# 大規模システムへのOracle FGACによるアクセス制御の適用事例

## Application of Access Control in Large Systems using Oracle FGAC

菅　井　綾　子

**要　約**　日本ユニシスが開発に携わった大規模システムのデータマートにおいて，データレコード単位での細やかなアクセス制御が必要とされた．そのアクセス制御方式として，Oracle Database 10gの機能であるFine Grained Access Control（以下，FGAC）を採用した．対象が大規模システムのため，アクセス権限情報の一元化やFGAC制御用ファンクションの共通化など，本システム独自の工夫を図ることで，開発工数の低減に繋げた．

他システムにFGACを適用する際には，今回の開発で見えてきた留意点を踏まえ，組織構成や情報の重要度の観点からの対象データの分析，レスポンスへの影響の評価・検討，FGACの存在の見え難さへの対策を考えていくことが重要であると考える．

**Abstract**　In a data mart of a large system that Nihon Unisys was involved in development, the fine-grained, data record-based access control has been required. We adopted Fine Grained Access Control (hereinafter called, FGAC), a feature of Oracle Database 10g, as the access control system. Focusing on a large-scale application, we have succeeded in improving the development cost (man-hours) by exercising our ingenuity unique to the targeted system such as the unification of access authority information and the standardization of FGAC control functions, etc.

When applying FGAC to other systems, it is important to consider to think of points to remember clarified in this development, analyze targeted data, assess and review effects, and respond to the invisibility of the FGAG existence from a viewpoint of the importance of organizational structure and information,

## 1. はじめに

2000年代初頭から，社会的にセキュリティへの関心が高まっている．従来は社外に対してのセキュリティのみを重視する傾向にあったが，情報漏洩防止や個人情報保護の観点から，同一企業内におけるセキュリティ対策にも注目が集まっている．しかし一方で，企業の持つデータを有効活用したいというニーズが高まりを見せていることも事実である．セキュリティとデータの活用，この二つを両立させることは，現代のセキュリティ対策における課題だと言える．

このような動きを受け，2005～2008年に日本ユニシスが開発に携わったA社の社内業務システムにおいても，データレコード単位での細かなアクセス制御を実現したいという要求があった．この要求は，このシステムのデータ分析機能においても同様である．ユーザーによる多角的な情報分析のためにシステムのほぼ全てのデータを扱うデータ分析機能では，このようなアクセス制御の実現のため，Oracle社のデータベース管理システムに搭載されたFine Grained Access Control（以下，FGAC）機能を活用して，行単位のアクセス制御であるRow Level Security（以下，RLS）を実装した．

本稿では，この適用事例を通じて，FGACを用いてRLSを実現する上で，今回採用した技

術的工夫，およびその工夫についての考察を述べる．2 章では，このシステムにおける RLS の必要性について述べる．3 章で FGAC による RLS 実現の一般的な方法を紹介し，4 章では，今回の対象システムの特性（大規模なシステム，複雑なアクセス制御）を考慮した上で，開発時の工数削減や保守時の運用性向上のために施した工夫について述べる．5 章以降では，今回のアプローチを考察する．5 章では FGAC 以外の RLS 実現方法との比較，6 章では今回の工夫のメリットおよびデメリットとその対策，7 章では FGAC によるアクセス制御における一般的なメリットと留意点について述べる．8 章において，今回の開発で得た，類似システムへ適用可能なノウハウを紹介する．

## 2. 対象システムにおける RLS の必要性

今回対象としたのは，社内の基幹業務を担うシステムである．このシステムにおける RLS の必要性について，以下に述べる．

### 2.1 システムの特徴

#### 2.1.1 機密情報の取り扱い

業務の中には，顧客を対象とするものも存在する．そのため，顧客に関するさまざまな情報を取り扱うこととなる．例えば，顧客情報や案件情報，交渉履歴情報といった種類の情報を扱う．名前から容易に推測できるとおり，個人情報や機密情報にあたるものが非常に多いことが特徴と言える．

#### 2.1.2 部署による顧客の種類・管轄

顧客を対象とする業務の場合，部署や業務上の役割によって，担当する顧客が違うということも特徴として挙げられる．大抵の場合，部署ごとに担当する顧客の種類が決められている．例えば，同一支社内においても，個人顧客のみを担当する部署と，法人顧客のみを担当する部署が存在する．また，本社からは全支社全事業所の顧客情報を参照する必要があるのに対し，支社からは自支社配下の事業所管内の，事業所からは自事業所管内の顧客のみの情報を扱うことになる．

