Universidad Complutense de Madrid
Facultad de Informática

Proyecto de Grado de Ingeniería del Software

Generación automática de reglas de cobertura para consultas SQL

Victor Manuel Alonso Rodríguez
Jaime B. Rodríguez García
Mª Pilar Grande Ayuso
Directora : Yolanda García Ruiz

Septiembre 2013

Agradecimientos

Queremos agradecer a todos los que han hecho posible este proyecto, tanto a las personas que han estado directamente relacionado con él, como a las personas que no estando directamente relacionado han sufrido nuestros agobios, malos ratos y alegrías.

A todos.

¡GRACIAS!.

Autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

Abstract

Testing represents an essential step of the software development process. The main objective of Software testing is finding potential execution errors as well as elements that do not meet the required specifications.

The testing process is driven by a set of test cases. Therefore, a preliminary design of such test cases is required.

High quality test cases are very important in the field of databases, specially for SQL language queries. These test cases must be small in size and capable of detecting a large number of errors. Their quality is assessed according to some established criteria. For instance, in the case of queries written in a SQL language, the SQLfpc (Full Predicate Coverage for SQL) criterion is generally employed. This criterion is defined by a set of rules derived from the semantics of the query. In this project, we have developed a prototype capable of automatically generating the set of rules that define the SQLfpc yardstick for SQL queries.

Keywords

Criteria coverage, Test case, Coverage rules, SQLfcp, Clauses, Database

Resumen

Una parte fundamental del desarrollo del software es la fase del testing, también conocida como fase de pruebas del software (en inglés *Software Testing*). La fase de testing del software tiene como objetivo principal, detectar posibles errores en el funcionamiento de las aplicaciones así como elementos que no cumplan con las especificaciones requeridas.

Durante el proceso de pruebas es necesario contar con un conjunto de casos de prueba, por lo que resulta necesario realizar un diseño previo de dichos casos de prueba para su posterior ejecución. En el campo de las bases de datos y en particular en el caso de las pruebas de consultas escritas en lenguaje SQL, es muy importante disponer de un conjunto de casos de prueba que sean de calidad, en el sentido de que sean de pequeño tamaño y que permitan detectar el mayor número de errores. Para medir la calidad del conjunto de casos de prueba se usan ciertos criterios de medida. En particular, para el caso de consultas escritas en lenguaje SQL, se usa el criterio de medida SQLfpc (*Full Predicate Coverage*). Este criterio esta definido mediante un conjunto de reglas que se derivan de la semántica de la consulta. En este proyecto hemos desarrollado un prototipo para la generación automática de las reglas que definen el criterio de medida SQLfpc para consultas SQL.

Palabras clave

Criterios de cobertura, Casos de prueba, Reglas de cobertura, SQLfcp, Bases de datos

Índice general

Capítulo 1

Introducción

Una parte fundamental del desarrollo del software es la fase del testing o también conocida como fase de pruebas del software. La fase de pruebas tiene como objetivo principal detectar posibles errores en el funcionamiento de las aplicaciones, así como elementos que no cumplan con las especificaciones requeridas. En general, para desarrollar el proceso de pruebas, es necesario contar con un conjunto de casos de prueba, por lo que resulta necesario, realizar un diseño previo de dichos casos de prueba, para su posterior ejecución. Los resultados producidos por las aplicaciones sometidas a estas pruebas, son comparadas con los resultados esperados. Cuando existe una discrepancia entre ambos, podemos afirmar que existe un error en la aplicación, por lo que es necesario dar comienzo a las tareas de depuración.

En el caso de las bases de datos relacionales, la fase de pruebas requiere el previo diseño de casos de prueba, es decir, de instancias válidas y de tamaño reducido para su posterior ejecución. Sin embargo este diseño no es trivial, debido a la gran cantidad de factores que han de tenerse en cuenta, para que los casos de prueba sirvan para detectar errores. Para garantizar la calidad de las pruebas, es necesario que el conjunto de los casos de prueba generados, cumplan una serie de condiciones. Los criterios de cobertura, definen el conjunto de reglas que debe cumplir dicho conjunto con respecto a las consultas que se desean probar. En [4] se propone un criterio de cobertura para consultas SQL denominado SQLfpc, el cual se expresa mediante un conjunto de reglas. Este criterio permite, por un lado, minimizar el número de instancias de base de datos que constituyen el conjunto final de los casos de prueba, y por otro lado, cubrir el mayor número de situaciones en las consultas SQL.

En este trabajo hemos desarrollado un prototipo de herramienta, capaz

de generar de forma automática, las reglas de cobertura que definen el criterio de cobertura SQLfpc para el caso de consultas SQL. En el proceso de generación, tenemos en cuenta la semántica de SQL junto con las restricciones de integridad impuestas por la base de datos. Aunque el subconjunto del lenguaje SQL que cubrimos incluye solo consultas básicas, nuestro prototipo es fácilmente ampliable para soportar un conjunto mayor de consultas SQL.

1.1. Criterios de cobertura

En general, el diseño de los casos de prueba se puede realizar desde dos perspectivas: de *caja blanca* y de *caja negra*. Las pruebas de caja blanca, son pruebas estructurales, conociendo el código y siguiendo su estructura lógica, se pueden diseñar pruebas destinadas a comprobar que hace correctamente lo que el diseño de bajo nivel indica. Las pruebas de caja negra, se centran en distintos módulos, en encontrar los errores en un fragmento de código, por eso también son conocidas como pruebas funcionales, ya que se centran en la entrada y en la salida.

En este trabajo perseguimos un diseño de caja blanca. Entre los criterios de cobertura de caja blanca para expresiones lógicas destacamos los siguientes [2, 1]:

- Criterio de cobertura de sentencias: Comprueba que todas las sentencias del programa se ejecuten al menos una vez.
- Criterio de cobertura de predicado o decisiones: Determina que todas las sentencias obtienen todos los posibles valores.

  Una condición es una expresión booleana que puede tomar valor true o false. Una decisión o predicado es una lista de condiciones conectadas por los operadores lógicos "*and*" y "*or*".

  Por ejemplo para que el predicado $p := (c_1 \wedge c_2)$ tome el valor cierto tiene que cumplirse que $c_1 = true$ y $c_2 = true$.

  Para que p tome el valor falso, basta con que c_1 o c_2 tome el valor falso.

  Por tanto el conjunto C de casos de prueba que verifica el criterio de cobertura de predicado ha de cumplir las dos siguientes reglas:

$$R1 : c_1 = true \wedge c_2 = true$$

$$R2 : c_1 = false \wedge c_2 = true$$

- Criterio de cobertura de condiciones: Este criterio, se basa en presentar un número suficiente de casos de prueba, para que cada condición en una decisión, tenga al menos una vez, todos los resultados posibles.

  Por ejemplo, para el predicado $p := c_1 \wedge c_2$, tenemos dos casos de prueba iniciales:

  $$R1 : c_1 = true, c_2 = false$$

  $$R2 : c_1 = false, c_2 = true$$

- Criterio de cobertura de condición de decisión: Cada condición de cada decisión toma valores cierto y falso al menos una vez y cada decisión toma valores cierto y falso al menos una vez.

