Property-based Testing en Bases de Datos

Jesús Manuel Almendros Jiménez.
Departamento de Informática.
Universidad de Almería.

19 de Marzo del 2024

Property-based testing en bases de datos

Resumen

- ¿En qué consiste el Property based Testing?
- Elementos del Property based Testing
- Unit testing versus Property based testing
- PBT en Java
- PBT en Python y Haskell
- PBT en SQL
- PBT en XQuery
- PBT en SPARQL
- Bibliografía

Property-based testing en bases de datos

¿En qué consiste el property based testing?

- Automatizar la prueba de un programa, procedimiento o función
- ¿Qué solemos hacer para probar un programa? Probamos con ejemplos.

  ✓ Por ejemplo, para un algoritmo de ordenación probamos con {12,25,2}, {72,3,15,90}, etc. y vemos si nos el algoritmo nos devuelve {2,12,25}, {3,15,72,90}, etc.

- Si alguno de los ejemplos no da el resultado esperado revisamos el algoritmo.
- Puede ocurrir que una gran mayoría de ejemplos funcione pero solo algunos den problemas. ¿Por qué no automatizarlo?
- ¿Qué indicios podemos encontrar de que un algoritmo no es correcto?

Property-based testing en bases de datos

Elementos del Property based Testing

- Generador automático de casos de prueba.
- Casos de prueba con datos aleatorios (y diferente tamaño).
- Se pueden configurar de modo que se adapten al programa a evaluar.
- Que cubran el mayor número de casos posible.
- Propiedad a comprobar sobre el resultado.

✓ Se puede a veces especificar una propiedad sobre la entrada, con el objeto de impedir falsos positivos.

Property-based testing en bases de datos

Elementos del Property based Testing

- La propiedad no tiene por qué ser la postcondición

  ✓ Basta con una propiedad simple necesaria que debe cumplir el resultado del programa.

- En caso de haber algún caso de prueba que no cumpla la propiedad (normalmente llamado contraejemplo), la herramienta de PBT lo mostrará al usuario.

- En algunas herramientas existe la posibilidad de hacer shrink, que significa que una vez obtenido un contraejemplo, se obtenga uno más sencillo a partir de el contraejemplo.

  ✓ Supongamos que nuestro contraejemplo para el algoritmo de ordenación es {20,2,15,72}. En tal caso, se puede intentar reducir el contraejemplo: {20,2,15}, {20,2}, {20}.

Property-based testing en bases de datos

Unit testing versus Property based testing

Unit Testing

- Se centran en probar unidades individuales de código, como funciones o métodos, de manera aislada. Las pruebas unitarias suelen estar diseñadas para validar casos específicos y predefinidos del comportamiento de una función o método.
- Requieren que el programador defina casos de prueba específicos, proporcionando datos de entrada y resultados conocidos.

Property-based Testing

- Se centra en probar propiedades generales del código en lugar de casos específicos. En lugar de proporcionar datos de entrada específicos, se definen propiedades que deben mantenerse independientemente de los datos de entrada.
- Utiliza generación automática de datos de entrada, a menudo de manera aleatoria, para explorar un amplio rango de posibles casos.

Property-based testing en bases de datos

¿En qué consiste el property based testing?

- Supongamos que tenemos un programa Java que calcula el máximo común divisor m de dos números x e y.
- Sabemos que:
  - m divide a x; m divide a y; ningún otro número k más grande que m divide a x e y.
  - m tiene que ser necesariamente menor o igual que x e y.
  - Si x=y entonces m=x=y.
  - mcd(x,y)=mcd(y,x)
  - Etc..
- Estas son propiedades necesarias que debe cumplir m para poder ser el máximo común divisor

Property-based testing en bases de datos

Construyamos nuestro propio PBT para números

```
for (int x = 1; x <= 50; x++) {

    for (int y = 1; y <= 50; y++) {

        int m = calcularMCD(x, y);

            if (m > x) {

            System.out.println("Failed test in x: " + x); }

            if (m > y) {

            System.out.println("Failed test in y: " + y); }}}
```

Generador de test (casos de prueba)

Propiedad

Propiedad

Mi propio PBT!

Property-based testing en bases de datos

Construyamos nuestro propio PBT para números

```
public int calcularMCD(int a, int b) {

while (b != 0) {

    int temp = b;

    // bug: should be b = a % b;

    b = b % a;

    a = temp;

}

return a;}
```

Failed test in x: 1

Failed test in x: 1

Failed test in x: 1

Failed test in x: 1

Failed test in x: 1

Failed test in x: 1

Failed test in x: 1

Failed test in x: 2

Failed test in x: 2

Failed test in x: 2

Failed test in x: 2

Property-based testing en bases de datos

¿En qué consiste el property based testing?

