DEIM Forum 2012 E9-3

個人/共有データを格納するための ロールベースアクセス制御機構付き Web データベースの提案

仙波　雅也† 　鎌田十三郎†

† 神戸大学大学院システム情報学研究科
〒 657-8501 神戸市灘区六甲台町 1-1
E-mail: †{senba,kamada}@cs26.scitec.kobe-u.ac.jp

あらまし　近年，Web 上に簡単にデータを置けるような Web サービスが広く利用されており，中には共有機能を持つものもある．我々は，様々な種類のデータを Web 上において共有しつつ，マッシュアップなどに手軽に利用できるような環境を目指している．本研究では，行単位のアクセス制御機構を備えた Web データベースサービスを提案し，実現する．提案システムでは，ユーザがテーブルを定義でき，同時に，テーブルのレコードに課すべきアクセス制御のルールを記述する．ルールには読込・生成・消去の 3 種類があり，簡単に記述できるようシンプルなモデルとなっている．また本論文では，いくつかの応用事例を通して，本システムの利便性や解決すべき課題などの検討を行う．
キーワード　アクセス制御，マッシュアップ

Web Data Store Service with Role-Based Access Control for Sharing Structured Data

Masaya SENBA† and Tomio KAMADA†

† Graduate School of System Infomatics, Kobe University
1-1 Rokkodai-cho, Nada-ku, Kobe 657-8501 Japan
E-mail: †{senba,kamada}@cs26.scitec.kobe-u.ac.jp

Abstract　Currently, many web services allow us to put data on the web and share the data among groups. This paper proposes a data store service with a role-based access control system. The service prepares Web API and can be used for mashups. The user can define tables with their access control rules. According to the table's rule, our access contol system determines whether users can access each row. Our case study shows some applications that store data in tables and require a fine-grained access control by row.
Key words　access control, mashup

1. はじめに

現在，個人のデータを簡単に Web 上に置けるサービスが広く利用されている．中には，Google カレンダーのように，グループ内でデータを共有できるものもある．また，Web API を通じてデータを外部に提供しているサービスも多くあり，それらを組み合わせて新しいサービスとするマッシュアップが盛んに行われている．

本研究は，より多くの種類のデータを適切なアクセス制限のもと Web 上に配置し，Web API を通じて各種用途に利用できるような環境を目指したものであり，シンプルなアクセス制御機構を備えた Web データベースサービスの提案・実現を行う．

例えば，各種業務に対する実態調査アンケートや学術論文に対する評論など，外部公開することなくグループ内で情報の共有を行いたいケースは多い．その際，個々のアンケート集計アプリや，SNS 上での評論の発表という形ではなく，個々の用途に応じた構造を持ち，Web API を用いた検索を通じて様々な用途に利用できればと考えたためである．

本研究では，データの構造としてテーブル形式を想定しており，ユーザは独自のテーブル構造の定義と，行単位のアクセス制御ルールの指定を行う．データの用途と構造が決まれば，アクセス制限の仕方も決まる場合が多く，似たようなルールをレコードごとに設定するのは非効率と考えたためである．アクセス制御ルールは「ロールを表す属性」と「クエリ発行ユーザとの関係」のみで表現する，シンプルなものである．アンケートを例にとると,「アンケートの実施者」や「アンケートの回答組

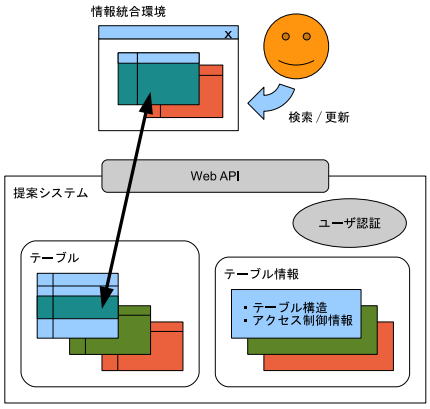

図 1　システムの構成要素

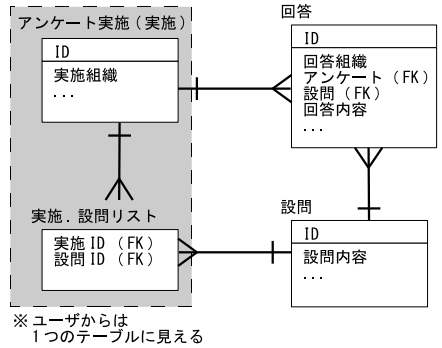

図 2　アンケート事例における ER 図

織」なる属性があり，回答データの書き込みは，クエリ発行者が回答組織に属していれば実行可能であり，読み取りは，クエリ発行者が回答組織に属しているか実施者に一致していれば実行可能であるといったものである．