  Así, un conjunto C de casos de prueba para el predicado $p := c_1 \wedge c_2$, cumple el criterio de cobertura de condición decisión si verifica las reglas:

  $$R1 : c_1 = true, c_2 = true$$

  $$R2 : c_1 = false, c_2 = false$$

- Criterio de cobertura modificada de condición decisión (MCDC): Este criterio requiere que cada condición afecte independientemente a cada decisión. Por ejemplo, para un predicado $p := (c_1 \wedge c_2)$, un conjunto de casos de prueba cumple este criterio si verifica las siguientes tres reglas:

  $$R1 : c_1 = true, c_2 = true$$

  $$R2 : c_1 = false, c_2 = true$$

  $$R3 : c_1 = true, c_2 = false$$

  Será un conjunto que verifica MCDC.

- Criterio de cobertura de condiciones múltiples: Requiere un número suficiente de casos de prueba, tal que todas las condiciones tomen valor cierto y falso de manera que se recorra la tabla de verdad completa de la decisión. Por tanto para el predicado $p := (c_1 \wedge c_2)$, obtenemos las siguientes cuatro reglas:

  $$R1 : c_1 = true, c_2 = true$$

$$R2 : c_1 = false, c_2 = true$$

$$R3 : c_1 = true, c_2 = false$$

$$R4 : c_1 = false, c_2 = false$$

En el campo de las bases de datos, el criterio de cobertura más apropiado, es el criterio SQLfpc (full predicate coverage) [4]. Este criterio, no es mas, que el MCDC aplicado al caso particular de las consultas SQL.

El procedimiento para determinar las reglas que definen este criterio se basa en un recorrido del árbol sintáctico del predicado a evaluar desde las hojas hasta la raíz. Dada una condición c, el objetivo es encontrar el valor del resto de las condiciones del predicado p para que c determine el valor de p.

1.2. Aportaciones

En este proyecto, implementamos las ideas presentadas en [4], en el sistema DES (Data Educational System) [3]. Actualmente este sistema permite generar de forma automática casos de prueba individuales para una consulta SQL que cumple el criterio de cobertura de predicado, lo que ha motivado este proyecto. Este sistema se encuentra desarrollado en lenguaje Prolog, y dispone de varios módulos que permiten la gestión de bases de datos deductivas y relacionales, permitiendo el uso del lenguaje SQL, tanto para la definición de datos, como para la manipulación de los mismos.

Nuestra herramienta toma como entrada un esquema de bases de datos y una vista SQL, produciendo como salida un conjunto de reglas de cobertura. Este conjunto de reglas permitirá, por un lado, comprobar que un conjunto de instancias de la base de datos cumple el criterio de cobertura SQLfcp, y por otro lado, generar de forma automática un conjunto de casos de prueba que cumple dicho criterio.

1.3. Estructura del trabajo

Este documento esta dividido en 5 capítulos. En el capítulo 2 se explica en rasgos generales el ámbito de la herramienta y el lenguaje utilizado para la realización de la misma. En el capítulo 3 se basa en las distintas reglas de cobertura y las diferentes formas de generarlas según la semántica de la

consulta. A lo largo del capítulo 4 se presenta la forma de ejecutar la herramienta y los distintos pasos a seguir. Finalmente en el capítulo 5 nuestras conclusiones tras haber desarrollado este proyecto y el trabajo que se podría realizar en versiones futuras.

Capítulo 2

Análisis y diseño

En este capítulo, se define el entorno de aplicación del proyecto. En primer lugar, exponemos unos conceptos básicos, que nos pueden resultar útiles para la comprensión de este documento.

2.1. Bases de datos relacionales

Una *base de datos relacional* es una base de datos, en donde todos los datos visibles al usuario, están organizados estrictamente como tabla de valores, y en donde todas las operaciones de la base de datos, operan sobre estas tablas.

Una *relación* se representa de la forma $R(A_1, \ldots, A_n)$ donde R es el nombre de la relación y $A_1, \ldots, A_n$ es un conjunto de *atributos* con dominio D_1,D_2, ...,D_n respectivamente. La relación $R(A_1, \ldots, A_n)$, se puede representar simplemente como R. El conjunto de tuplas de una relación, se denomina *instancia* de la relación. En el caso particular de SQL, una relación es una tabla o una vista. Los atributos son columnas, y las tuplas son las filas. Se dice, que un *atributo es base*, si no depende de otros atributos. En caso contrario se dice que es un *atributo derivado*. El conjunto de atributos base de un atributo derivado F, se denota por $battrs(F)$. Un atributo es *anulable* si puede tomar el valor NULL en la instancia de la base de datos. Así el predicado booleano *nullable(A_i)* es cierto si el atributo A_i es anulable.

En SQL los predicados siguen una lógica trivaluada para permitir representar los valores nulos del modelo relacional. Así, un predicado es una expresión de la lógica trivaluada que toma los valores cierto, falso e indefinido o nulo. Un predicado puede estar formado por variables de tipo booleano

y no booleano. Los *operadores lógicos* que pueden aparecer en un predicado son: $\{\neg, \wedge, \vee\}$. Un ejemplo de predicado podría ser:

$$P \equiv ((a < b) \vee c) \wedge p(x))$$

Una *cláusula*, es un predicado que no contiene ningún operador lógico. Si partimos del ejemplo del predicado anterior, obtenemos tres cláusulas, las cuales son $C_1 \equiv (a < b)$, la función booleana $C_2 = p(x)$, y una variable booleana $C_3 \equiv c$.

El valor de un atributo A_i en una tupla dada toma el valor NULL si y solo si su valor es desconocido en el momento en que se evalúa dicho atributo. Definimos el predicado booleano $nl(A_i)$, como el predicado que devuelve *true*, para aquellas tuplas en las cuales el atributo A_i toma el valor NULL. Un predicado p, se dice que es *base* o *básico*, si es una expresión lógica trivaluada, que no contiene operadores lógicos. Se dice que es *derivado*, en caso contrario. Dado un predicado p, el predicado $bpreds(p)$, devuelve el conjunto de todos los predicados básicos en p.

Por ejemplo, el predicado $p := A_1 + A_2 = 0 \wedge nl(A_3)$ está compuesto por dos predicados base. Por tanto: $bpreds(p) = \{(A_1 + A_2 = 0), nl(A_3)\}$. El primer predicado base contiene dos atributos base, $battrs(A_1 + A_2 = 0) = \{A_1, A_2\}$. El segundo predicado tiene un atributo anulable, por tanto, $battrs(nl(A_3)) = \{A_3\}$.

2.2. Ámbito de aplicación

El prototipo de herramienta que hemos desarrollado, admite código SQL de definición de datos, así como, lenguaje de consulta de datos con las siguientes características:

- Restricciones de integridad: se tienen en cuenta las restricciones de integridad impuestas por el esquema de base de datos (claves primarias, claves ajenas y restricciones de tipo NOT NULL).
- Las consultas SQL admitidas tienen las siguientes características:
  - Consultas básicas: las consultas básicas, se representan como:

    $$Z \longleftarrow R[p(A)]$$

    Son consultas que acceden a una única relación de la base de datos, y cuyo resultado es el conjunto Z de tuplas de la relación $R(A)$

que cumple la condición expresada por el predicado *p(A)*. Estas consultas se representan en SQL mediante la siguiente expresión:

CREATE VIEW Z AS
SELECT *
FROM R
WHERE p(A)

Un ejemplo de las consultas básicas sería:
CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.A >= 3
AND T.B = 0

- Consultas de combinación: se representan como:

$$Z \longleftarrow R[p(A,B)]S$$

Se trata de consultas que combinan las filas de dos relaciones (tablas o vistas). El resultado es el conjunto Z de tuplas de las relaciones *R(A)* y *S(B)* que cumplen la condición especificada en el predicado *p(A,B)*.