- Supongamos que tenemos un programa Java que ordena un vector
- Sabemos que:
  - El primer elemento del resultado debe ser menor que el segundo (o cualesquiera dos en posiciones consecutivas)
  - La suma de los elementos del vector es la misma ordenado o no ordenado
  - El mínimo del vector aparece en la primera posición cuando está ordenado
  - Si aplicamos dos veces la ordenación el resultado es el mismo
- Estas son propiedades necesarias que debe cumplir el resultado para poder ser una ordenación válida

Property-based testing en bases de datos

Construyamos nuestro propio PBT para listas

```
int size_vectors = 5;
int number_of_test = 10;
Random random = new Random();
for (int size = 2; size <= size_vectors; size++){
        int[] vector = new int[size];
            for (int test = 1; test <= number_of_test; test++){
                for (int j = 0; j < size; j++){
                    vector[j] = random.nextInt();
                }
        ordenarBurbuja(vector);
        if (vector[0] > vector[1]){
                    System.out.println("Failed Test: " + test + " with size: " + size);
                    imprimir(vector);
        }
```

Generador de test (casos de prueba)

Propiedad

Mi propio PBT!

Property-based testing en bases de datos

Construyamos nuestro propio PBT para listas

```
public void ordenarBurbuja(int[] entrada){
int n = entrada.length;
for (int i = 0; i < n - 1; i++){
 for (int j = 0; j < n - i - 1; j++){
   if (entrada[j] >= entrada[j + 1]){
          int temp = entrada[j];
          entrada[j] = entrada[j + 1];
          // bug: j should be j+1
          entrada[j] = temp;
     }}}}
```

Failed Test: 1 with size: 2

119572626

-1627845081

Failed Test: 5 with size: 2

1579250802

-493931991

Failed Test: 6 with size: 2

-506612227

-1293472414

Failed Test: 8 with size: 2

-824130431

-845353468

Property-based testing en bases de datos

Property-Based Testing with jqwik (Java)

Especificación de los casos de prueba

```
@Property(tries = 5000, shrinking =
ShrinkingMode.OFF, generation =
GenerationMode.RANDOMIZED)
```

Salida del PBT

```
AssertionFailedError: Property
[ListReverseProperties:reverseKeepsTheOri
ginalList] falsified with sample [[0, -1]]

tries = 2                | # of calls to property
checks = 2                 | # of not rejected calls
sample = [[0, -1]]
```

Especificación de propiedades

```
@Property
fun `reversing keeps all elements`(@ForAll list: List<Int>){
    assertThat(list.reversed()).containsAll(list)
}
@Property
fun `reversing twice results in original list`(@ForAll list: List<Int>){
    assertThat(list.reversed().reversed()).isEqualTo(list)
}
@Property
fun `reversing swaps first and last`(@ForAll @Size(min=2) list: List<Int>){
    val reversed = list.reversed()
    assertThat(reversed[0]).isEqualTo(list[list.size - 1])
    assertThat(reversed[list.size - 1]).isEqualTo(list[0])
}
```

Property-based testing en bases de datos

Property-Based Testing with Hypothesis (Python)

Especificación de casos de prueba y propiedades

```
@given(s=text())
@example(s="")
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

```
@given(st.lists(st.integers()))
def test_reversing_twice_gives_same_list(xs):
    ys = list(xs)
    ys.reverse()
    ys.reverse()
    assert xs == ys
```

Salida del PBT

```
Falsifying example: test_decode_inverts_encode(s='001')
```

Property-based testing en bases de datos

Property-Based Testing with QuickCheck (Haskell)

Especificación de la propiedad

```
import Test.QuickCheck

prop_reverse :: [Int] -> Bool

prop_reverse xs = reverse (reverse xs) == xs
```

Salida del PBT

```
>>> quickCheck prop_reverse

*** Failed! Falsified (after 1 test):

[1,2,3] /= [3,2,1]
```

Generación de casos de prueba

```
arbitraryList :: Arbitrary a => Gen [a]

arbitraryList =

 sized $

  \n -> do

   k <- choose (0, n)

   sequence [ arbitrary | _ <- [1..k] ]
```

Property-based testing en bases de datos

Property-Based Testing en SQL

“Empleados de una empresa cuyo sueldo es más del doble de todos los empleados con la misma edad y categoría profesional”

