/non

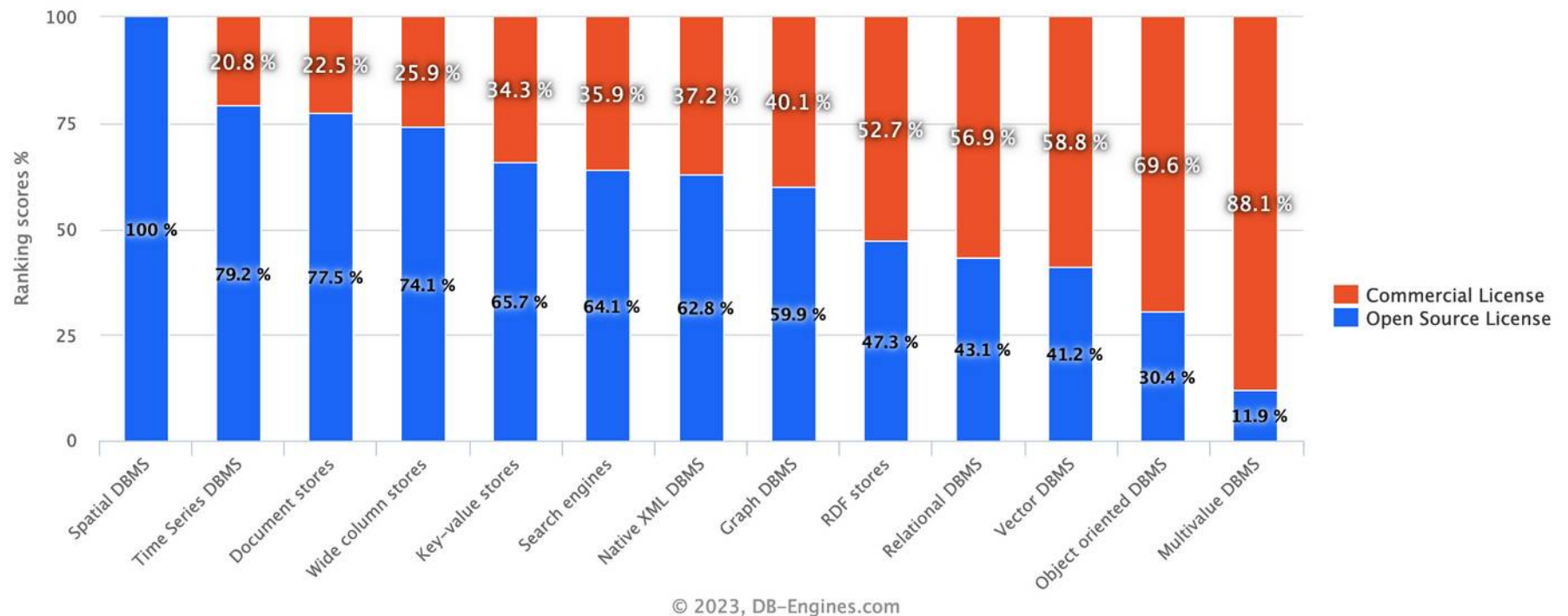
## Popularity trend



[https://db-engines.com/en/ranking\\_osvsc](https://db-engines.com/en/ranking_osvsc)

# Popularité SGBD commerciaux/non par type

Popularity broken down by database model, October 2023



[https://db-engines.com/en/ranking\\_osvsc](https://db-engines.com/en/ranking_osvsc)