提案 Web データベースサービスは，各単一テーブルに対する CRUD 系クエリをサポートする Web API を介してアクセスする．システムがクエリを受け取ると，定義されたアクセス制御ルールに基づいてクエリを拡張し，ユーザがアクセス可能な領域に対してのみ検索や更新を行う．

本論文の構成は，以下の通りである．まず 2. 節にて，提案システムの概観を述べた後，アンケート事例を通して実現したいアクセス制御についての方針を立てる．次に 3. 節で，方針に基づいて定めたデータモデルとアクセス制御モデルについて概説する．4. 節で応用事例や今後の課題について検討し，5. 節で簡単な評価実験を行う．最後に，6. 節で関連研究を紹介し，7. 節でまとめとする．

2.　提案システム

2.1　システム概観

図 1 は，提案システムの概観を表したものである．データベースには，複数のテーブルが格納され，各テーブルの構造やアクセス制御設定などのテーブル情報はシステムによって管理される．システムは，ユーザ認証・管理機構と Web API も備え，ユーザがデータアクセスする際は，まず認証を行ってシステムにログインしてから，Web API を通じたデータのやり取りを行う．このため，格納された情報は各種マッシュアップなどに利用可能であり，我々のグループの別プロジェクトとして，検索・更新・マッシュアップなどを手軽に行うための情報統合環境の開発も進めている [8].

テーブルの行をエントリと呼び，システムから与えられる ID を用いた識別・直接アクセスが可能である．本システムにおけるユーザは，システムが提供するユーザテーブルのエントリとして表現される．グループはユーザの集合であり，これもまた，グループテーブルのエントリとして表現される．ユーザとグループの所属関係は，システムによって管理される．また，特殊なグループとして任意のユーザが所属する ANY と，誰も所属しない EMPTY が存在する．テーブルには，その要素エントリが持つべき属性が決められている．属性の型には，整数などの基本型や別エントリへの参照型がある．特に，ユーザテーブルのエントリへの参照型である属性をユーザ型属性，グループテーブルのエントリへの参照型である属性をグループ型属性と呼び，後述するアクセス制御モデルにおいて中核となる．

2.2　利用事例とデザインの方針

本節では，事例を通してデータ構造表現と実現したいアクセス制御について簡単な紹介を行い，アクセス制御モデルのデザイン方針について述べる．例として，少々複雑なアクセス制御が要求されるアンケートの事例を用いる．我々は，この程度の複雑さについては，簡単に記述できるようなアクセス制御モデルのデザインを目指す．

この事例の登場人物として，まず，アンケートの実施と回収を行う実施者が存在する．また，回答して欲しい対象として，部署や研究室などの組織があり，アンケートへの回答は，各組織に属するユーザのうちの誰かが回答する形にしたいとする．ここでの組織は，本システムにおけるグループで表現できる．

図 2 は，本事例で利用するデータの構造と関係を ER 図にまとめたものである．テーブルは，今後，各種アンケートを格納し，検索などを通じて再利用する目的で構成されている．実施者は，設問を取りまとめるためのアンケート実施エントリを作成する．実施エントリには，アンケートを実施する組織を表す属性や，設問エントリリストの属性などを設けてある（後述するが，本システムでは属性型としてリスト型を準備している）．設問エントリは，設問内容を保持するためのものであり，実施テーブルとは別の設問テーブルを設けている．テーブルを分離することで，個々の設問や回答の再利用性を高めることができる．実施者は，設問テーブルから聞きたいことを選び，あるいは新しく設問エントリを作成し，それらの参照を設問リストに持たせる．各回答者は，設問に対する回答エントリを作成する．回答エントリには，どの組織を代表しての回答かを示す回答組織属性や，どのアンケートのどの設問への回答かといった属性がある．特に，どのアンケートへの回答か，どの設問への回答かという属性は，それぞれ実施エントリ，設問エントリへの参照で表している．