```
 SELECT e.nombre, e.edad, e.salario, e.categoria_profesional
FROM empleados e
WHERE NOT EXISTS (
    SELECT e2
    FROM empleados e2
    WHERE e.edad = e2.edad
    AND e.categoria_profesional = e2.categoria_profesional
    AND e.salario <= e2.salario * 2
    AND e.nombre != e2.nombre
);
```

Property-based testing en bases de datos

Property-Based Testing en SQL

¿Qué sabemos sobre el resultado de la consulta?

- ¿El resultado puede ser vacío si la tabla original no es vacía?
- ¿Qué relación hay entre el resultado y la tabla original?
- ¿Puede haber elementos repetidos en la tabla resultado?
- ¿Qué ocurre si ordenamos o agrupamos el resultado?
- ¿Qué ocurre si aplicamos la misma consulta sobre el resultado?

Property-based testing en bases de datos

Property-Based Testing en SQL

¿Cómo generamos casos de prueba?

- Necesitamos generar nombres para los empleados, edades, salarios y categorías
- Necesitamos generar tablas de distinto tamaño
- ¿Necesitamos generar algún dato que esté repetido?

Nombre	Edad	Salario	Categoría
N1	30	100	C1
N2	50	200	C2
N3	60	100	C3
N4	30	250	C1

Nombre	Edad	Salario	Categoría
N1	30	100	C1
N2	30	500	C2
N3	60	100	C3
N4	30	250	C1

Nombre	Edad	Salario	Categoría
N1	30	100	C1

Nombre	Edad	Salario	Categoría
N1	30	100	C1
N2	50	200	C2

Property-based testing en bases de datos

Property-Based Testing en SQL

Procedimiento Almacenado para generar filas de una tabla

```
CREATE PROCEDURE InsertarEmpleadoAleatorio(
    @nombre VARCHAR(100),
    @edad INT,
    @salario DECIMAL(10, 2),
    @categoria_profesional VARCHAR(100)
)
AS
BEGIN
    DECLARE @id_empleado INT;
    SELECT @id_empleado = MAX(id) + 1 FROM empleados;
     INSERT INTO empleados (id, nombre, edad, salario, categoria_profesional).
                    VALUES (@id_empleado, @nombre, @edad, @salario, @categoria_profesional);
 END;
```

Property-based testing en bases de datos

Property-Based Testing en SQL

Generador aleatorio de tablas de hasta 5 filas

```
CREATE PROCEDURE InsertarEmpleadosAleatoriosBucle AS
BEGIN
    DECLARE @contador INT;
    SET @contador = 1;
    WHILE @contador <= 50
    BEGIN
        DECLARE @nombre VARCHAR(100);  DECLARE @edad INT; DECLARE @salario DECIMAL(10, 2); DECLARE
@categoria_profesional VARCHAR(100);
      SET @nombre = 'Empleado' + CAST((RAND() * 1000) AS VARCHAR(10));
       SET @edad = CAST((RAND() * 41 + 20) AS INT);
       SET @salario = CAST((RAND() * 80001 + 20000) AS DECIMAL(10, 2));
       SET @categoria_profesional = 'Categoria' + CAST((RAND() * 10 + 1) AS VARCHAR(10));
       EXEC InsertarEmpleadoAleatorio @nombre, @edad, @salario, @categoria_profesional;
       SET @contador = @contador + 1;
    END
END;
```

Property-based testing en bases de datos

Property-Based Testing en SQL

Propiedad: No hay duplicados en el resultado

```
CREATE PROCEDURE ComprobarEmpleadosDuplicadosPorEdadYCategoria
AS
BEGIN
    IF EXISTS (
        SELECT e1.edad, e1.categoria_profesional
        FROM empleados e1
        INNER JOIN empleados e2 ON e1.edad = e2.edad AND e1.categoria_profesional = e2.categoria_profesional
        WHERE e1.nombre <> e2.nombre
    )
    BEGIN
        PRINT 'Failed test';
    END
END;
```

Property-based testing en bases de datos

Property-Based Testing en XQuery

XQuery:

- Lenguaje de consulta de XML
- XML es una estructura arbórea
- XML maneja datos semi-estructurados: un registro de la base de datos puede contener un número variable de atributos. Además los registros pueden estár anidados.
- XQuery maneja for-let-where-order by-group by
- XQuery es un lenguaje funcional. Todo es una expresión.
- Una consulta es una expresión XQuery que puede ser una función Booleana, por tanto, una propiedad