#### 2.1.3 システムの規模

今回開発したシステムのデータ分析機能では，オンライン業務や他システムから連携したデータを蓄積し，このデータを使用してユーザーが情報分析を行う．この機能が扱うデータの範囲については，「本システムで扱う全データは，ユーザーが自由に抽出し，業務で活用できること」といった業務要件があった．即ち，データ分析機能は，システムで扱うほぼ全てのデータを対象とするのである．具体的な数字を次に示す．

1） マートテーブル数：約 200 エンティティ
2） エンドユーザー数：約 1,400 ユーザー
3） Oracle ユーザー数：約 40 ユーザー（部署ごとに設定）

テーブル数やユーザー数が多く，比較的大規模なシステムであると言える．なお，マートテーブルとは，エンドユーザーがデータ抽出の対象とするデータマートのテーブルのことである．管理上，Oracle ユーザーは部署ごとに設定し，複数のエンドユーザーで共通して使用す

る運用となっている．

### 2. 2　RLS の必要性

前述したように，今回対象としたシステムでは，個人情報や機密情報といった非常にセンシティブな情報を扱う必要があるため，同一社内とは言え，全データを全ユーザーに公開することはできない．例えば，法人顧客のデータは，関係部署ではない個人顧客を担当する部署のユーザーからは見えないようにしたい．このため，オンライン業務やデータ分析機能を含むシステム全体に，共通のセキュリティ要件を満たすアクセス制御を施す必要があった．

2. 1 節の対象システムの特徴から，ユーザーの所属部署によるアクセス制御が要件として挙げられた．顧客情報，案件情報といったエンティティ単位でのアクセス制御ならば，スキーマレベルのアクセス権操作により比較的容易に実現可能なのだが，一つのエンティティの中にも各部署にとって閲覧可のデータと不可のデータが混在しているため，レコードに対して部署単位にアクセス制御を行わなくてはならない．また，一言で部署単位でのアクセス制御と言っても，データの特性によって，データの重要性や契約規模による制御であったりデータ作成者の所属部署に依存する制御であったりと，要求されるアクセス制御のレベルが異なる．この要件を満たすため，システム全体に，実に細やかな行単位でのアクセス制御，即ち，RLS の実現が必要不可欠となった．

## 3.　FGAC 機能による一般的な RLS の仕組み

今回の開発では，Oracle Database 10g（以下，OracleDB）と，データ分析のための OLAP ツール[*1] として Oracle Business Inteligence Discoverer（以下，Discoverer）を採用し，Discoverer ワークブックを通してデータベース（以下，DB）のデータにアクセスするようにした．しかし，OracleDB の通常のスキーマ定義と Discoverer の機能を組み合わせても，エンティティ単位のアクセス可否設定に留まり，レコード単位での緻密なアクセス制御は不可能である．