この例の回答エントリに対して，ユーザが実現したいであろうアクセス制御の事例について検討する．まず，回答者に対し

・読込

- 回答組織に属しているか対象アンケートの実施者に一致すれば可

・生成

- 回答組織に属していれば可

・消去

- 回答組織に属していれば可

図 3　アンケート事例におけるアクセス制御ルール

て，各々が所属する組織の回答エントリを読み込むことと書き込むことを可能にしたいとする．また，回答エントリを作る際に，回答組織属性の値に不適切なグループを設定することによる”なりすまし”を防止したい．一方で，実施者に対して，自身が実施したアンケートに対する回答エントリ全てを読み込み可能にして，結果の集計ができるようにしたい．

これらをまとめると，図 3 となる．読込がエントリの読み込みに関するルール，生成と消去がエントリの書き込みに関するルールである．生成は，不当な値でエントリを生成されるのを防ぐためのルールであり，この例では，新しい回答組織属性の値として，自身が属しているグループのみ設定可能という制約が課される．一方で消去は，今ある値を消してしまってもよいかを判定するためのルールである．エントリを更新するときは，消去と生成の両方の判定が行われる．

次に，回答組織や実施者といった属性を一種のロールとみなし，この「ロールを表す属性」と各種クエリ発行ユーザとの「関係（一致／所属）」だけで表現する形にすれば，シンプルになると考えた．ロールを表す属性には，基本的に当該テーブルが持つ属性を用いるが，この例では回答エントリに実施者の情報がないため，参照している実施エントリの実施者属性を用いた判断が必要となる．そこで，当該テーブルから参照しているテーブルの属性も利用可能とし，例えば「アンケート. 実施者」のように参照属性と参照先の属性を並べて書くようにすることで，ルール表記の簡潔さを維持する．ここで，クエリ実行者が，参照している実施エントリ自体の読み込み権限を持っていない場合は，実施エントリの属性値を使った条件である「実施者に一致」は不成立とすべきだと判断した．

実施エントリと設問エントリについては，いずれも回答エントリで出てきた形式で記述できると考えたため，割愛する．

最後に，デザインの特徴を以下にまとめる．

- アクセス制御モデルは，読込・生成・消去の 3 種類のアクセス制御ルールからなる
- アクセス制御ルールは，ロールとクエリ発行ユーザとの関係を使って表現される
- ロールは，テーブルが持つ属性で表現される．また，当該テーブルが参照しているテーブルの属性も使用可能

3. モ デ ル

3.1 データモデル

本システムではエントリ単位でデータを取り扱い，アクセス制御もエントリ単位で可否判定を行う．エントリは属性と値の組の集合であり，テーブルごとに決められた構造をとる．属性

```
 1: <table name="回答"> <!-- 回答テーブルの定義 -->
 2:   <permission>
 3:     <read>
 4:       <belongsTo attr="回答組織"/>
 5:       <equals attr="アンケート. 実施者"/>
 6:     </read>
 7:     <create>
 8:       <belongsTo attr="回答組織"/>
 9:     </create>
10:     <delete>
11:       <belongsTo attr="回答組織"/>
12:     </delete>
13:   </permission>
14:   <!-- 構造定義部は省略 -->
15:     ...
16: </table>
```

図 4　アンケート事例におけるアクセス制御ルールの記述例

には，ユーザが定義する属性と，システムが付与する属性（以下，システム属性）の 2 種類がある．システム属性には，ID・エントリ作成者・最終更新者・最終更新時刻の 4 種類がある．システム属性の値はシステムが自動的に設定し，ユーザが直接設定・更新することはできない．属性の型には，整数や文字列などの基本的な型と，別エントリへの参照型，複数の属性をまとめて 1 つの型とするレコード型（例えば，緯度経度）がある．加えて，型が指定された要素の並びを保持するためのリスト型を整備中である．

