設計モデルを利用したテスト用データベース自動生成手法

丹野治門[†1]　張暁晶[†1]　星野隆[†1]

本研究は，関係データベース (DB) を用いる業務システムの機能テストを対象として，DB への参照を行う各テストケースに対し，適切な DB 初期状態を自動生成する問題を扱う．既存技術では，複数回の DB 参照や，主キーかつ外部キーの制約，文字列の部分一致検索などを扱った，複雑なビジネスロジックをテストするための適切な DB 初期状態を生成できない問題があった．本研究では上述したような複雑なビジネスロジックを記述できる設計モデルを規定し，その設計モデルに記述された制約条件を段階的に解くことで，テストのための適切な DB 初期状態を自動生成する手法を提案する．本手法を用いると業務システムのより多くのテストケースに対して適切な DB 初期状態を生成可能である．実際の業務システム 2 件を用いた本手法の適用評価では，それぞれ全体の 72%，100% のテストケースに対して，事前状態として適切な DB 初期状態を生成することができ，手法の妥当性を確認できた．

Design Model Based Generation of Initial Database for Testing

Haruto TANNO ,[†1] Xiaojing ZHANG [†1]
and Takashi HOSHINO[†1]

This research focuses on how to automatically generate initial test data of relational database properly for each test case, when testing enterprise systems. Existing approaches cannot generate initial database for complicated business logic such as reading database more than once, or, searching database by partial string matching. To solve these limitations, we propose a method for initial database generation. This method adopts a design model which can describe complicated business logic given above, and from this design model, our method generates appropriate initial database , by step-by-step solving constraints extracted from the design model. The proposed method enables us to automatically generate initial database for a wide range of test cases in testing of enterprise systems. Using two real enterprise systems as case studies, we confirmed that our method properly generate initial database for 72% to 100% of the test case which needs an initial database.

1. はじめに

関係 DB を用いたソフトウェアのテストを行う際には，確認したいソフトウェアの特定の振る舞いを起こすために，テストケースごとに適切な DB 初期状態を用意する必要がある．

DB 初期状態を作成するには 3 つの方法がある．第 1 の方法は，手動でデータを設計し DB 初期状態を用意する方法であり，ユーザがテストケースごとに DB 初期状態を XML で記述して用意する DBUnit はこれに相当する．第 2 の方法は乱数を用いて DB データを自動生成する方法である．この方法では大量の DB レコードを用意することができるので，性能に関する様々な観点を確認する際に有用であり，DBMonster [5]，Visual Studio Database Edition [6] などのツールがある．

しかしながら，第 1 の方法ではゼロから DB 初期状態を作成しなければならないためテスト実施者の負担が大きい．また，第 2 の方法では生成した DB 初期状態が特定のテストケースの事前状態として適切でないときに，DB 初期状態を手動で調整する手間が発生してしまう．

第 2 の方法で問題が起こる具体的な例として，社内研修の対象者である社員を検索する図 1 のような機能のテストを行う場合を考えてみる．この機能は，研修対象である入社 3 年以内であり，かつ在籍中である社員を検索し，検索結果の人数が多いときには 100 人ずつページを区切って表示し，ユーザは年齢と部署名を指定することで絞り込み検索を行うこともできるという仕様を持つ．この仕様に対して,「検索が成功し，検索結果表示画面（図 1(b)）に遷移する」というテストを行う場合,「入社年次が 3 年以下かつ在籍という条件と，ユーザが指定した絞り込み条件という 2 つの条件を満たす社員のレコードを 101 件以上持つ」という制約を満たす DB 初期状態が必要である．

前述した第 2 の方法で用意した DB 初期状態がこのような条件を満たすレコードを偶然持っていればそのままテストを行うことが可能であるが，持たない場合は期待するソフトウェアの振る舞いを起こすことがでないため，テスト実施者はレコードを適切に追加，削除

†1 NTT サイバースペース研究所
NTT Cyber Space Laboratories

することで DB 初期状態を調整するか，第 1 の方法で新たに作成する必要がある．

このように，テストケースごとに DB の主キーの制約，外部キーの制約など各種制約を崩さないように，多数のテーブルを手動で調整する，もしくはゼロから作成するのは効率が悪く，ミスも起こりやすいため，テスト実施者にとって大きな負担である．