```
<bib>
 <book year="1995">
  <author>Buneman</author>
  <title>UML</title>
  <price>80</price>
 </book>
<book year="1993">
  <author>Abiteboul</author>
  <title>XML</title>
  <price>100</price>
 </book>
</bib>
```

Property-based testing en bases de datos

Property-Based Testing en XQuery

Generador de casos de prueba

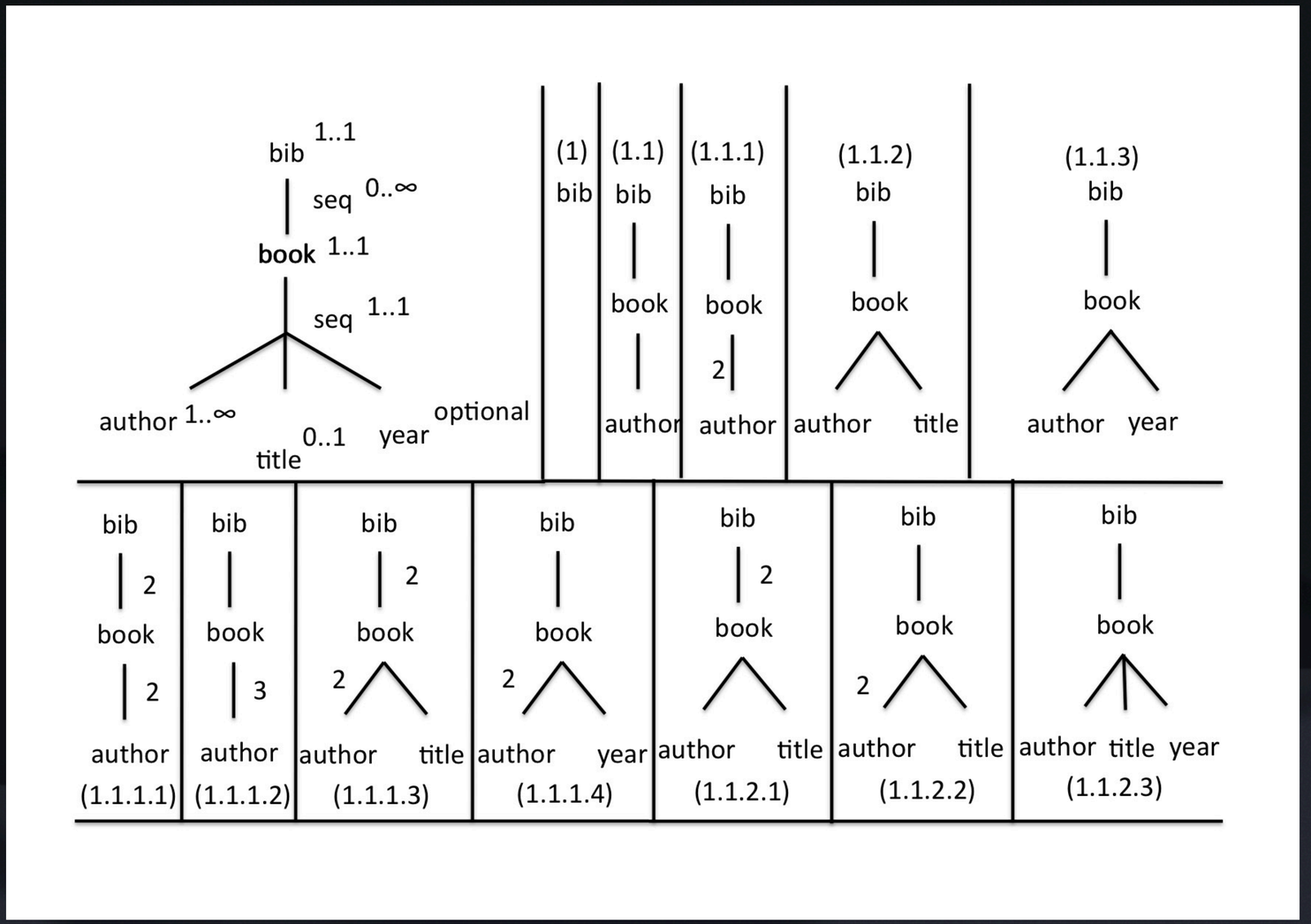
Generación automática de árboles

Combinaciones de etiquetas

Número de hijos variable

A partir de XML Schema

Anotado con valores



Property-based testing en bases de datos

Property-Based Testing en XQuery

Consulta XQuery