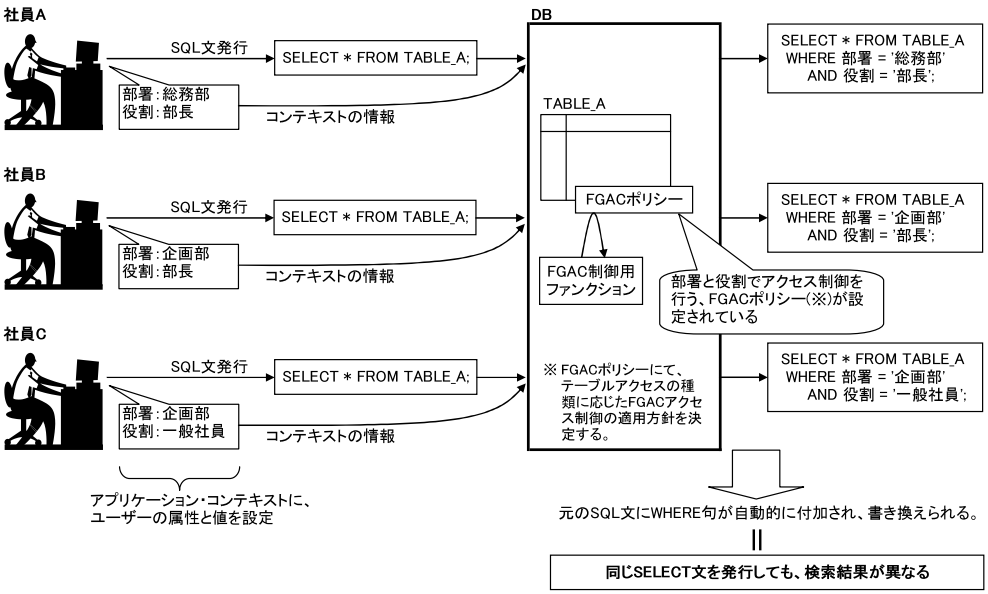


**図 1　FGAC によるアクセス制御の仕組み**

そのため，OracleDB の機能である FGAC を採用する運びとなった．ここで，FGAC を使用してアクセス制御を行う仕組みについて，簡単に紹介する．

FGAC によるアクセス制御では，各エンティティへのアクセス要求が発生するごとに，アクセス先のテーブルやビューに設定された FGAC ポリシーが適用される．この FGAC ポリシーはエンティティごとに設定する必要があり，それぞれ FGAC ポリシー適用時に呼び出す FGAC 制御用ファンクションを指定する．FGAC ポリシーにより起動された FGAC 制御用ファンクションは，アプリケーション・コンテキストに設定されたユーザー情報を基に，実行要求のあった SQL 文に対して WHERE 句を自動的に付加することで，ユーザーに意識させずに RLS を実現する（図 1）．この FGAC 機能により，RLS を実装していくこととした．

## 4. 大規模システムへの適用事例

ここで，今回開発したシステムのデータ分析機能における FGAC の適用事例を紹介する．アクセス制御の大まかな流れは，前述した一般的な FGAC の適用方法と同様であるが，今回の開発では，テーブル数やユーザー数が多く，大規模なシステムであるという特徴を踏まえ，開発工数低減と保守時点での運用性を考慮した工夫をいくつか施した．対象システムにおける FGAC 適用の具体的なイメージは，図 2 のとおりである．

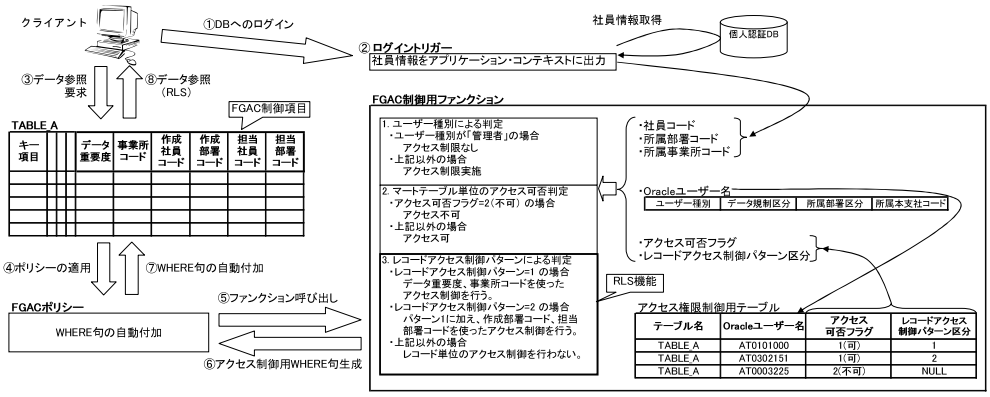


**図 2　システムへの FGAC 適用イメージ**

### 4. 1　アプリケーション・コンテキストの設定

本開発では，社員個人を指定してアクセス制御を施す必要があるため，エンドユーザー個人を識別する情報として「社員コード」を使用した．DB に対してログイントリガーを設定し，ログイン要求（図 2 ①）をきっかけに個人認証 DB から社員コードなどの社員情報を取得後，アプリケーション・コンテキストに格納した（図 2 ②）．

### 4. 2　Oracle ユーザー名の活用

Oracle ユーザー名が FGAC 制御の中で容易に利用できることから，Oracle ユーザー名に命名規則を設け，「ユーザー種別」，「データ規制区分」，「所属部署区分」，「所属本支社コード」といった，アクセス制御に必要な情報を持たせた．図 2 では FGAC 制御用ファンクション内に示した Oracle ユーザー名がこれにあたる．これにより，ユーザーが DB にアクセスするだ

けで，自動的にこれらの情報を取得することができる．また，Oracle ユーザー名は個人認証 DB にて管理されているため，ユーザーが Oracle ユーザー名の存在を意識する必要はない．