テスト実施者の負担を減らすため，第 3 の方法として，ソフトウェアの設計情報から適切な DB 初期状態を自動生成するアプローチがある．Willmor らの研究 [7] では，ユーザがテストケースごとに事前，事後条件として DB 初期状態が満たすべき制約を記述し，その事前，事後条件に合うように，自動で DB レコードを追加，削除することで DB を調整する．また，藤原らの研究 [9] では，事前，事後条件の制約を満たすような DB 初期状態をテストケースごとに生成する．しかし，図 1 のような業務システムのテストを対象にしたとき，[7]，[9] の手法では，次のような問題点がある．

- 事前，事後条件だけでは，DB 参照を複数回行う複雑なビジネスロジックに対応できない．そのため，図 1 においてアクセス権限等を DB に問い合わせて確認してから，条件を満たすレコードの検索処理を行う，といった処理の振る舞いを確認するテストケースに対応できない．
- 業務システムで頻繁に用いられるような制約のうち対応できないものがある．DB スキーマにおいて「主キーかつ外部キーの制約」は，リソース効率を良くするために 1 つのテーブルを 2 つに分割する際によく用いられるが，この制約に対応できない．また，DB 検索の条件として頻繁に用いられる文字列の部分一致比較を扱うこともできない．

本研究では上に挙げる問題点を解決し，業務システムに対する，より多くのテストケースに対して，テストの事前状態として適切な DB 初期状態を自動生成することを目指す．本論文では，DB を用いたアプリケーションで特に多く用いられる参照系アクセスを対象とする．そのため，本論文における適切な DB 初期状態の定義は，「DB スキーマを満たし，かつ DB 参照の検索条件に合う一定件数のレコードを含むもの」となる．

本論文の貢献点は以下の 3 点である．

(1)　モデルベーステストの考え方に基づいた既存のテストケース生成手法の設計モデルを拡張し，DB スキーマや DB 参照の検索条件を記述可能にし，DB 参照を行う業務システムの仕様を記述可能にした．提案手法の設計モデルは，複数回の DB 参照，や，DB 検索条件における文字列の部分一致比較など複雑なビジネスロジックや，DB スキーマにおける主キーかつ外部キーの制約を記述できることを特徴とする．

(2)　提案手法のアルゴリズムでは，設計モデルから抽出した制約を 3 ステップの問題に分割し，DB レコード件数，文字列長という配列の長さに相当する変数の制約を先に SMT で解くことで，テストケースの事前状態として適切な DB 初期状態を求めることができる．

(3)　提案手法を実装したツールを，実際の業務システム 2 件に適用し，生成された DB 初期状態を有識者により目視確認した結果，手法の妥当性を確認した．

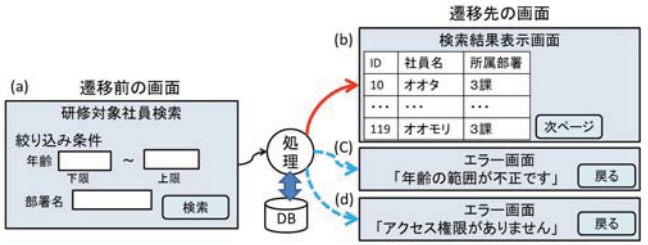

図 1　DB を用いたアプリケーションのテストの例

2. 提案する DB 初期状態生成手法

本研究では，モデルベーステスト [4] の考え方を用い，評価対象システムの設計情報をモデル化した**設計モデル**から，テストケースとそのテストケースの事前状態として適切な DB 初期状態の生成を行う．提案する手法の特徴を以下に示す．

- 提案手法が入力とする設計モデルは，張ら [3] の提案した設計モデルを拡張し，**処理フロー**，**入力定義**，に加えて **DB スキーマ**を記述することができ，処理フローでは，DB 参照を複数回行う複雑なビジネスロジックの振る舞いを明示的に記述できるため，DB 参照を行う業務システムを記述することが可能である，DB 参照を行うときの検索条件には文字列の比較や四則演算などを用いた多様な条件が記述できる．入力定義は処理フローで用いる各種入力（ユーザからの入力，設定ファイルやシステムから得る入力）とその定義域を記述できる．DB スキーマでは，主キー，外部キーといった関係 DB の各種制約を記述可能である．
- 本手法が出力するテストケースは処理フローの各**経路**に関連づけられており，DB 初期状態と入力値から構成される．テストケースを生成するために，設計モデルから「適切な DB 初期状態」とそれとペアをなす入力値を作るための条件を抽出し，それらの条件を制約充足問題に帰着する．この問題を複数ステップからなる部分問題に分割し，そ