また，本システムは各種 Web サービスとのマッシュアップ用途への利用も想定しているため，参照型属性としてシステム外の Web サービスが提供するリソースへの参照も扱う．ただし，実際にマッシュアップする際データ結合の仕方は本論文のテーマではないため，割愛する．

3.2 アクセス制御モデル

本システムのアクセス制御モデルは，read/create/delete の 3 種類のアクセス制御ルールからなり，それぞれ読込・生成・消去に対応している．いずれも，クエリ発行ユーザが満たすべき条件を列挙し，いずれかの条件が満たされれば，クエリ実行が許可される．更新クエリは，delete と create の双方，すなわち，今ある値を上書きして消してもよいかの判定と，更新後の新しい値が妥当かの判定の両方を行う．また，列挙された条件にかかわらず，エントリ作成者は常に read と delete が許可される．ユーザは，テーブルを定義するときに，これらのルールを併せて定義する．定義されたルールは，そのテーブルのエントリ全てに適用される．

各条件は，ロールを表す属性と，ロールとクエリ発行ユーザとの関係（一致する，所属する）の組で表現する．ロールを表す属性として，実施者属性や回答組織属性といった，ユーザ型またはグループ型の属性が選択できる．また，当該テーブルが別テーブルへの参照型属性を持つ場合，参照先テーブルのユーザ／グループ型属性を使うこともできる．ただし，参照先エントリの属性値を使ってアクセス可能かを判定する場合は，クエリ発行ユーザが参照先エントリを読み込み可能である必要がある．参照先エントリが存在しない場合，アクセスは拒否される．

2.2 節で述べた事例におけるアクセス制御を，このモデルを用いて記述すると，図 4 のようになる．permission 要素が，アクセス制御ルールの定義部であり，その下に read/create/delete に対応した要素が並ぶ．その子要素として，クエリ発行ユーザ

との関係を表す equals または belongsTo 要素があり，attr 属性の値としてロールを表す属性を指定する．read を例にとると，4 行目が「回答組織に所属していれば」に，5 行目が「アンケートの実施者に一致していれば」に対応している．また，参照先テーブルのユーザ／グループ型属性を使う場合は，5 行目の attr 属性値のように，実施テーブルへの参照である「アンケート」属性と，実施テーブルが持つ「実施者」属性とをピリオドでつなぐ．

4. 応用検討

4.1 利用形態

提案システムは，Web サービスとして公開されており，CRUD 系の Web API を介してアクセスすることができる．本節では，提案システムがどのような形で利用できるかを概説する．

まず，専用の Web アプリケーションとして，2.2 節で述べたアンケートを扱うアプリケーションの事例を考える．クライアントは，JavaScript を用いた RIA として作成し，提案システム上のアンケート実施・設問・回答テーブルに Web API を介してアクセスする．ユーザへのアンケート一覧の提示は，実施テーブルの検索によって行われる．アンケートが大量に存在する場合は，タイトルや時期などの条件を指定して検索する．あるアンケートの設問一覧の取得は，まずアンケート実施エントリから設問エントリの ID のリストを取得し，次にその ID を用いて設問テーブルを検索することでできる．ある設問に対する過去の回答を表示して参考にしてもらうという機能をつけたい場合は，設問 ID を条件にして回答テーブルを検索すればよい．当該ユーザがアクセス可能な回答エントリを表示することができる．このように，提供されている API を用いて，アクセス保護が施されたデータを取り扱うようなアプリケーションを作ることができる．

先の例では，提案 Web サービスしか用いていなかったが，他の Web サービスの情報と連携して，マッシュアップアプリケーションを作ることができる．前述したように，外部の Web サービスが提供するリソースへの参照を持たせることも可能なので，例えばホテル検索サービスの各リソースに対して，コメントをつけたり，ブックマークを作成したりといったことができる．

さらに，本研究の別プロジェクトとして情報統合環境の開発があり，任意の Web サービスを柔軟にマッシュアップできることを目指している．本システムは，情報統合環境が用いる Web サービスの 1 つとして，アクセス保護がなされたデータを生成・更新・検索するストレージの機能を担っている．