Podemos distinguir tres tipos:

  - combinación interna:

$$Z \longleftarrow R[p(A,B)]^I S$$

Esta operación, permite emparejar, filas o tuplas de distintas tablas, haciendo el producto cartesiano completo, y luego seleccionando las filas que cumplen la condición de emparejamiento. Sólo se muestran las filas que aparecen en ambas tablas.

  - combinación por la izquierda:

$$Z \longleftarrow R[p(A,B)]^L S$$

Esta operación, consiste en añadir al resultado del INNER JOIN, las filas de la tabla de la izquierda, que no tienen correspondencia en la otra tabla (rellenando los atributos de la tabla de la derecha con los valores NULL).

- combinación por la derecha:

$$Z \longleftarrow R[p(A,B)]^R S$$

Esta operación consiste en añadir al resultado del INNER JOIN, las filas de la tabla de la derecha, que no tienen correspondencia en la otra tabla (rellenando los atributos de la tabla de la izquierda con los valores NULL).

Estas consultas se representan en SQL mediante la siguiente expresión:

```
CREATE VIEW Z AS
SELECT *
FROM R INNER JOIN S ON P(A,B)
```

Por ejemplo:

```
CREATE VIEW Z AS
SELECT *
FROM T1 INNER JOIN T2
WHERE T1.A = T2.A
```

- Consultas de agregación: se representan como:

$$Z \longleftarrow R[p(A)]//G[q(G,F)]$$

Mediante las consultas de agregación es posible agrupar las tuplas de la relación R por un conjunto de atributos G a los denominamos atributos de agrupación. Sobre cada uno de los grupos de tuplas es posible especificar una condición de agregación q, F es el conjunto de atributos derivados de la forma f(A). Cada función de agregación f se aplica a las filas de cada uno de los grupos. Por ejemplo la media aritmética AVG, máximo MAX, mínimo MIN, suma SUM, etc. Estas consultas se representan en SQL mediante la siguiente expresión:

```
CREATE VIEW Z AS
SELECT G, F
```

```
FROM R
WHERE p(A)
GROUP BY G
HAVING q(G,F)
```

Por ejemplo:

```
CREATE VIEW Z AS
SELECT T.C, SUM(T.B)
FROM T
GROUP BY T.C
HAVING SUM(T.B) > 150
```

- TIPOS: los datos pueden ser de distintos tipos, de tipo entero INT, tipo decimal FLOAT, o de cadena de caracteres CHAR.

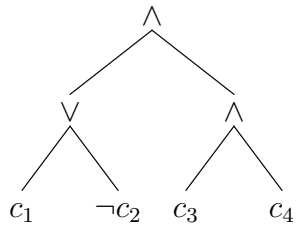
Capítulo 3

Generación de reglas de cobertura

3.1. Preliminares

- Un predicado p se puede representar mediante un árbol sintáctico donde las hojas se corresponden con predicados $c_i \in bpreds(p)$ y los nodos intermedios se corresponden los operadores lógicos $\vee$, $\wedge$.

  Por ejemplo, dado el predicado $p := (c_1 \vee \neg c_2) \wedge (c_3 \wedge c_4)$, su representación en forma de árbol es la siguiente:



- Condiciones de test: Las condiciones de test de una consulta básica es cada uno de los predicados $c_i \in$ *bpreds(p)*. Así el valor de cada condición de test en el cada una de las tuplas del resultado de la consulta puede tomar los valores cierto, falso o null.

- Sea una consulta básica $R[p(A)]$. Sea el predicado p de la forma p:= p_1 lop p_2 lop .... lop p_n, con lop $\in \{\vee, \wedge\}$ y cada p_i un predicado base o derivado. Definimos SIP (Sibling Independence Predicate) para cada sub-predicado p_i de la siguiente manera:

$$SIP(p_i) = \begin{cases} \wedge_{i \neq j} p_j \text{ si lop es } \wedge \\ \wedge_{i \neq j} \neg p_j \text{ si lop es } \vee \end{cases}$$

- El predicado independiente (IP) de p con respecto a la condición de test $c_i \in$ *bpreds(p)* es un predicado definido de forma recursiva de siguiente manera:

$$IP(c_i) := SIP(c_i) \wedge IP(padre(c_i))$$

- NR (Null Reduction Transformation): Sea p un predicado, una consulta básica $R[p(A)]$ y sea $A_k \in battrs(p)$. Definimos $NR(p, A_k)$ como el predicado que se obtiene después de eliminar de p todas las apariciones $c_i \in bpred(p)$ tales que $A_k \in battrs(c_i)$. Por ejemplo si tenemos la consulta $R[(A_1 = 0 \wedge A_2 = 3) \vee A_1 = A_2]$, obtenemos:

  $NR(p, A_1) \equiv A_2 = 3$

  $NR(p, A_2) \equiv A_1 = 0$

- $\Phi_T(p, c_i)$: Transformación positiva de p con respecto a c_i. El cálculo se define de la siguiente manera:

$$\Phi_T(p, c_i) := c_i \wedge IP(c_i)$$

- $\Phi_F(p, c_i)$: Transformación negativa de p con respecto a c_i. Se define de la siguiente manera:

$$\Phi_F(p, c_i) := \neg c_i \wedge IP(c_i)$$

- $\Phi_N(p, c_i, A_k)$: Transformación nula de p con respecto a c_i y a A_k. Se define de la siguiente manera:

$$\Phi_N(p, c_i, A_k) := nl(A_k) \wedge NR(IP(c_i), A_k)$$

3.2. Reglas de cobertura para consultas básicas

Sea la consulta $R[p(A)]$ con c_i cada uno de los predicados básicos del predicado p. El conjunto de reglas de cobertura Δ_T (regla de cobertura positiva), Δ_F (regla de cobertura negativa) y Δ_N (regla de cobertura del nulo), se definen como:

1. $\forall c_i \in bpreds(p) : \Delta_T(p, c_i) := R[\Phi_T(p, c_i)]$
2. $\forall c_i \in bpreds(p) : \Delta_F(p, c_i) := R[\Phi_F(p, c_i)]$
3. $\forall c_i \in bpreds(p), \forall A_k \in battrs(c_i) \mid nullable(A_k) : \Delta_N(p, c_i, A_k) := R[\Phi_N(p, c_i, A_k)]$

Si el predicado c_i es de la forma $nl(A)$ o $not(nl(A))$, solo es necesario generar las reglas asociadas a los puntos 1 y 3 que se corresponden respectivamente a las transformaciones Φ_T y Φ_N.