### 4. 3　アクセス権限情報の一元化

今回の開発における特徴的な工夫の一つが，アクセス権限情報の一元化である．FGAC によるアクセス制御では，各エンティティにアクセス権限情報を持たせる方法が一般的ではある．だが今回は，約 200 のマートテーブルと約 40 の Oracle ユーザーからなる複雑なアクセス権限の情報を，各エンティティに個別に持たせるのではなく，アクセス権限制御用テーブルを作成し，そこにアクセス権限の情報を集約することで一括して管理する方法を考案した．図 2 では，FGAC 制御用ファンクションで使用するアクセス権限制御用テーブルがこれにあたる．マートテーブルと Oracle ユーザーごとに，「アクセス可否フラグ」，「レコードアクセス制御パターン区分」を保持する構造とし，運用開始後に発生し得る変更にも柔軟に対応できるようにした．「アクセス可否フラグ」には，ユーザーごとに各エンティティへのアクセス可否を設定する．「レコードアクセス制御パターン区分」には，後述のレコードアクセス制御パターンの分類を設定する．

開発時には Microsoft Excel 上に一覧表の形でアクセス権限情報をまとめており，この一覧表から Windows の OLE 機能[*2] を使って直接 DB に出力することでアクセス権限制御用テーブルを作成している．よって，保守時にも Excel 表をメンテナンスすることで簡単に対応可能となる．

### 4. 4　FGAC 制御用ファンクションの共通化

#### 4. 4. 1　FGAC 制御用ファンクション共通化のイメージ

FGAC 制御用ファンクションは，FGAC ポリシーごと，即ち，エンティティごとに作成するのが一般的である．つまり，対象エンティティの数が多ければ多いほど，FGAC 制御用ファンクションの設計・開発・保守にかかる工数が増大することになる．前述したように，今回のシステムは大規模であるため，この規模のマートテーブルとユーザー単位で，単純にエンテ

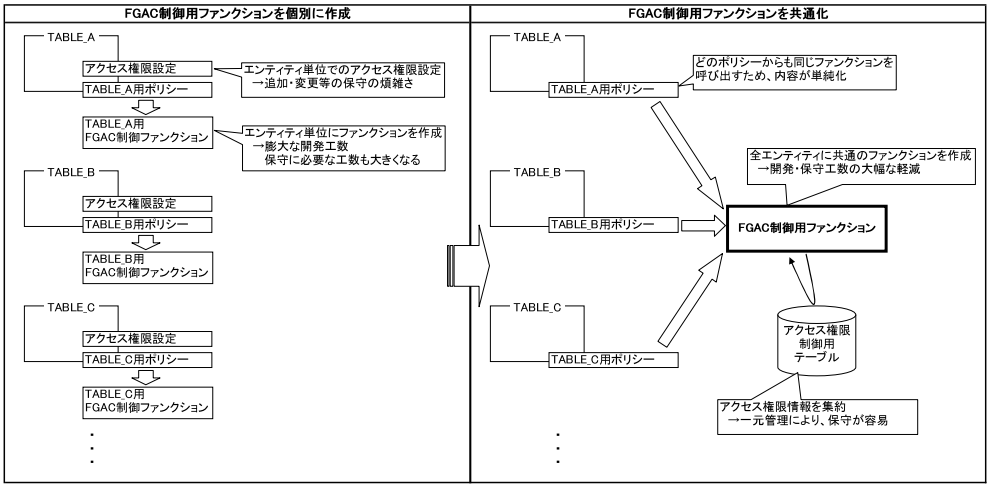


**図 3　FGAC 制御用ファンクション共通化のイメージ**

ィティごとにFGAC制御用ファンクションを実装すると，膨大な工数を必要とする．

この問題を解決するため，今回の開発においては，エンティティごとにFGAC制御用ファンクションを作成するのではなく，全エンティティに共通した一つのファンクションのみを作成することで，開発・保守工数の低減を図った．結果，PL/SQLで170ステップ程度の比較的小さなファンクションに共通化することができた．FGAC制御用ファンクション共通化のイメージは図3のようになる．

#### 4.4.2　各エンティティへのFGAC制御項目の追加

FGAC制御用ファンクションの共通化が可能となった背景の一つとして，FGAC制御項目の考案により，アクセス制御のレベルが異なるエンティティに対しても，同一ロジックでWHERE句を生成することが可能となったことが挙げられる．