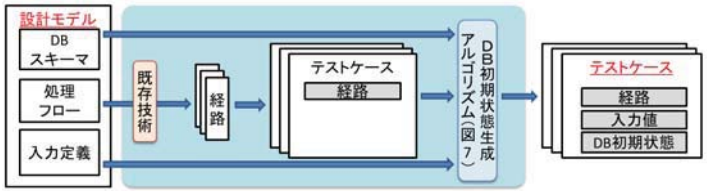

図 2　提案手法の概要

れらを順次，制約ソルバを用いて解く．

これらの特徴により，業務システムにおいて，複雑なビジネスロジックの振る舞いを確認するための様々なテストケースに対して事前状態として適切な DB 初期状態を生成することが可能である．図 2 に手法の全体像を示す．以降，本手法で扱う設計モデル，テストケース，そして DB 初期状態生成アルゴリズムについて詳しく述べる．

2.1　設計モデル

設計モデルは処理フロー，DB スキーマ，入力定義から構成される．具体的な例として，図 1 で紹介した社内研修対象者検索機能の設計モデルを図 3,4,5 に示す．図 3 の処理フローでは，入力値に対して年齢下限が年齢上限以上になっていないかのチェック (図 3(2)) を行った後に，アクセス権限をチェック (図 3(4)) し．アクセス権限があるユーザならば社員検索 (図 3(6)) を行うようになっている．また，図 5 の DB スキーマでは，社員の情報の格納された社員基本情報テーブルや，付随する情報の格納された社員付随情報テーブルなどが定義されており，これらのテーブルには主キーかつ外部キーといった制約が記述されている．

2.2　テストケース

本手法が出力するテストケースは経路，DB 初期状態，入力値の 3 者で構成されている．具体的な例として，図 1 において，検索結果表示画面へ遷移する経路（図 3 における (1), (2), (4), (6), (7) の経路）に対応するテストケースを図 6 に示す．図 6 のテストケースの DB 初期状態は，社員基本情報テーブル，管理者テーブルといった直接参照されるテーブルの他に，部署テーブル，社員付随情報テーブルといった DB のリレーションを満たすために必要なテーブルのレコードも存在する．

2.3　DB 初期状態と入力値の生成アルゴリズム

DB 初期状態と入力値の生成アルゴリズムについて説明する．本手法では図 7 のように，設計モデルから抽出した制約（以後，**制約群**と呼ぶ）と変数（以後，**変数群**と呼ぶ）を，変

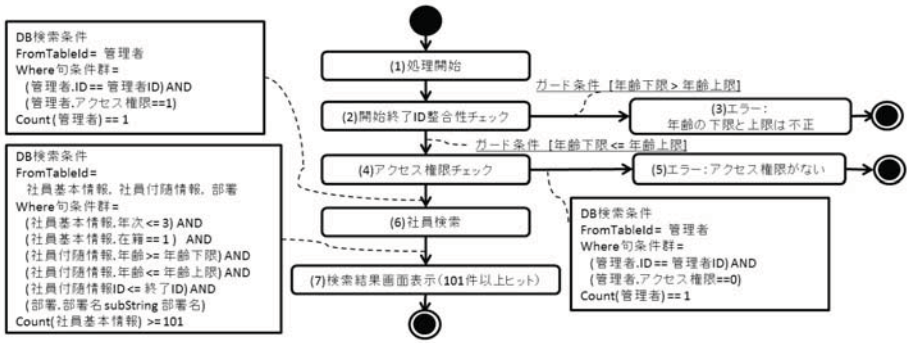

図 3　設計モデル：処理フローの例

VariableId	型	定義域
年齢下限	IntegerType	MinValue = 18, MaxValue = 120
年齢上限	IntegerType	MinValue = 18, MaxValue = 120
部署名	StringType	MinLength = 0, MaxLength = 16
管理者ID	IntegerType	MinValue = 50, MaxValue = 60