Ejemplo: Sea la siguiente consulta básica:

$$Z \leftarrow T[A = 1 \wedge (B + C = 2 \vee C = B)]$$

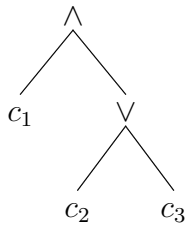
La consulta anterior se representa en el lenguaje SQL con la siguiente expresión:

```
CREATE VIEW Z AS
SELECT *
FROM T
WHERE T.A = 1 AND (T.B + T.C = 2 OR T.C = T.B)
```

En primer lugar determinamos los predicados básicos del predicado definido en la sección WHERE de la consulta. Así,

$$bpreds(p(A = 1 \wedge (B + C = 2 \vee C = B))) = \{(A = 1), (B + C = 2), (C = B)\}$$

Como se puede apreciar, hay tres predicados básicos $\{c_1 := A = 1, c_2 := B + C = 2, c_3 := C = B\}$.
A continuación, con los predicados básicos y los operadores lógicos, se puede construir el siguiente árbol sintáctico:



Sea $c_1 := T.A = 1$, $c_2 := T.B + T.C = 2$ y $c_3 := T.C = T.B$. Las reglas de cobertura asociadas a la consulta son:

1. Reglas de cobertura positivas:

   R1. $\Delta_T(p, c_1) = T[(T.A = 1) \wedge (c_2 \vee c_3)]$
   R2. $\Delta_T(p, c_2) = T[(T.B + T.C = 2) \wedge (\neg c_3 \wedge c_1)]$
   R3. $\Delta_T(p, c_3) = T[(T.C = T.B) \wedge (\neg c_2 \wedge c_1)]$

2. Reglas de cobertura negativas:

   R4. $\Delta_F(p, c_1) = T[\neg(T.A = 1) \wedge (c_2 \vee c_3)]$
   R5. $\Delta_F(p, c_2) = T[\neg(T.B + T.C = 2) \wedge (\neg c_3 \wedge c_1)]$
   R6. $\Delta_F(p, c_3) = T[\neg(T.C = T.B) \wedge (\neg c_2 \wedge c_1)]$

Supongamos que el atributo B es anulable, es decir, B puede tomar el valor NULL en la instancia de la base de datos. Como se puede apreciar, el atributo B forma parte de los predicados básicos c_2 y c_3, por lo que es posible generar las reglas de cobertura del nulo asociadas a los predicados c_2 y c_3 y el atributo B.

3. Reglas de cobertura del nulo:

   R7. $\Delta_N(p, c_2, B) = T[nl(B) \wedge c_1]$
   Lo primero que se hace es modificar el predicado $p(A = 1 \wedge (B + C = 2 \vee C = B))$, eliminando los predicados básicos en los que aparece el atributo B.
   R8. $\Delta_N(p, c_3, B) := T[nl(B) \wedge c_1]$

A continuación mostramos la representación mediante lenguaje SQL de las reglas de cobertura R1 - R8 para la consulta:

```
CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.A = 1 AND (T.B + C = 2 OR T.C = T.B)
```

R1(SQL).
```
CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.A = 1 AND (T.B + T.C = 2 OR T.C = T.B)
```

R2(SQL). CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.B + T.C = 2 AND (NOT T.C = T.B AND T.A = 1)

R3(SQL). CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.C = T.B AND (NOT T.B + T.C = 2 AND T.A = 1)

R4(SQL). CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE NOT(T.A = 1) AND (T.B + T.C = 2 OR T.C = B)

R5(SQL). CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE NOT (T.B + T.C = 2) AND (NOT T.C = T.B AND T.A = 1)

R6(SQL). CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE NOT (T.C = T.B) AND (NOT T.B + T.C = 2 AND T.A = 1)

R7(SQL). CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.B IS NULL AND T.A = 1

R8(SQL). CREATE VIEW Z AS
SELECT T.A, T.B
FROM T

WHERE T.B IS NULL AND T.A = 1

3.2.1. Reglas de cobertura con limites

Sea c_i un predicado base de la forma $a\ rop\ b$, donde rop es un operador relacional en $\{=, \neq, >, \geq, <, \leq\}$. Si el dominio de ambos atributos es numérico, las reglas de transformación Φ_T y Φ_F son reemplazas por:

$$\Phi_{B_+}(p, c_i) := (a \ = \ b + 1) \wedge IP(c_i)$$

$$\Phi_{B_=}(p, c_i) := (a \ = \ b) \wedge IP(c_i)$$

$$\Phi_{B_-}(p, c_i) := (a \ = \ b - 1) \wedge IP(c_i)$$

Las reglas de cobertura Φ_T y Φ_F se sustituyen por las siguientes reglas de cobertura:

1. $c_i \in bpreds(p) : \Delta_{B_+}(p, c_i) := R[\Phi_{B_+}(p, c_i)]$
2. $c_i \in bpreds(p) : \Delta_{B_=}(p, c_i) := R[\Phi_{B_=}(p, c_i)]$
3. $c_i \in bpreds(p) : \Delta_{B_-}(p, c_i) := R[\Phi_{B_-}(p, c_i)]$

En el ejemplo anterior los atributos A, B y C son numéricos, por lo que las reglas de cobertura positivas y negativas serán sustituidas respectivamente por sus reglas con limites.

1. Para la cláusula c_1 las reglas de cobertura asociadas a la consulta son:

   R1. $\Delta_{B_+}(p, c_1) := T[(A = 2) \wedge c_2 \vee c_3]$

   R2. $\Delta_{B_=}(p, c_1) := T[(A = 1) \wedge c_2 \vee c_3]$

   R3. $\Delta_{B_-}(p, c_1) := T[(A = 0) \wedge c_2 \vee c_3]$

2. Para la cláusula c_2 las reglas de cobertura asociadas a la consulta son:

   R4. $\Delta_{B_+}(p, c_2) := T[(B + C = 3) \wedge c_3 \wedge c_1]$

R5. $\Delta_{B_=}(p, c_2) := T[(B + C = 2) \wedge c_3 \wedge c_1]$

R6. $\Delta_{B_-}(p, c_2) := T[(B + C = 1) \wedge c_3 \wedge c_1]$

R7. $\Delta_N(p, c_2, B) := T[nl(B) \wedge c_1)]$

A continuación mostramos la representación mediante lenguaje SQL de las reglas de cobertura R1 - R7 para la consulta:

```
CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.A = 1 AND (T.B + T.C = 2 OR T.C = T.B)
```

R1(SQL).
```
CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.A = 2 AND (T.B + T.C = 2 OR T.B = T.C)
```

R2(SQL).
```
CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.A = 1 AND (T.B + T.C = 2 OR T.B = T.C)
```

R3(SQL).
```
CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.A = 0 AND (T.B + T.C = 2 OR T.B = T.C)
```

R4(SQL).
```
CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.B + T.C = 3 AND ( NOT T.C = T.B AND T.A = 1)
```

R5(SQL). CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.B + T.C = 2 AND (NOT T.C = T.B AND T.A = 1)

R6(SQL). CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.B + T.C = 1 AND (NOT T.C = B AND T.A = 1)

R7(SQL). CREATE VIEW Z AS
SELECT T.A, T.B
FROM T
WHERE T.B = NULL AND T.A = 1

3.3. Reglas de cobertura para consultas agrupadas

Después de generar las reglas de cobertura anteriormente descritas, se explica el desarrollo y generación de las reglas de cobertura para el operador GROUP BY y HAVING.