FGACによるRLSは，SQL文へのWHERE句の自動付加によって実現するということは，先に述べたとおりである．テーブルに既存の項目を使用してWHERE句を組み立てる場合，同一ファンクション内の同一ロジックにおいては，全エンティティで同じ意味合いの項目は必ず同じ英字項目名でテーブルに持たせる必要があった．例えば，自動付加のWHERE句で使用したい「部署名」という項目があったとする．これが，テーブルAでは「BUSHO」，テーブルBでは「BUSHO_MEI」となっていると，FGACで付加すべきWHERE句に相違が出てしまい，WHERE句を生成するファンクションが共通化できないということになる．FGAC制御用ファンクションを共通化し，同一のロジックで各エンティティを扱うためには，各エンティティに共通した項目が必要となるのである．本開発で使用するのは，オンライン業務での使用を優先して設計されたエンティティであるため，全エンティティに項目名統一の徹底を求めることは難しい．また，オンライン業務で使用する項目のみでは，FGAC制御に不足する情報もあるため，FGACによるアクセス制御に使用する項目は，FGAC制御項目としてマートテーブルの各エンティティに付加する運用とした．

FGAC制御項目は「データ重要度」,「事業所コード」,「担当部署コード」など，FGAC制御用ファンクションによるWHERE句生成の判定に必要な項目で，これらの項目の値は日次バッチ処理によって設定される．

#### 4.4.3　レコードアクセス制御パターンの分類

FGAC制御項目に加え，FGAC制御用ファンクションの共通化に重要な役目を果たしているのが，レコードアクセス制御パターンである．2.2節で述べたように，今回の開発では，データの特性によってレコード単位にレベルの異なるアクセス制御を実現する必要があった．ファンクションの共通化を考え，アクセス制御対象となるデータを整理した結果，大きく2種類にパターン化することができた．このパターンの分類は，4.3節で述べたアクセス権限制御用テーブルに，レコードアクセス制御パターン区分として保持している．

区分1のパターンは，ユーザーの所属部署に依存するパターンで，データの重要性によりアクセスできるユーザーを大別した上で，所属によるデータの絞り込みを行う．このため，FGAC制御項目の内，主要項目（データ重要度，事業所コード）のみを使用する．

区分2は，レコード作成者個人に依存するパターンで，区分1のパターンで行うユーザーの所属部署での絞り込みに加え，当該レコードの関係者にしかデータを扱えなくする．FGAC

制御項目の主要項目に加え，「作成部署コード」や「担当部署コード」を使用することで，「ユーザーがデータの作成者と同じ部署に所属する場合のみアクセスを許可する」といった制御を行う．

なお，どちらのパターンでも FGAC 制御項目を使用していることからわかるように，このパターン分類も各エンティティに共通の FGAC 制御項目を設けたことで可能となった．

## 5. 今回の開発における FGAC 採用の妥当性

本開発における FGAC の適用方法については，これまで述べたとおりである．しかし，FGAC を使用せずにアクセス制御を行う方法が皆無というわけではない．そこで，FGAC を採用しなかった場合の RLS 実装方法として，以下の 1）および 2）を考察し，これを踏まえて今回の開発における FGAC 採用の妥当性を検証する．

1）　ビューまたはワークブックの使用による RLS

テーブル自体に対するアクセスは認めず，代わりに，部署ごとに閲覧可能なデータのみを表示するビューまたは Discoverer ワークブックを作成し，それぞれの部署に適したアクセス権限を設定することで実現する．

この方法には，スキーマのデータ定義や Discoverer の機能のみで比較的容易に実装できるというメリットはある．だが，デメリットとして，一つのテーブルに対し，部署の数だけ専用のビューやワークブックを作成するため，管理対象が膨大な数になってしまうという問題がある．また，ビューやワークブックごとに，データの閲覧を制限するための複雑な条件を設定する必要があるため，開発・保守は容易ではない．

さらに，Discoverer ワークブックの使用による RLS の場合には，「エンドユーザー自身がテーブル項目に紐付いたアイテムを指定して Discoverer ワークブックを自由に作成したい」というニーズに対し，実際には，自由にワークブックを作成することができるのは全データの閲覧を許可された特権ユーザーのみであるため，「ユーザーによる自由な抽出」という業務要件が満たせているとは到底言えない．

2）　オンライン業務における RLS

同じシステムの内，オンライン業務においても同様の要件でアクセス制御を行っているが，こちらは FGAC を使用せず，プログラムの作り込みによって実装している．ログイン時に部署と職責を取得して 18 種類の権限パターンに分類し，その権限パターンを各画面に引き渡す．各画面は，権限パターンに応じて，テーブルアクセス時に条件を設定することで，アクセス権限のないレコードをユーザーに扱わせないようにするのである．

部署や職責単位に条件を指定してアクセス制御を行うところは FGAC 使用時と同じように見える．だが，設定する条件は，画面ごとに 18 種の権限パターンに対して独自で生成している．オンライン業務では，各サブシステムで扱う画面やテーブルが予め決まっており，画面から発行する SQL は常に固定であるために，画面ごとに WHERE 句をコーディングする方法が成り立っているのである．この方法は，使用するテーブルや SQL が固定的でなく，WHERE 句生成に柔軟性が必要なデータ分析機能においては，非現実的と言える．