図 4　設計モデル：入力定義の例

数群に対する連立制約式とみなし，以下の方針に従って解く．

- 整数に対する制約は連立不等式として扱い既存の制約ソルバである Satisfiability Modulo Theories の Arithmetic Theory (以降，単に SMT と呼ぶ) を用いて解を求める．
- テーブルはレコードの配列として扱う．配列構造は一般的には可変長であるため，そのまま連立不等式の解法へと帰着して SMT で解くことはできない [8]．そこで，まずレコード数を配列の長さに関する制約として表現し，個々のレコードが保持するデータよりも先に SMT を用いてレコード数を決定する．次に，レコード数を新たな制約として加え，再度 SMT を用いることでレコードの保持するデータを決定する．
- 文字列は文字の配列として扱う．テーブルと同様に，文字列長は配列の長さに関する制約として表現し，文字列の内容よりも先に SMT を用いて文字列長を決定する．文字列を構成する文字は数が多いため，ひとつひとつについて制約を SMT で記述すると，解を効率よく求められないので，SMT を用いずに後述する独自の手法で決定する．

TableId = 社員基本情報

ColumnId	型	定義域	カラムの種類
ID	IntegerType	Min = 0, Max = 2000	PKFK 社員付随情報.ID
名前	StringType,	MinLength = 0, MaxLength = 16	Normal
年次	IntegerType	Min = 1, Max = 60	Normal
在籍	IntegerType	Min=0, Max=1	Normal
部署ID	IntegerType,	Min = 0, Max = 1000	FK 部署.ID

TableId = 社員付随情報

ColumnId	型	定義域	カラムの種類
ID	IntegerType,	Min = 0, Max = 2000	PK
年齢	IntegerType,	Min = 18, Max = 120	Normal

TableId = 管理者

ColumnId	型と定義域		カラムの種類
ID	IntegerType,	Min = 0, Max = 60	PK
社員情報 アクセス権限	IntegerType,	Min = 0, Max = 1	

TableId = 部署

ColumnId	型	定義域	カラムの種類
ID	IntegerType	Min = 0, Max = 1000	PK
部署名	StringType,	MinLength = 0, MaxLength = 16	Normal

図 5　設計モデル：DB スキーマの例

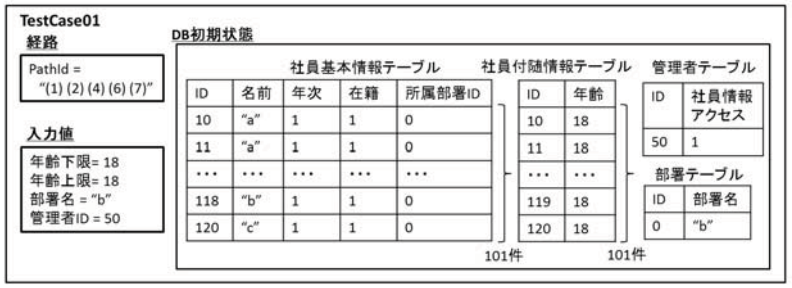

図 6　テストケースの例

これらの方針により，本手法では図 7 で示すように，Step1 で SMT を用いてレコード件数変数の値を求め，Step2 で SMT を用いて文字列長変数と整数変数の値を求め，Step3 で文字変数の値を求める，という段階的な解法で DB 初期状態と入力値を求める．以降の節で，図 7 における Step0,1,2,3 についてそれぞれ詳しく説明する．

2.3.1　Step0・設計モデルから変数群，制約群を抽出

Step0 では，図 7 中の手順 (1)〜(6) により，設計モデルから制約群と変数群を抽出する．

DB の参照整合性制約を満たすために，外部キーの参照先のレコードが少なくとも 1 つは必要になる．そのため，手順 (6) では経路中の DB 検索条件の検索対象であるテーブルに