3.3.1. Reglas de cobertura sin operador HAVING

Sea $Z \leftarrow R[p(A)]///G$ una consulta agrupada, con $G \subseteq A$ el conjunto de atributos de agrupación tal que $G = \{G_1, \ldots, G_n\}$. Sea $B \in A$ el conjunto de atributos que aparecen en las funciones de agrupación de la sección SELECT. Para la consulta Z se generan las reglas de cobertura Φ_T, Φ_F y Φ_N como se indica en la Sección 3.2. De forma adicional se generan las siguientes reglas de cobertura relacionadas con el conjunto de atributos G y B:

1. $\Delta_G := R[p(A)]///G[count(*) > 1]$
2. $\forall G_i \in G, X = G - G_i : \Delta_G(G_i) := R[p(A)]///X[count(distinct(G_i)) > 1]$

3. $\forall B_i \in B$:

   3.1) Si B_i es un atributo anulable:

   - $\Delta_G(B_i, 1) := R[p(A)]///G[count(distinct(B_i)) > 1 \, and \, count(B_i) > count(distinct(B_i))]$
   - $\Delta_G(B_i, 2) := R[p(A)]///G[count(distinct(B_i)) > 1 \, and \, count(*) > count(B_i)]$

   3.2) Si B_i no es un atributo anulable, se genera solo la regla de cobertura $\Delta_G(B_i, 1)$.

Sea la siguiente consulta agrupada:

$$Z \leftarrow T[p(A)]///G$$

con $p(A) := T.C = T.B + 1$ y $G = \{T.A\}$

La consulta anterior se representa en el lenguaje SQL con la siguiente expresión:

```
CREATE VIEW Z AS
SELECT T.A
FROM T
WHERE T.C = T.B + 1
GROUP BY T.A
```

Sea $c_1 := T.C = T.B + 1$. Las reglas de cobertura asociadas a la consulta son:

1. Reglas de cobertura positivas:

   R1. $\Delta_T(p, T.C = T.B + 1) = T[T.C = T.B + 1]$

2. Reglas de cobertura negativas:

   R2. $\Delta_F(p, T.C = T.B + 1) = T[\neg(T.C = T.B + 1)]$

3. Suponiendo que el atributo T.B es el único atributo anulable, las reglas de cobertura del nulo son:

R3. $\Delta_N(p, T.C = T.B + 1, B) = T[nl(B)]$

4. Reglas de cobertura relacionadas con el conjunto de atributos G:

   R4. $\Delta_G := T[T.C = T.B + 1]///G[count(*) > 1]$

   R5. $\Delta_G(T.A) := T[T.C = T.B + 1]///X[count(distinct(T.A)) > 1]$, con $X = \emptyset$

A continuación mostramos la representación mediante lenguaje SQL de las reglas de cobertura R1 - R5:

```
R1(SQL). CREATE VIEW Z AS
         SELECT T.A
         FROM T
         WHERE T.C = T.B + 1
         GROUP BY T.A

R2(SQL). CREATE VIEW Z AS
         SELECT T.A
         FROM T
         WHERE NOT(T.C = T.B + 1)
         GROUP BY T.A

R3(SQL). CREATE VIEW Z AS
         SELECT T.A
         FROM T
         WHERE T.B IS NULL
         GROUP BY T.A

R4(SQL). CREATE VIEW Z AS
         SELECT T.A
         FROM T
         WHERE T.C = T.B + 1
         GROUP BY T.A
         HAVING count (*) > 1
```

R5(SQL). CREATE VIEW Z AS
SELECT T.A
FROM T
WHERE T.C = T.B + 1
HAVING count (distinct (T.A)) > 1

3.3.2. Reglas de cobertura con operador HAVING

Sea $Z \leftarrow R[p(A)]///G[q(G,F)]$ una consulta agrupada, con $G \subseteq A$ el conjunto de atributos de agrupación tal que $G = \{G_1, \ldots, G_n\}$. Sea $B \in A$ el conjunto de atributos básicos incluidos en el conjunto de atributos derivados F. Así, B contiene todos los atributos $B_i \in A$ tal que B_i es un operando de una función de agregación f. Para la consulta Z se generan las reglas de cobertura Φ_T, Φ_F y Φ_N como se indica en la Sección 3.2. tanto para el predicado $p(A)$ como para el predicado $q(G,F)$. De forma adicional se generan las siguientes reglas de cobertura:

1. $\Delta_G := R[p(A)]///G[q(G,F)\ and\ count(*) > 1]$
2. $\forall G_i \in G, X = G - G_i : \Delta_G(G_i) := R[p(A)]///X[q(G,F)\ and\ count(distinct(G_i)) > 1]$
3. $\forall B_i \in B$:

   3.1) Si B_i es un atributo anulable:
   - $\Delta_G(B_i, 1) := R[p(A)]///G[q(G,F)\ and\ count(distinct(B_i)) > 1\ and\ count(B_i) > count(distinct(B_i))]$
   - $\Delta_G(B_i, 2) := R[p(A)]///G[q(G,F)\ and\ count(distinct(B_i)) > 1\ and\ count(*) > count(B_i)]$

   3.2) Si B_i no es un atributo anulable, se genera solo la regla de cobertura $\Delta_G(B_i, 1)$.

Sea la siguiente consulta agrupada:

$$Z \leftarrow T[p(A)]///G[q(G,F)]$$

con $p(A) := T.C = T.B + 1$, $G = \{T.A\}$ y $q(G, F) := SUM(T.A) > 1$

La consulta anterior se representa en el lenguaje SQL con la siguiente expresión:

```
CREATE VIEW Z AS
SELECT T.A
FROM T
WHERE T.C = T.B + 1
GROUP BY T.A
HAVING SUM(T.A) > 1
```

Sea $c_1 := T.C = T.B + 1$. Las reglas de cobertura asociadas a la consulta son:

1. Reglas de cobertura positivas para el predicado $T.C = T.B + 1$:

   R1. $\Delta_T(p, T.C = T.B + 1) = T[T.C = T.B + 1]$

2. Reglas de cobertura negativas para el predicado $T.C = T.B + 1$:

   R2. $\Delta_F(p, T.C = T.B + 1) = T[\neg(T.C = T.B + 1)]$

3. Reglas de cobertura del nulo para el predicado $T.C = T.B + 1$:

   R3. $\Delta_N(p, T.C = T.B + 1, B) = T[nl(B)]$

4. Reglas de cobertura positivas para el predicado $SUM(T.A) > 1$:

   R4. $\Delta_T(q, SUM(T.A) > 1) = T[SUM(T.A) > 1]$

5. Reglas de cobertura negativas para el predicado $SUM(T.A) > 1$:

   R5. $\Delta_F(q, SUM(T.A) > 1) = T[\neg SUM(T.A) > 1]$

6. Reglas de cobertura relacionadas con el conjunto de atributos G:

   R6. $\Delta_G := T[T.C = T.B + 1]///G[SUM(T.A) > 1 \; and \; count(*) > 1]$

   R7. $\Delta_G(T.A) := T[T.C = T.B + 1]///X[SUM(T.A) > 1 \, and \, count(distinct(T.A)) > 1]$, con $X = \emptyset$

A continuación mostramos la representación mediante lenguaje SQL de las reglas de cobertura R1 - R7:

R1(SQL). CREATE VIEW Z AS
SELECT T.A
FROM T
WHERE T.C = T.B + 1
GROUP BY T.A
HAVING SUM(T.A) > 1

R2(SQL). CREATE VIEW Z AS
SELECT T.A
FROM T
WHERE NOT(T.C = T.B + 1)
GROUP BY T.A
HAVING SUM(T.A) > 1

R3(SQL). CREATE VIEW Z AS
SELECT T.A
FROM T
WHERE T.B = NULL
GROUP BY T.A
HAVING SUM(T.A) > 1

R4(SQL). CREATE VIEW Z AS
SELECT T.A
FROM T
WHERE T.C = T.B + 1
GROUP BY T.A
HAVING SUM(T.A) > 1

R5(SQL). CREATE VIEW Z AS
SELECT T.A
FROM T
WHERE T.C = T.B + 1
GROUP BY T.A

```
        HAVING NOT(SUM(T.A) > 1)
```

R6(SQL).
```
CREATE VIEW Z AS
SELECT T.A
FROM T
WHERE T.C = T.B + 1
GROUP BY T.A
HAVING SUM(T.A) > 1 AND count (*) > 1
```

R7(SQL).
```
CREATE VIEW Z AS
SELECT T.A
FROM T
WHERE T.C = T.B + 1
HAVING SUM(T.A) > 1 AND count (distinct (T.A)) > 1
```

3.4. Reglas de cobertura para consultas de combinación

Sea $Z \leftarrow R[p]S$ una consulta de combinación como las presentadas en la sección 2.2. A continuación definimos las transformaciones de la combinación necesarias para definir las reglas de cobertura para este tipo de consultas.

- $\Phi_{R[p]S}(JT, R[p]S)$: Transformación de la operación JOIN. Sea $R[p]S$ una consulta de combinación. Sea $JT = \{L, R, I\}$ el conjunto de tipos de operación representando las operaciones left outer join, right outer join y full outer join respectivamente. La regla de transformación $\Phi_{R[p]S}(JT, R[p]S)$ se define como:

$$\Phi_{R[p]S}(JT, R[p]S) = R[p]^{JT}S$$

Sean lattrs(p) y rattrs(p) el conjunto de atributos de la relación izquierda y derecha de la consulta respectivamente que aparecen en el predicado p.

- Φ_{LOI}(R[p]S): Transformación de la operación LEFT OUTER JOIN. Sea $R[p]S$ una consulta de combinación. La regla de transformación Φ_{LOI}(R[p]S) se define como:

$$\Phi_{LOI}(R[p]S) := (\wedge_{A_i \in lattrs(p)} \neg nl(A_i)) \wedge (\wedge_{B_i \in rattrs(p)} nl(B_i))$$

- Φ_{ROI}(R[p]S): Transformación de la operación RIGHT OUTER JOIN. Sea $R[p]S$ una consulta de combinación. La regla de transformación Φ_{ROI}(R[p]S) se define como:

$$\Phi_{ROI}(R[p]S) := (\wedge_{A_i \in lattrs(p)} nl(A_i)) \wedge (\wedge_{B_i \in rattrs(p)} \neg nl(B_i))$$

A continuación definimos las transformaciones de la combinación externa por la izquierda y por la derecha en el caso de la existencia de atributos anulables. Sea $R[p]S$ una consulta de combinación y p un predicado con atributos anulables. Sean A_k y B_k cada uno de los atributos anulables tal que $A_k \in lattrs(p)$ y $B_k \in rattrs(p)$. Se definen las siguientes transformaciones:

- Φ_{NLOI}(R[p]S, A_k): Transformación de la operación LEFT OUTER JOIN con atributos anulables. La regla de transformación Φ_{NLOI}(R[p]S) se define como:

$$\Phi_{NLOI}(R[p]S, A_k) := (\wedge_{A_i \in lattrs(p)-\{A_k\}} \neg nl(A_i)) \wedge nl(A_k) \wedge (\wedge_{B_i \in rattrs(p)} nl(B_i))$$

- Φ_{NROI}(R[p]S, B_k): Transformación de la operación RIGHT OUTER JOIN con atributos anulables. La regla de transformación Φ_{NROI}(R[p]S) se define como:

$$\Phi_{NROI}(R[p]S, B_k) := (\wedge_{A_i \in lattrs(p)} nl(A_i)) \wedge nl(B_k) \wedge (\wedge_{B_i \in rattrs(p)-\{B_k\}} \neg nl(B_i))$$

3.4.1. Reglas de cobertura para consultas de combinación

Sea $Z \leftarrow R[p]S$ una consulta combinada. Para la consulta Z se generan las siguientes reglas de cobertura:

1. $\Delta_I(R[p]S) := \Phi_{R[p]S}(I, R[p]S)$
2. $\Delta_L(R[p]S) := \Phi_{R[p]S}(L, R[p]S)[\Phi_{LOI}(R[p]S)]$
3. $\Delta_R(R[p]S) := \Phi_{R[p]S}(R, R[p]S)[\Phi_{ROI}(R[p]S)]$

En el caso de que aparezcan atributos anulables en el predicado p, se definen las siguientes reglas de cobertura:

1. $\forall A_i \in lattrs(p) \backslash nullable(A_i)$:
   $\Delta_{NL}(R[p]S, A_i) := \Phi_{R[p]S}(L, R[p]S)[\Phi_{NLOI}(R[p]S), A_i]$

2. $\forall B_i \in rattrs(p) \backslash nullable(B_i)$:
   $\Delta_{NR}(R[p]S, B_i) := \Phi_{R[p]S}(R, R[p]S)[\Phi_{NROI}(R[p]S), B_i]$

Veamos un ejemplo. Sea la siguiente consulta agrupada:

$Z \leftarrow T[A_1 = B_1]^F S$

donde F representa la operación FULL OUTER JOIN.

La consulta anterior se representa en el lenguaje SQL con la siguiente expresión:

CREATE VIEW Z AS
SELECT *
FROM T FULL OUTER JOIN S ON $A_1 = B_1$

Las reglas de cobertura asociadas a la consulta son:

1. Regla de cobertura asociada al inner join:

   R1. $\Delta_I(T[A_1 = B_1]^F S) := T[A_1 = B_1]^I S$

2. Reglas de cobertura para el left outer join:

   R2. $\Delta_L(T[A_1 = B_1]^F S) := (T[A_1 = B_1]^L S)[\neg nl(A_1) \wedge nl(B_1)]$

3. Reglas de cobertura para el right outer join:

   R3. $\Delta_R(T[A_1 = B_1]^F S) := (T[A_1 = B_1]^R S)[nl(A_1) \wedge \neg nl(B_1)]$

Supongamos ahora que el atributo A_1 es anulable, las reglas de cobertura asociadas a la consulta son:

1. Regla de cobertura asociada al inner join:

   R1. $\Delta_I(T[A_1 = B_1]^F S) := T[A_1 = B_1]^I S$

2. Reglas de cobertura para el left outer join:

   R2. $\Delta_{NL}(T[A_1 = B_1]^F S, A_1) := (T[A_1 = B_1]^L S)[nl(A_1) \wedge nl(B_1)]$

3. Reglas de cobertura para el right outer join:

   R3. $\Delta_R(T[A_1 = B_1]^F S) := (T[A_1 = B_1]^R S)[nl(A_1) \wedge \neg nl(B_1)]$

A continuación mostramos la representación mediante lenguaje SQL de las reglas de cobertura R1 - R3, en el caso de que todos los atributos son no nulos:

R1(SQL). CREATE VIEW Z AS
SELECT *
FROM T INNER JOIN S ON $A_1 = B_1$

R2(SQL). CREATE VIEW Z AS
SELECT *
FROM T LEFT OUTER JOIN S ON $A_1 = B_1$
WHERE A_1 IS NOT NULL AND B_1 IS NULL

R3(SQL). CREATE VIEW Z AS
SELECT *
FROM T RIGHT OUTER JOIN S ON $A_1 = B_1$
WHERE A_1 IS NULL AND B_1 IS NOT NULL

Capítulo 4

Pruebas de ejecución

En este capítulo se mostrarán dos ejemplos de consultas para las cuales generaremos las reglas de cobertura de forma automática mediante nuestro prototipo de herramienta.