1)および2)に述べたいずれの方法を用いても，今回の開発の要件を満たすアクセス制御を実装することは難しい．したがって，本開発における RLS 実装への FGAC の採用は妥当であったと考えられる．

## 6. 今回の開発における FGAC 適用のメリット・デメリット

本開発においては，対象システムが大規模であるため，開発・保守工数の膨張を抑える工夫が必要とされた．そして，実際にその工夫を交えた実装を施したことは，既に述べたとおりである．ここで，今回の開発における FGAC 適用のメリット，デメリットとその対策について整理する．

### 6.1 今回の開発における FGAC 適用方法のメリット

#### 6.1.1 FGAC 制御用ファンクション共通化による開発工数低減

4.4節で紹介したように，今回の開発では全エンティティに対して一つの共通した FGAC 制御用ファンクションを作成した．一般的な方法ではエンティティの数に比例して増大する FGAC 制御用ファンクションの数を最小限に抑えたことにより，開発工数の大幅な低減が可能となった．

#### 6.1.2 アクセス権限情報の一元化による保守工数低減

システム運用開始後に保守作業が発生した場合の開発者による対応を，表1にまとめている．特徴として，今回の開発ではアクセス権限情報をアクセス権限制御用テーブルで集中管理しているため，部署の統廃合やアクセス権限関連の変更の場合は，このアクセス権限制御用テーブルのメンテナンスのみで対応することができる．これに対し，集中管理しない場合は個々のエンティティへのメンテナンスが必要となる．このため，集中管理しない場合と比較して，保守作業の工数が大幅に軽減されると考える．

また，万が一，データ規制区分や所属部署区分などの Oracle ユーザー名に埋め込んだ情報の変更が発生し，FGAC 制御用ファンクションの改修が必要になった場合にも，メンテナン

表1 想定される主な保守要件とメンテナンス対象

| | | メンテナンス対象 | | | |
|---|---|---|---|---|---|
| | | FGACポリシー | FGAC制御用ファンクション | アクセス権限制御用テーブル | Discoverer データ定義 |
| 保守要件 | データ種類の追加（テーブル追加） | ○ | − | ○ | ○ |
| | データ種類の削除（テーブル削除） | − | − | ○ | ○ |
| | データの定義内容変更（テーブル項目変更） | − | − | − | ○ |
| | データのアクセス制御レベル変更（テーブルのレコードアクセス制御パターンの変更） | − | − | ○ | − |
| | 部署の追加（ユーザー追加） | − | − | ○ | − |
| | 部署統廃合（ユーザーの削除） | − | − | ○ | − |
| | 権限の変更（ユーザー権限変更） | − | − | ○ | − |
| | データ規制区分の追加／削除 | − | ○ | − | − |
| | 所属部署区分の追加／削除 | − | ○ | − | − |

（凡例） ○:影響有　−:影響なし

ス対象が一つであるため，保守工数が低減できると考える．

### 6. 2　今回の開発における FGAC 適用方法のデメリットとその対策

#### 6. 2. 1　アクセス権限情報の一元化によるレスポンス低下

アクセス権限情報を一つのテーブルに集約して一元管理する方法のデメリットとして，ユーザーから DB へのアクセス要求が発生する度に，FGAC 制御用ファンクションからアクセス権限制御用テーブルへの SELECT 文が発行されるため，アクセス権限情報を各エンティティに直接保持する場合に比べ，レスポンスの低下が懸念される．

今回の対象システムにおけるアクセス権限制御用テーブルの検索では，二つのキー項目（テーブル名，Oracle ユーザー名）を指定して 1 件を選択する方法をとっている．アクセス権限制御用テーブルのレコードは，テーブル単位かつ Oracle ユーザー単位に作成されており，2. 1. 3 項の数字からレコード数を概算すると，約 200 エンティティ×約 40 ユーザー＝約 8,000 レコードとなる．そして，この約 8,000 レコードから，キー項目のインデックスを使用して 1 件を選択するため，選択性は高いと言える．また，A 社のシステム運用の考え方から，レコード数が将来的に著しく増加することは考え難い．このため，実際にレスポンスに与える影響は小さいと判断し，この方法を採用した．