4.2 モデルの有効性

提案システムでは，各種用途に応じた構造を持ったデータを，それぞれのデータに応じたアクセス制御をつけた形で登録・公開・編集することを可能としている．本節では，提案システムが有効に機能する事例について，簡単な検討を行う．

まず，2.2 節でのアンケートの事例において，ユーザが作成したデータの再利用性について考える．当初の目的は，アンケート実施者が回答データを回収することである．データを構造化して保存しておくことにより，以降に同様のアンケートが行われる場合に，過去にどのような回答を行っていたのかを各設問について調べることができる．

また，回答内容を積極的に公開したい場合，回答テーブルを定義する際に「公開対象属性」を設け，読込ルールに「公開対象に所属」を追加しておけば，回答内容を回答者間で共有し，参考にしながら回答するようなこともできる．回答者が，自分の回答を他人に見せたくない場合は，回答エントリ作成時に公開対象属性に EMPTY を指定すればよい．

一方で，現状のアクセス制御では不満が残るケースも存在する．現在のモデルでは，エントリが読み込み可能と判定されると，全ての属性の値が可視となる．しかし，例えば匿名のアンケートを実施したい場合は，回答内容に関する属性は実施者に見せてよくても，回答者に関する属性（エントリ作成者，回答組織など）は見せるべきではないだろう．あるいは，アンケートに回答期限を設け，期限が過ぎたのちは，回答エントリの生成・更新を禁止したいようなケースも考えられる．現在のアクセス制御ルールのシンプルさや可読性を保ちつつ，拡張していく方法について，今後検討していきたい．

次に，メールや，グループ内で閉じたチャットのような目的で使う場合を考える．このときのデータ構造として，送信者・受信者・返信先・タイトル・本文などの属性が挙げられる．返信先は，このエントリが収められるテーブルへの参照型属性であり，メッセージの起点（返信先がない）の場合は NULL を指定する．アクセス制御ルールもシンプルであり，

- **読込**
  - 送信者に一致，または受信者に所属していれば可
- **生成**
  - 送信者に一致していれば可
- **消去**
  - 送信者に一致していれば可

のようになるだろう．一方で，例えば新着メールを確認するために，クライアントが定期的に発行するクエリ全てに対して，内部的に行われる結合演算まで実行していては，サーバ側の負荷が大きくなるという懸念がある．そこで，Publish/Subscribe システムのような非同期メッセージを扱う機構を導入することを検討している．pub/sub システムに取得したいビューのクエリを登録しておき，エントリの生成や更新の際に，対応するビューを更新し，検索クエリに対しては，最新のビューを返すといった対策が必要になるであろうと考えている．

5. 評価実験

本システムのアクセス制御は，他の細粒度アクセス制御機構と同じく，ユーザが発行したクエリにアクセス制御のための条件を付加することで実現している．本節では，このアクセス制御ルールの付加によって，クエリ実行のオーバーヘッドがどの程度発生するのかを，簡単な実験を通して評価する．

実験は，本システム内でも使用している MySQL 上で行った．検索対象となるテーブル T と，T から参照しているテーブル R の 2 つを用意し，それぞれ 10 万件ずつエントリを生成した．各テーブルの構造を，表 1 および表 2 に示す．インデックスは，

表 1　テーブル T の構造

属性	型	備考
ID	binary(22)	主キー
data	int	
A0	binary(22)	ユーザ型
A1	binary(22)	グループ型
ref	binary(22)	R への参照

表 2　テーブル R の構造

属性	型	備考
ID	binary(22)	主キー
B0	binary(22)	ユーザ型
B1	binary(22)	グループ型

表 3　テーブル T に与えるアクセス制御ルール

ルール	条件
1	なし
2	A0 に一致
3	A1 に所属
4	A0 に一致，または A1 に所属
5	ref.B0 に一致
6	ref.B1 に所属
7	ref.B0 に一致，または ref.B1 に所属

表 4　計 測 結 果

ルール	実行時間 [μs]	結果件数	時間/件数
1	6380	5014	1.27
2	516	10	51.60
3	3839	98	39.17
4	4317	108	39.97
5	25507	10	2550.70
6	27188	98	277.43
7	52816	108	489.04