```
<bib>
 {
  for $b in $bib//books
  where $b[author="Buneman"]
  return
     <book>
       { $b/title }
       { for $a in $b/author[position()<=2]
          return $a}
       {if (count($b/author) > 2)
          then <et-al/>
          else ()}
     </book>
  }
</bib>
```

Especificación de la propiedad

```
every $b in $bib//book
 satisfies $b/author="Buneman"
```

Salida del PBT

```
Falsifiable after 216 tests.
Counterexamples:
<bib>
 <book year="1995">
  <author>Abiteboul</author>
  <author>Abiteboul</author>
  <author>Buneman</author>
  <title>UML</title>
  <price>80</price>
 </book>
</bib>
```

Property-based testing en bases de datos

Property-Based Testing en XQuery

SPARQL:

- Lenguaje de Consulta de RDF
- RDF es un un grafo
- SPARQL es similar a SQL. Maneja patrones que han de ajustarse al grafo.
- SPARQL incluye WHERE, FILTER, UNION, EXISTS, NOT EXISTS, Agregación.
- OWL es un lenguaje de ontologías extensión de RDF que permite definir propiedades

```
<owl:NamedIndividual rdf:about=“#antonio">
<age rdf:datatype="&xsd;integer">40</age>
 <name rdf:datatype="&xsd;string">antonio</name>
 </owl:NamedIndividual>
```

Property-based testing en bases de datos

Property-Based Testing en SPARQL

Consulta XQuery

```
SELECT ?person
WHERE {
?person sn:name  ?name1 ;
                sn:age  ?age1 .
sn:antonio sn:name  ?name2 ;
                sn:age  ?age2 .
FILTER (?age1 > ?age2 && ?name1 != ?name2)
}
</bib>
```

Generación de casos de prueba

Grafo RDF con combinaciones de datos.

Modificación del hecho para XQuery

Especificación de la propiedad

?person: Young

Where Young = age some integer[< 40]

Salida del PBT

```
Falsifiable after 18 tests.
 ---------------
 Counterexample:
 ---------------
<rdf:RDF>
  <:Person rdf:about="#luis">
    <:age rdf:datatype="#integer">50</:age>
    <:name>luis</:name>
  </:User>
</rdf:RDF>
```

Property-based testing en bases de datos

Bibliografía

- PBT en JAVA: https://jqwik.net/
- PBT en Python: https://hypothesis.readthedocs.io/en/latest/
- PBT en Haskell: https://github.com/nick8325/quickcheck
- PBT en Scala https://scalacheck.org/
- PBT en Elixir y Erlang https://github.com/pragdave/quixir
- PBT en .NET https://github.com/fscheck/FsCheck
- PBT en JavaScript/TypeScript https://github.com/dubzzz/fast-check

Property-based testing en bases de datos

Bibliografía

- Claessen, K., & Hughes, J. (2000, September). QuickCheck: a lightweight tool for random testing of Haskell programs. In Proceedings of the fifth ACM SIGPLAN international conference on Functional programming (pp. 268-279).
- De Angelis, E., Fioravanti, F., Palacios, A., Pettorossi, A., & Proietti, M. (2019). Property-based test case generators for free. In Tests and Proofs: 13th International Conference, TAP 2019, Held as Part of the Third World Congress on Formal Methods 2019, Porto, Portugal, October 9–11, 2019, Proceedings 13 (pp. 186-206). Springer International Publishing.
- Goldstein, H., Cutler, J. W., Dickstein, D., Pierce, B. C., & Head, A. (2024, March). Property-Based Testing in Practice. In 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE) (pp. 971-971). IEEE Computer Society.
- Hebert, F. (2019). Property-Based Testing with PropEr, Erlang, and Elixir: Find Bugs Before Your Users Do. Pragmatic Bookshelf.
- MacIver, D. R., & Hatfield-Dodds, Z. (2019). Hypothesis: A new approach to property-based testing. Journal of Open Source Software, 4(43), 1891.
- Almendros-Jiménez, J. M., & Becerra-Terón, A. (2017). Automatic property-based testing and path validation of XQuery programs. Software Testing, Verification and Reliability, 27(1-2), e1625.
- Almendros-Jiménez, J. M., & Becerra-Terón, A. (2017, September). Property-based testing of SPARQL queries. In Proceedings of the 16th International Symposium on Database Programming Languages (pp. 1-11).