#### 6. 2. 2　FGAC 制御項目の使用によるレスポンス低下

本開発における工夫の一つが，FGAC 制御用ファンクションの共通化である．そして，この共通化のために，4. 4. 2 項で述べたように FGAC 制御項目を各エンティティに付加した．多くの場合，FGAC 制御用ファンクションでの WHERE 句生成に使用する項目はエンティティごとに固定的である．今回の開発では，各エンティティに設けた FGAC 制御項目がこれにあたり，今回実装したアクセス制御方式ではほぼ必ず使用される．そのため，WHERE 句を付加された SQL が FGAC 制御項目を参照することによりレスポンスに与える影響が懸念される．

この場合の対策としては，FGAC 制御用項目にインデックスをつけて検索効率を上げることが考えられる．ただし，取得するデータ量やレコード位置の物理的なばらつきによっては，効果が薄かったり，逆にパフォーマンスの低下を招いたりする場合があるため，インデックスの使用にあたっては，その有効性を吟味する必要がある．

#### 6. 2. 3　FGAC 制御項目設定のためのバッチ処理の必要性

4. 4. 2 項で述べたとおり，FGAC 制御項目の値は日次バッチ処理によって設定される．つまり，FGAC 制御項目に値をセットするためのバッチアプリケーションの開発が必要となったのである．

FGAC 制御項目値設定のために，新規にバッチアプリケーションを作成する方法も考えられた．だが，今回の開発においては，既存のバッチ処理を利用する方法を採用した．データマートをオンライン業務用の DB とは別に構築し，バッチアプリケーションにてデータマートにデータを蓄積していく方法は，通常よく見られる方式である．今回対象としたシステムにおいても同様に，日次バッチ処理にてデータマートへの連携を行っているため，この既存のバッチアプリケーションに追加する形で，FGAC 制御項目を設定するバッチ処理を実装した．これにより，バッチアプリケーション開発にかかる工数を必要最小限に抑えることができた．

## 7. FGAC によるアクセス制御の一般的なメリットと留意点

これまで述べてきたように，FGAC は大変便利な機能であり，今回の開発への採用は妥当であったと言える．しかし，実際に使用していく上で留意すべき点も存在した．ここで，今回の開発での要件に依存しない，FGAC によるアクセス制御の一般的なメリットと留意点について考察する．

### 7.1 FGAC によるアクセス制御の一般的なメリット

#### 7.1.1 ユーザーに意識させない堅固なアクセス制御

ユーザーのデータ要求に対して自動的に RLS が為されることで，そのユーザーのアクセスを許可しないデータの存在自体を隠蔽することが可能となる．また，DB のデータ自体へのアクセスが制限されるため，アプリケーションを迂回した不正アクセスの対策としても有効である．つまり，ユーザーにアクセス制御の存在を意識させずに，堅固なアクセス制御が実現できると考える．

#### 7.1.2 ビジネスロジックへの柔軟な対応

FGAC によるアクセス制御は，アプリケーション・コンテキストと FGAC 制御用ファンクションにより実現される．アプリケーション・コンテキストには独自の属性を設定できるため，対象システムごとのアクセス制御要件に応じた属性を，ユーザー情報として使用することができる．そのため，自由度の高い RLS 実装が可能となり，実際にそのアプリケーション・コンテキストを使用する FGAC 制御用ファンクションとして実装可能なニーズであれば，基本的にはどのようなセキュリティ要件にも対応できる．即ち，ビジネスロジックへの柔軟な対応が可能である．

### 7.2 FGAC によるアクセス制御の一般的な留意点

#### 7.2.1 ファンクション呼び出しによるレスポンスへの影響

一般的に，FGAC によるアクセス制御においては，FGAC ポリシー適用時に FGAC 制御用ファンクションを呼び出すため，そのオーバーヘッドによるレスポンスへの影響が想定される．これは全データを閲覧可能な特権ユーザーの場合も，付加する WHERE 句を NULL としているだけで，ファンクション自体は動作しているため同様である．

Oracle 社提供の検証結果[1]では，FGAC の使用によるレスポンスへの影響を懸念する必要はないと結論付けているが，システム全体の効率評価を行う場合には，FGAC ポリシー適用の影響を実機評価のポイントとして含めることが望ましい．

#### 7.2.2 FGAC の存在の見え難さによるリスク

1） DB への設定

開発・保守において，FGAC の存在の見え難さがネックになることが考えられる．アプリケーション・コンテキストや，エンティティに設定する FGAC ポリシー，FGAC 制御用ファンクションなど，FGAC によるアクセス制御に関わる設定は DB と密接に関係しているが，その存在を意識することは少ない．そのため，運用開始後の保守作業として DB のテーブル定義変更などを行う場合，FGAC ポリシー等，FGAC に関連する設定が保守対象か