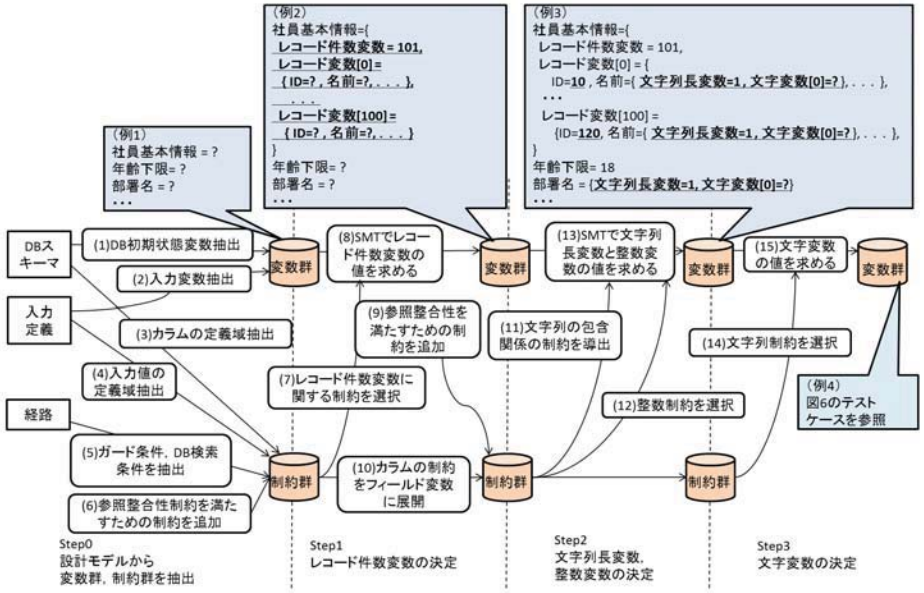

図 7　DB 初期状態生成アルゴリズムの全体像

対応するテーブル変数 t_{src} から外部キー，主キーの参照関係で辿ることのできるテーブル変数 t_{target} に対して，「$Count(t_{target}) >= 1$」という制約を追加する．また，主キーかつ外部キーの制約があった場合，すなわち，外部キーの参照元のカラムが主キーの制約も持っていた場合は，最低，参照元のレコードの数だけ外部キーの参照先のレコードが必要になるため，外部キーの参照元，参照先のテーブル変数 $t1, t2$ に対し，「$Count(t1) <= Count(t2)$ レコード件数変数」という制約を追加する．

2.3.2　Step1・レコード件数変数の値を決定

Step1 では，図 7 中の手順 (7)〜(10) により，レコード件数変数の値を決定する．

手順 (8) では，DB レコード件数変数に関する制約を連立不等式とみなし SMT に与え解を求める．

手順 (9) では，主キーである各フィールドが確実に一意になるようにするため，n 件のレコード変数をもつテーブル変数の，主キーであるカラムに対応するフィールド変数 $pk_1, ..., pk_n$ に対して，IntegerType の場合は「$pk_1 < pk_2$」，...,「$pk_{n-1} < pk_n$」，StringType の場

合は「pk_1 $subString$ c_1」，...,「pk_n $subString$ c_n」($c_1, ..., c_n$ は互いに異なる定数文字列)，という制約を新たに生成し制約群に追加する．ここで，A $subString$ B は文字列 A は文字列 B を部分文字列として含むことを表す制約である．また，外部キーによる参照元のフィールド変数と参照先のフィールド変数の値が等しくなるようにするため，外部キーであるカラムに対応するフィールド変数 fk に対して，参照先のテーブル変数のフィールド変数 pk を任意に選び（参照元が主キーかつ外部キーのカラムに対応していた場合は，それぞれ別々のフィールド変数 pk を選ぶ）IntegerType ならば「$fk == pk$」，StringType ならば「fk $equalsString$ pk」という制約を新たに生成して制約群に追加する．ここで，A $equalsString$ B は，文字列 A と文字列 B が等しいこと表す制約である．

手順 (10) では，DB 検索条件，DB スキーマにおけるカラムの定義域から抽出したカラム c への制約「c $Operator$ x」を，カラム c に対応する各フィールド変数 $f_1, ...f_n$ への制約「f_1 $Operator$ x」，...,「f_n $Operator$ x」として展開し，展開した制約を制約群へと追加する．

Step1 が終了するとテーブル変数のレコード変数の値が決まるため，変数群は図 7 の（例 1）の状態から（例 2）のような入力変数とテーブル変数の各フィールド変数が未決定の状態となる．