En dicha memoria se adjuntan un CD. Para que la herramienta implementada pueda realizar su función es necesario tener instalado el *SICstus Prolog VC9 4.1.2*. En el CD se encuentra la carpeta *des* con toda la implementación. En esta carpeta nos centraremos en dos ficheros principalmente, el *des_crules.pl* y el *myex.sql*. En el fichero *des_crules.pl* es donde esta toda la implementación llevada a cabo, y en el fichero *myex.sql* es donde se encuentra la base de datos para realizar las diferentes pruebas.

Para realizar una ejecución en nuestro sistema hay que seguir los siguientes pasos:

Paso 1 Para comenzar ha utilizar nuestra herramienta tenemos que adecuar el entorno. Lo primero es descargar la herramienta DES del CD adjunto a este documento. A continuación, debemos instalar *SICstus Prolog VC9 4.1.2*. Una vez instalado el programa, se crea un acceso directo en el escritorio. Nos situamos en dicho acceso directo y pulsamos el botón derecho, elegimos la opción *propiedades* y nos lleva a una pantalla en la cual hay que poner la ruta donde se encuentra la carpeta "des"descargada anteriormente del CD, dicha ruta hay que especificarla en *Iniciar en*.

Paso 2 Ejecutar SICStus Prolog y escribir la instrucción: *[des]*.

Paso 3 En la misma consola, habilitar la opción de lenguaje SQL, con el comando: */sql*

Paso 4 Para crear la base de datos en el sistema DES se ejecuta el comando: */process nombrearchivo.sql*. Una vez se ha ejecutado esta instrucción, ya se tiene cargada la base de datos en el sistema DES.

Paso 5 Para generar las reglas de cobertura para un consulta V es necesario ejecutar el comando */test_suite* V. V es la vista para la cual se quiere generar las reglas de cobertura. Si se quiere ejecutar con la opción de limites, tal y como se detalla en la Sección 3.2.1, no es necesario poner ningún parámetro adicional. Por el contrario, si se quiere generar las reglas de cobertura sin limites, hay que escribir el siguiente comando: */test_suite* V *noboundaries*. Por ejemplo, si se tiene una vista V cargada en el sistema, y queremos generar sus reglas de cobertura sin limites, el comando seria: */test_suite* V *noboundaries*.

Paso 6 Si todo se ha ejecutado correctamente, en la pantalla del entorno, nos aparecerán todas las reglas. También se genera un fichero de texto en la carpeta "des". El nombre del fichero se llama: *rules_nombrevista_opcionbounadries_opcionesjoin*.

En este fichero aparece toda la información sobre las reglas de cobertura que ha generado dicha vista. Las reglas aparecen agrupadas, en cada grupo se muestra únicamente la parte que cambia de la consulta inicial.

Así por ejemplo para las reglas de cobertura positivas Δ_T solo se muestra la parte del WHERE ya que es la única que cambia. Así para todas y cada una de las reglas de cobertura generadas.

A continuación mostramos dos ejemplos de ejecución.
Ejemplo 1:

Tabla:

CREATE OR REPLACE TABLE T (A int NOT NULL PRIMARY KEY, B int NOT NULL, C int NOT NULL)

Consulta inicial (limites desactivados):

CREATE OR REPLACE VIEW Z AS
SELECT COUNT (T.C), AVG (T.A), T.C, SUM (T.B), AVG (T.B)
FROM T

WHERE T.A >= T.B
AND T.C = 1
GROUP BY T.C, T.A, T.B

Reglas Generadas:

R1. CREATE VIEW Z AS
SELECT COUNT (T.C), AVG (T.A), T.C, SUM (T.B), AVG (T.B)
FROM T
WHERE T.A >= T.B
AND T.C = 1
GROUP BY T.C, T.A, T.B

R2. CREATE VIEW Z AS
SELECT COUNT (T.C), AVG (T.A), T.C, SUM (T.B), AVG (T.B)
FROM T
WHERE NOT (T.A >= T.B)
AND T.C = 1
GROUP BY T.C, T.A, T.B

R3. CREATE VIEW Z AS
SELECT COUNT (T.C), AVG (T.A), T.C, SUM (T.B), AVG (T.B)
FROM T
WHERE NOT (T.A >= T.B)
AND T.C = 1
GROUP BY T.C, T.A, T.B

R4. CREATE VIEW Z AS
SELECT COUNT (T.C), AVG (T.A), T.C, SUM (T.B), AVG (T.B)
FROM T
WHERE T.A >= T.B
AND NOT (T.C = 1)
GROUP BY T.C, T.A, T.B

R5. CREATE VIEW Z AS
SELECT COUNT (T.C), AVG (T.A), T.C, SUM (T.B), AVG (T.B)

```
    FROM T
    WHERE T.A >= T.B
    AND T.C = 1
    GROUP BY T.C, T.A, T.B
    HAVING count (*) > 1
```

R6.
```
    CREATE VIEW Z AS
    SELECT count (*)
    FROM T
    WHERE T.A >= T.B
    AND T.C = 1
    GROUP BY T.A, T.B
    HAVING count (distinct (T.C)) > 1
```

R7.
```
    CREATE VIEW Z AS
    SELECT count (*)
    FROM T
    WHERE T.A >= T.B
    AND T.C = 1
    GROUP BY T.C, T.B
    HAVING count (distinct (T.A)) > 1
```

R8.
```
    CREATE VIEW Z AS
    SELECT count (*)
    FROM T
    WHERE T.A >= T.B
    AND T.C = 1
    GROUP BY T.C, T.A
    HAVING count (distinct (T.B)) > 1
```

R9.
```
    CREATE VIEW Z AS
    SELECT COUNT (T.C), AVG (T.A), T.C, SUM (T.B), AVG (T.B)
    FROM T
    WHERE T.A >= T.B
    AND T.C = 1
    GROUP BY T.C, T.A, T.B
```

HAVING count (distinct (T.C)) > 1 AND count (T.C) > count (distinct (T.C))

R10. CREATE VIEW Z AS
SELECT COUNT (T.C), AVG (T.A), T.C, SUM (T.B), AVG (T.B)
FROM T
WHERE T.A >= T.B
AND T.C = 1
GROUP BY T.C, T.A, T.B
HAVING count (distinct (T.A)) > 1
AND count (T.A) > count (distinct (T.A))

R11. CREATE VIEW Z AS
SELECT COUNT (T.C), AVG (T.A), T.C, SUM (T.B), AVG (T.B)
FROM T
WHERE T.A >= T.B
AND T.C = 1
GROUP BY T.C, T.A, T.B
HAVING count (distinct (T.B)) > 1
AND count (T.B) > count (distinct (T.B))