テーブル T の ref 属性以外の全ての属性に対して作成した．

使用する検索クエリは

```
select T.ID
from   T
where  T.data between N and N+500
       (N は 0〜9499 でランダム)
```

であり，これを表 3 にあるようなルールで拡張し，発行する．例えば，ルール 5 なら，以下のように拡張される．

```
select T.ID
from   T join R on T.ref = R.ID
where  R.B0 = 'XXX...XX'
       && T.data between N and N+500
```

各条件の選択率は，data 属性の範囲指定が 5%,「A0 に一致」および「ref.B0 に一致」が 0.2%,「A1 に所属」および「ref.B1 に所属」が 2% とした．ルールごとにクエリを 200 回ずつ実行し，実行に要した時間の平均を表 4 にまとめた．

実行時間を見てみると，ルール 5〜7 が多くの時間を要している．これはクエリを拡張する際にテーブル結合を用いているためであり，コストが増えること自体は妥当といえる．参照先テーブルの属性を使うルールが定義されたら，その時点で結合済みのビューを作るなど，コスト軽減の手法を検討していくことが今後の課題となる．

```
 1: CREATE OR REPLACE FUNCTION check_read_answer(
 2:     schema_p IN VARCHAR2,
 3:     table_p  IN VARCHAR2)
 4:   RETURN VARCHAR2
 5:   AS
 6:     pred VARCHAR2 (400);
 7:   BEGIN
 8:     pred :=
 9:      'EXISTS(SELECT FROM membership_table '
10:       + 'WHERE user_id=SYS_CONTEXT("context", "user_id")'
11:       + ' AND group_id=回答. 回答組織)'
12:       + 'OR アンケート. 実施者=SYS_CONTEXT("context", "user_id")';
13:   RETURN pred;
14: END;
```

図 5　ポリシー関数の作成

```
1: BEGIN
2:   DBMS_RLS.ADD_POLICY(
3:     object_schema   => '...',
4:     object_name     => ' 回答-アンケートビュー',
5:     policy_name     => '...',
6:     policy_function => 'check_read_answer',
7:     statement_types => 'SELECT');
8: END;
```

図 6　セキュリティ・ポリシーの作成

6. 関 連 研 究

6.1 アクセス制御の動向

近年，細粒度のアクセス制御の重要性が高まっている [2]．アプリケーションレベルでアクセス制御を行うのではなく，Oracle Virtual Private Database [6]（以下，VPD）のように，データベースシステムに対する付加機能としてアクセス制御機構を提供するものも登場している．現在，クラウドコンピューティングの浸透などに伴い，データベースが複数のアプリケーションから利用される状況も進んでおり，データベース階層におけるアクセス制御実現の重要度は増している．

Oracle VPD は，行および列レベルでのアクセス制御を行う機構であり，セキュリティ・ポリシーが設定されたテーブルに対して発行される SQL 文に対し，動的な WHERE 句を追加することで，ユーザがアクセスできる領域を制限する．また，アプリケーションコンテキストを使用することで，ユーザのセッション情報（ユーザ ID など）に基づいたアクセス制限を記述できる．追加する WHERE 句として自由な条件を書くことができるため，高い柔軟性を持つ一方で，本論文で述べていた参照先テーブルの属性を用いたアクセス制限などを記述する場合，条件式が複雑なため可読性が落ちる恐れがある．

例えば，2.2 節のアンケート事例におけるアクセス制御を Oracle VPD を用いて実現するには，以下のことを行う．まず，予めアプリケーション・コンテキストとログイン・トリガーを作成しておく．アプリケーション・コンテキストにはユーザ ID などをキャッシュしておくことができ，ログイン・トリガーによって，ユーザのログイン時にアプリケーション・コンテキストの内容を設定する．また，グループとユーザの所属関係を管理するための membership_table も準備しておく．このテーブルには，グループとユーザのペアが保持されている．次に，回答テーブルの読み込み可否判定には，アンケート実施テーブルも用いるため，両者を結合したビューを作成する．最後に，ポリシー関数の作成（図 5）とポリシー関数の適用（図 6）を行う．図 5 では，SQL 文に追加する WHERE 句を定義する．9-11 行