2.3.3　Step2・文字列長変数，整数変数の値を決定

Step2 では，図 7 中の手順 (11)〜(13) により，文字列長変数，整数変数の値を決定する．

部分文字列の文字列長の和が，それらを含む文字列の文字列長を超えないようにするため，手順 (11) では，ある文字列変数 s が部分文字列として $s_1, ..., s_n$ を含んでいたとき文字列変数同士の包含関係に基づき「$Length(s) \geq Length(s_1) + ... + Length(s_n)$」という整数制約を導出し，制約群へ加える．文字列同士 x, y が等しい場合は $Length(x) == Length(y)$ となる．

Step2 が終了すると，全ての文字列長変数，整数変数の値が決まるため，変数群は図 7 の（例 2）の状態から（例 3）に状態に変わり，未決定なのは文字列変数を構成する文字変数のみとなる．

2.3.4　Step3・文字変数の値を決定

Step3 では，図 7 中の手順 (14)〜(15) により，文字変数の値を決定する．

手順 (15) では，ある文字列 $s0$ と別の文字列 $s_1, ..., s_n$ の相関関係があるとき，部分一致や等価という制約であれば，それぞれの位置を揃えてから，対応する文字が同じになるよう文字変数の値を決定し，部分一致や等価の否定の場合は，一カ所文字が異なるよう文字変数の値を決定する．

Step3 が終了すると，制約群が空になり変数群の全ての値が決定し，図 7 の（例 3）の状態から（例 4）に状態に変わり，図 6 のようにテストケースの事前状態として適切な DB 初期状態と入力値が求まる．

3. 評　　価

提案手法に対する評価観点及び評価結果を示し，考察を述べる．

3.1 観　　点

本研究では，既存研究の問題点を解決することによって，業務システムにおけるより多くのテストケースに対してテストケースの事前状態として適切な DB 初期状態を自動生成することを目指している．そのため，評価観点は以下の 2 点である．

- DB 参照を行うテストケースのうちどれだけに対して DB 初期状態を生成できたか．
- 生成した DB 初期状態はテストケースの事前状態として適切であるか．

3.2 方　　法

前章で解説した提案手法に基づいて DB 初期状態生成ツールを作成した．整数制約を解く際（2.3.2 節，2.3.3 節）に用いる SMT に基づく制約ソルバとして，Z3 [10] を用いた．

本手法は DB への参照系アクセスを行うテストケースを対象としているため，2 つの業務システム（以後，A，B と呼ぶことにする）のうち参照系アクセスが多い機能をそれぞれ選び（図 1）評価に用いた．本手法のプロトタイプツールでは XML 形式の設計モデルを入力とする．そのため，A，B の設計書から既存技術 [2] を用いて処理フローの経路を抽出した後，手作業で XML 形式の設計モデルを記述し，更に A，B の設計書から DB スキーマと処理フローの経路中の検索条件に相当する設計情報を抽出して設計モデルに書き加えた．この設計モデルを入力として DB 初期状態生成ツールに DB 初期状態を生成させ，それが正確に経路を通るもの，すなわちテストケースの事前状態として適切なものになっているかを確認した．なお，作業は全て筆者と異なる業務システム開発経験者 1 名が行った．DB 初期状態生成ツールを動作させた環境は CPU が Intel Core i7 3.0GHz，メモリは 6GB，OS は Windows Vista Ultimate SP2 である．

3.3 結　　果

A，B の評価結果を表 1 に示す．また，本手法が特徴とする機能の使用頻度を表 2 に示す．参照系の DB アクセスを行っているテストケースに対し A では 100% ，B では 72% に対し DB 初期状態を生成することができた．DB 初期状態生成の実行時間は，1 テストケー

表 1　評価結果

			A(4 テーブル，11 画面)		B (12 テーブル，12 画面)	
提案手法の対象	DB 初期状態と入力値を要する	適用可	74	74/74(100%)	83	60/83(72%)
		適用不可		0/74(0%)		23/83(28%)
	入力値のみ要する	適用可	5	5/5	36	36/36
		適用不可		0/5		0/36
提案手法の対象外	異常系		149	142/149	260	256/260
	DB 更新			7/149		4/260
テストケース総数			228		379	
生成時間			0.1s/テストケース		0.1s/テストケース	