Ejemplo 2:

Tabla:

CREATE OR REPLACE TABLE S (A int NOT NULL PRIMARY KEY, B int, C int)

Consulta inicial (limites activados):

CREATE OR REPLACE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C >= 30
AND S.C +S.B < S.A
GROUP BY S.A

HAVING count (S.C) > 1

Reglas Generada:

R1. CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C = 31
AND S.C +S.B < S.A
GROUP BY S.A
HAVING count (S.C) > 1

R2. CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C = 30
AND S.C +S.B < S.A
GROUP BY S.A
HAVING count (S.C) > 1

R3. CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C = 29
AND S.C +S.B < S.A
GROUP BY S.A
HAVING count (S.C) > 1

R4. CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.B IS NULL
GROUP BY S.A
HAVING count (S.C) > 1

R5.
```
CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.C IS NULL
GROUP BY S.A
HAVING count (S.C) > 1
```

R6.
```
CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C >= 30
AND S.C +S.B < S.A
GROUP BY S.A
HAVING count (S.C) > 1
```

R7.
```
CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C >= 30
AND NOT (S.C +S.B < S.A)
GROUP BY S.A
HAVING count (S.C) > 1
```

R8.
```
CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.B IS NULL
GROUP BY S.A
HAVING count (S.C) > 1
```

R9.
```
CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.C IS NULL
GROUP BY S.A
HAVING count (S.C) > 1
```

R10.
```
CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C >= 30
AND S.C +S.B < S.A
GROUP BY S.A
HAVING count (S.C) > 1
```

R11.
```
CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C >= 30
AND S.C +S.B < S.A
GROUP BY S.A
HAVING NOT (count (S.C) > 1)
```

R12.
```
CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C >= 30
AND S.C +S.B < S.A
GROUP BY S.A
HAVING S.C IS NULL
```

R13.
```
CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C >= 30
AND S.C +S.B < S.A
GROUP BY S.A
HAVING count (S.C) > 1
AND count (*) > 1
```

R14.
```
CREATE VIEW Z AS
SELECT count (*)
FROM S
```

WHERE S.A + S.B + S.C >= 30
AND S.C +S.B < S.A
HAVING count (S.C) > 1
AND count (distinct (S.A)) > 1

R15. CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C >= 30
AND S.C +S.B < S.A
GROUP BY S.A
HAVING count (S.C) > 1
AND count (distinct (S.C)) > 1
AND count (S.C) > count (distinct (S.C))

R16. CREATE VIEW Z AS
SELECT S.A, AVG(S.C)
FROM S
WHERE S.A + S.B + S.C >= 30
AND S.C +S.B < S.A
GROUP BY S.A
HAVING count (S.C) > 1
AND count (distinct (S.C)) > 1
AND count (*) > count (S.C)

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

En el sector de la informática la parte más importante es la fase de pruebas, sin ella no sería posible la detección, y después corrección de errores en las aplicaciones.

Tras la realización de nuestro proyecto, nos damos cuenta que esta detección de errores y futuras correcciones, no sería posible si no se dispone de una serie de casos de prueba que pueden determinar si existe algún error.

Nuestro proyecto ha consistido en la búsqueda de estos casos de prueba y que se cumplan una serie de reglas de cobertura. Gracias a estas reglas se podrá encontrar mejores casos de prueba, es decir, cuanto menos casos de prueba, es más óptimo.

La elección de estas reglas de cobertura que son generadas, dependen de la semántica de la consulta, tabla o vista en lenguaje SQL.

El desarrollo de esta herramienta debe ser una aportación al campo de las bases de datos relacionales. En definitiva que se establezca una forma de generar casos de prueba en las bases de datos.

Esta aportación se podrá ver reflejado en el trabajo futuro 5.2.

5.2. Trabajo futuro

Después del trabajo realizado hemos identificado una serie de pasos para ampliar este prototipo.

Se ha clasificado en dos partes, la primera parte corresponde con el trabajo inmediato sobre nuestro proyecto, así como la manera de ampliarlo, y

la segunda parte corresponde al trabajo futuro y a la forma de emplear este proyecto.

`TRABAJO INMEDIATO`

1. **Relaciones:**
   Actualmente en nuestra aplicación existen limitaciones para la sintaxis SQL. Para ampliar esta sintaxis es necesario permitir la definición de consultas con más de dos relaciones. En este momento sólo se pueden generar consultas con una tabla, o dos si es de tipo JOIN.

2. **Multiples Relaciones :**
   En este proyecto sólo se contempla las relaciones de dos tablas, de este modo se podría desarrollar, la forma de relacionar mas de dos tablas o vistas,por el conjunto de varios atributos de dichas tablas o vistas.

3. **Operadores Join dentro de la SELECCIÓN:**
   Este tipo de operador es muy común en el mundo de las bases de datos, ya que se puede aplicar una condición a un atributo involucrado en una relación entre tablas,como puede pasar en las consultas de exclusión, las cuales tienen la siguiente forma:

   SELECT *
   FROM T RIGHT OUTER JOIN B on T.c=B.m
   WHERE B.m = null

   De esta forma habría que aumentar el código para incluir este tipo de consultas.

4. **Expresión CASE:**
   Existe una función que actúa como un "$ifthenelse$", esta función es útil para comparar un atributo con uno o varios datos y se mostraría en una consulta de la siguiente manera:

   CASE WHEN p1 THEN v1
   WHEN p2 THEN v2
   ELSE v3
   END CASE

   Para la inclusión en nuestro proyecto sería un proceso costoso, debido a que es un caso totalmente no reflejado en este proyecto, y se debería convertir en disyunciones, es decir cada THEN por un $\vee$ u "OR".

5. **Consultas anidadas:**
   Habitualmente en las bases de datos se utilizan consultas anidadas, las cuales consisten en utilizar consultas dentro de consultas como se muestra a continuación:

   SELECT *
   FROM T
   WHERE m IN (SELECT n FROM B)

   De esta forma, para la utilización de dichas consultas hay que aumentar la sintaxis de SQL permitida en este proyecto, tener en cuenta las condiciones (IN,EXISTS,ALL,SOME,ANY).

`TRABAJO FUTURO`

1. **Optimización de reglas generadas:**
   Nuestro proyecto muestra la salida en un lenguaje poco legible para personas que no estén relacionadas estrechamente con la aplicación, de esta forma como tarea futura se podría transformar la cadena generada que muestra la salida, en lenguaje legible.

2. **Creación de una interfaz gráfica:**
   Como trabajo final se podría utilizar nuestro proyecto para la creación de una interfaz gráfica como medio de interacción directa con el usuario.

Bibliografía

[1] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008.

[2] P. Ammann, J. Offutt, and H. Huang. Coverage criteria for logical expressions. In *Proceedings of the 14th International Symposium on Software Reliability Engineering*, ISSRE '03, pages 99–, Washington, DC, USA, 2003. IEEE Computer Society.

[3] F. Sáenz-Pérez. Datalog Educational System. User's Manual version 3.2, February 2013. Available from `http://des.sourceforge.net/`.

[4] J. Tuya, M. J. Suárez-Cabal, and C. de la Riva. Full predicate coverage for testing sql database queries. *Softw. Test. Verif. Reliab.*, 20:237–288, September 2010.