目が「回答組織に所属」に，12 行目が「実施者に一致」に相当する．また，SYS_CONTEXT(”context”, ”user_id”) は，予め作成したアプリケーション・コンテキスト context からユーザ ID を取得する．以上により，回答テーブルのエントリに対する読み込みアクセス制御が実現される．実現したいアクセス制御に応じて，同様のことを UPDATE や INSERT に対しても記述する．

アクセス制御記述については，複雑な権限設定や多様なアクセスを記述できる XACML [7] の標準化なども進んでいる．我々のアクセス制御では扱えられていないような事例も多いため，今後，検討を行っていきたい．

6.2 クラウドにおけるデータの共有

現在，クラウドサービスとして，Google App Engine [3] や Amazon Web Service [1] などが提供されており，データベース機能や Web アプリケーション実行環境が提供されている．ただし，アプリケーションレベルのユーザに対するアクセス制御は，アプリケーション階層で実現するのが一般的である．

エンドユーザレベルで，汎用的構造化文書を共有する仕組みとしては，Google Docs [4] が存在する．Docs 中の表形式ドキュメントは Google Fusion Tables [5] を用いることで，結合演算を施したり，各種ビューア機能を通して表示させたりする事ができる．一方で，データ共有はドキュメント単位で行うものであり，テーブル形式のドキュメントであっても，行や列レベルでのアクセス制御ができるわけではない．

また Google Docs では，スプレッドシートとフォーム機能を用いて，簡単なアンケートを作成できる．フォームから送信された回答は，自動的にシートに挿入されていくため，結果の集計が容易である．シートの共有設定に関係なく，フォームの URL を知っていれば誰でも回答することができ，シートを閲覧できないユーザでも，集計結果をグラフ化した概要を閲覧可能である．しかし，シートに挿入された回答内容の編集権限はシートの共有設定によるため，シートの閲覧権限を持たない回答者は，自分の回答内容を見ることや修正することができない．

7. ま と め

本論文では，シンプルなアクセス制御機構を備えた Web データベースサービスの提案と実現を行った．提案システムでは，テーブル構造を定義する際に，読込・生成・消去の 3 種類のアクセス制御ルールを同時に設定する．単一テーブルに対する CRUD 系 Web API を提供し，クエリを受け取ると，定義されたルールに基づいてクエリを拡張する．

また，本システムを用いた応用事例の検討と，アクセス制御ルールが加わることによるオーバーヘッドの評価を行い，機能面・性能面における課題を考察した．

今後の課題として，4. 節で挙げたアクセス制御ルールの拡張などの他にも，スケーラビリティの確保が挙げられる．本システムでは，アクセス制御ルールに応じてクエリが拡張されるが，拡張後も単一テーブル内に収まっているケースであれば，スケーラビリティは期待できる．一方で，拡張後にクエリが複数のテーブルにまたがってしまうケースでは，アクセス制御ルールと参照関係に応じて割り振るノードを決定するなどといった対策が必要であり，現在検討中である．

謝　　辞

本研究の一部は，科学研究費補助金基盤研究 (C) 22500091 の補助を受けたものである．

文　　献

[1] Amazon Web Services, http://aws.amazon.com/.
[2] Elisa Bertino, Gabriel Ghinita, and Ashish Kamra, Access Control for Databases: Concepts and Systems, *Found. Trends databases*, Vol. 3, pp. 1–148, (2011).
[3] Google App Engine, http://code.google.com/appengine/.
[4] Google Docs, http://docs.google.com/.
[5] Google Fusion Tables, http://www.google.com/fusiontables/.
[6] Oracle database 11g virtual private database, http://www.oracle.com/technology/deploy/security/database-security/virtual-privatedatabase/index.html.
[7] Oasis consortium, extensible access control markup language (xacml) committee specification, version 1.1, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.
[8] 熊谷良夫, 仙波雅也, 鎌田十三郎, データの編集・登録が可能なエンドユーザ向け情報統合環境, *Proc. of DEIM2012*, (2012).