表 2　本手法が特徴とする機能の使用頻度

	A	B
総数	74	60
DB 参照を 2 回以上行う	39/74(47%)	39/60(65%)
主キーかつ外部キーのカラムをもつ DB テーブルへアクセスする	28/74(38%)	0/60(0%)
文字列部分一致比較を行う	12/74(16%)	10/60(17%)

スあたり 0.1 秒程度であった．また，複数回の DB 参照アクセス，DB スキーマにおける主キーかつ外部キーの制約，文字列の部分一致を行う検索条件といった本手法の特徴も，表 2 のように出現頻度から鑑みると有効であった．

3.4 考　　察

評価に用いた A，B のテストケースは，ユーザのアクションによってシステムがある画面から次の画面へ遷移するという単発のインタラクションが正しく動作するかという観点で作成されていた．そのため，最初にアクセス権限等のチェックを行ってから次に本来の目的である検索を行うというパターンが多く，このとき参照するのはそれぞれ別のテーブルであったため，複数回 DB アクセスに対応している本手法で多くのテストケースに対し適切な DB 初期状態を生成することができた．

B では，複合キー制約を要するテーブルがあり，主キーかつ外部キーの制約と併用し，「顧客情報と，顧客ごとの購買記録」といったような 1 対多の関係の記述に用いられており，本手法を適用することができなかった．このような制約への対応は今後の課題といえる．

各テストケースは，その目的によって必要な量の DB レコードを用意するだけで十分である．例えば，ログイン処理，アクセス権限のチェックなら 1 件，検索結果ならば 10 数件程度のレコードが事前状態としてあればよいというパターンが多かった．本手法では大量にレコードを生成するツール [5], [6] と異なり，必要のないテーブルのレコードは生成せずテストケースごとに必要最低限のレコードのみを生成しているため，DB 初期状態の生成時間は実用的な範囲内に収まったと考えられる．

4. 結　　論

DB スキーマを満たし，かつ DB 参照の検索条件に対して，条件に合う件数のレコードを返す，テストの事前状態として適切な DB 初期状態を自動生成する手法を提案した．本手法では，既存のテストケース生成手法で用いられる設計モデル記法を拡張し，DB 参照を含む業務システムを記述できるように記述能力を向上した．そして，その設計モデルから，経路を網羅するためのテストケースの事前状態として適切な DB 初期状態を自動生成するアルゴリズムを考案した．また，提案手法をツールとして実装し，実際の業務システム 2 件に適用することにより手法の妥当性を確認した．今後は，より多くのテストケースにおいて DB 初期状態が生成できるように，設計モデルで扱える制約を増やし，適用範囲を拡大していきたい．

参　考　文　献

1) 情報処理推進機構ソフトウェア・エンジニアリング・センター. ソフトウェア開発データ白書 2009. 日経 BP 出版センター, September 2009.
2) 丹野治門, 張暁晶, 星野隆. 結合テストにおけるテスト項目自動生成手法の提案と評価. 信学技報, 第 110 巻 of *SS2010-34*, pp. 37–42, 2010.
3) 張暁晶, 星野隆. 設計モデルを用いたテスト項目抽出とテストデータ生成手法. 信学技報, 第 109 巻 of *SS2009-7*, pp. 37–42, 2009.
4) Arilo C.Dias Neto, Rajesh Subramanyan, Marlon Vieira, and GuilhermeH. Travassos. A survey on model-based testing approaches: a systematic review. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies*, pp. 31–36, 2007.
5) DBMonster. http://dbmonster.kernelpanic.pl/.
6) Visual studio database edition. http://www.microsoft.com/japan/msdn/vstudio/.
7) David Willmor and SuzanneM. Embury. An intensional approach to the specification of test cases for database applications. In *Proceedings of the 28th international conference on Software engineering*, pp. 102–111. ACM, 2006.
8) J. Edvardsson. A survey on automatic test data generation. In *Proceedings of the Second Conference on Computer Science and Engineering in Linkoping*, pp. 21–28 ECSEL, 1999.
9) 藤原翔一朗, 宗像一樹, 片山朝子, 前田芳晴, 大木憲二, 上原忠弘, 山本里枝子. Smt solver を利用した web アプリケーション用テストデータの生成. 情報処理学会創立 50 周年記念（第 72 回）全国大会講演論文集, 2010.
10) Z3. http://research.microsoft.com/en-us/um/redmond/projects/z3/.