ら漏れることが考えられる．

こういった事態を防ぐために，運用マニュアル等には FGAC の存在を考慮した記述をしておくことが必要である．また，想定される保守の内容に対し，それぞれの場合のメンテナンス対象について，事前にまとめておくことも有効だと考える．

2)　SQL の自動編集

FGAC による RLS は WHERE 句の自動付加により実現するため，ユーザーが発行した SQL と実際に発行される SQL との間に相違が出る．そのため，開発環境の共有にも注意が必要となる．例えば，FGAC によるアクセス制御機能の開発と他の開発とでスキーマを共有利用する場合，他の開発におけるエンティティへのアクセスにも FGAC ポリシーが適用されるため，登録したはずのデータが検索されず，プログラムの不具合だと勘違いされることが考えられる．このため，FGAC によるアクセス制御の開発においては，開発環境を他の開発と共有することは避けた方がよい．

## 8.　他システム開発へのノウハウの展開

最後に，他システムへの FGAC 適用に関して，今回の開発で得たノウハウを述べる．

### 8.1　組織構成・情報の重要度の観点からの対象データの分析

ユーザーの所属部署などによるアクセス制御の要件は，今回対象としたシステムに限らず，他のシステムにおいてもしばしば見られる．設計段階において，組織構成や扱う情報の重要度などに着目し，アクセス制御の対象とするデータを分析することが重要であると考える．その上で，今回紹介したようなユーザー情報の活用やアクセス権限情報の一元化，FGAC 制御用ファンクションの共通化を検討してみるのがよいと思われる．

### 8.2　レスポンスへの影響の検討

FGAC ポリシー適用時に FGAC 制御用ファンクションを呼び出すことにより，オーバーヘッドが発生することは 7.2.1 項で述べた．基本的には，FGAC 制御用ファンクションはユーザーからのアクセス要求の度に動作することになるため，ファンクション呼び出しのオーバーヘッド，ならびに，FGAC 制御用ファンクション自体の処理（今回の開発では，アクセス権限制御用テーブルからの情報取得）に起因するレスポンスへの影響について，事前に検討しておくことが必要だと思われる．

### 8.3　FGAC の存在の見え難さへの対策

7.2.2 項で述べたように，FGAC に関する設定が意識され難いことや，ユーザーが発行した SQL が自動編集されることなど，FGAC の存在の見え難さへの対策が重要だと考える．運用や保守の観点から，FGAC に起因するスキーマへの設定をマニュアル等にまとめておくこと，また，開発環境を他の開発と区別しておくことが望ましい．

## 9.　お　わ　り　に

これまで述べてきたように，FGAC の特徴をよく吟味して適用していくことで，自在なア

クセス制御が可能となる．

今回の開発では，大規模システムへのFGAC適用のための工夫として，アクセス権限情報の一元化やFGAC制御用ファンクションの共通化を考案した．そういったFGAC適用のための工夫も勿論重要ではあったが，それに加えて，FGACに関わる設定がDBと密接に関係していることや，FGAC制御項目をバッチ処理によって設定することから，開発時にはそれぞれを担当する開発チームとの協力が大変重要であった．大規模システムの開発においてはチーム間の連携を取りやすい体制も大切な要素であると，改めて認識した次第である．

世間的にもセキュリティ重視の流れが強まる中，FGACによるアクセス制御は今後も要望の高まっていく技術だと思われる．本稿で紹介した事例が，今後同様のシステム開発をしていく上で，何らかの参考となれば幸いである．

---

＊1　OLAPとはonline analytical processingの略．エンドユーザーが直接データベースの検索・集計を行い，その中から問題点や課題を発見する多次元分析処理のためのツールを指す．

＊2　OLEとはObject Linking and Embeddingの略．Windowsにおいて，アプリケーション間でデータを転送・共有するための仕組みである．

**参考文献**　［1］　岸和田 隆，「Oracle Database 10g 徹底検証レポート・第9回 Oracle 10gによるデータベースのセキュリティ対策」，オラクル通信，日本オラクル，No.84，2005年2月，http://www.oracle.co.jp/2shin/no84/o15oracle10g.html（2009年8月7日確認）

**執筆者紹介**　菅　井　綾　子（Ayako Sugai）

2001年USOL中部株式会社（当時，中部ソフト・エンジニアリング株式会社）入社．電力会社の社内システム開発・保守，アパレル商社の基幹業務システム開発などに従事．現在，エネルギーシステムプロジェクトに所属．