日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2021年8月16日月曜日

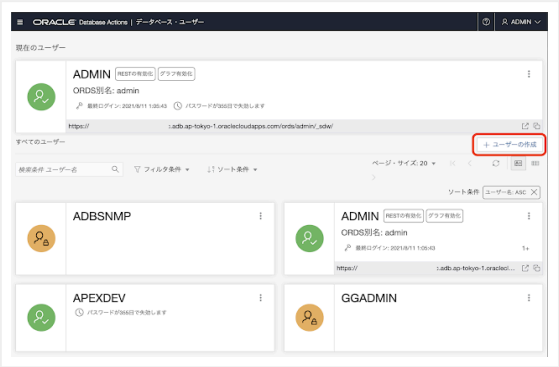
データベース・セキュリティの活用(8) - Real Application Security

Real Application Securityは仮想プライベート・データベースの後継となる機能です。軽量のアプリケーション・セッションを導入し、そのセッションに紐づくユーザー（アプリケーション・ユーザー）およびロール（アプリケーション・ロール）を定義します。これらの定義を元に、データを保護します。

実際に仮想プライベート・データベースで構成した保護を、Real Application Securityで行っていきます。

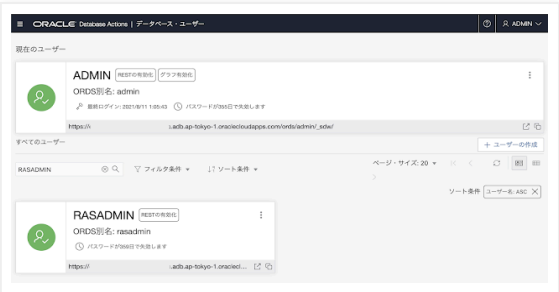
ユーザーRASADMINの作成

Real Application Securityを構成するユーザー**RASADMIN**を作成します。**データベース・アクション**にユーザー**ADMINで接続**し、**管理**の**データベース・ユーザー**を開きます。**ユーザーの作成**をクリックします。



ユーザー名を**RASADMIN**とし**パスワード**を設定、**Webアクセス**を**ON**にします。**ユーザーの作成**をクリックします。

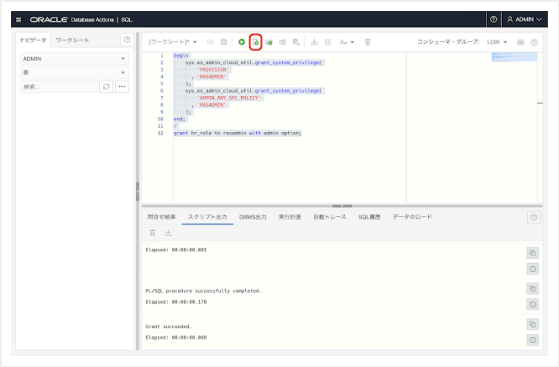
ユーザーRASADMINが作成されます。



開発の**SQL**開いて、ユーザーRASADMINに必要な権限を与えます。Autonomous DatabaseではプロシージャXS_ADMIN_CLOUD_UTIL.GRANT_SYSTEM_PRIVILEGEを呼び出します。ポリシーやACLの構成する権限**ADMIN_ANY_SEC_POLICY**と、ユーザーやロールを構成する権限**PROVISION**を与えます。パッケージXS_ADMIN_CLOUD_UTILはAutonomous Database向けに準備されているパッケージであり、オンプレミスではXS_ADMIN_UTILパッケージがその機能を提供しています。

```
begin
    sys.xs_admin_cloud_util.grant_system_privilege(
        'PROVISION'
      , 'RASADMIN'
    );
    sys.xs_admin_cloud_util.grant_system_privilege(
        'ADMIN_ANY_SEC_POLICY'
      , 'RASADMIN'
    );
end;
/
grant hr_role to rasadmin with admin option;
```

seminar200825-rasadmin-privs.sql hosted with ❤️ by GitHub view raw



APEXアプリケーションでは、仮想プライベート・データベースによる保護と同様に表AUTH_USERSを使用してユーザー認証を行います。そのために表AUTH_USERSを誰からでもアクセスできるようにします。

```
grant select on apexdev.auth_users to public;
```

seminar200825-grant_auth_users.sql hosted with ❤️ by GitHub view raw

ユーザー認証に使用する表を誰でもアクセス可能にする、といった実装は普通は行いません。今回の作業例ではReal Application Securityを外部ユーザーとして認証するよう実装します。この場合、ユーザーは外部サービスであるディレクトリ・サービス等で定義されていることを前提としますが、外部サービスの代わりに表AUTH_USERSを使用しているため誰でも検索できるようにしています。

ユーザーRASADMINの作成は完了です。データベース・アクションからサインアウトします。

認証スキームの構成

ユーザーRASADMINでサインインし、認証スキーム内で指定するアプリケーション・ロールとネームスペース・テンプレートの作成を行います。

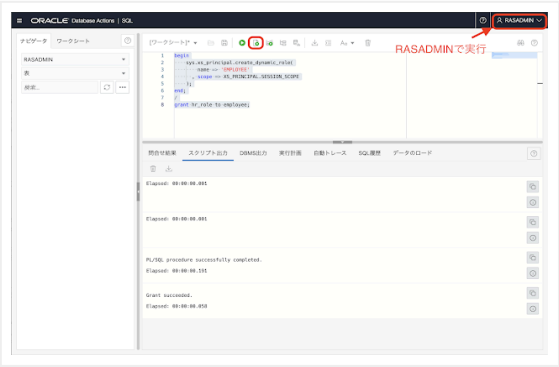
アプリケーション・ロール**EMPLOYEE**を作成します。Real Application Securityのユーザーを外部ユーザーとして構成するため（アプリケーション・ユーザーはデータベースに作成されない）、ロールを事前にユーザーに割り当てることはできません。そのためロールは**動的ロール**として作成します。

動的ロールの作成にはプロシージャXS_PRINCIPAL.CREATE_DYNAMIC_ROLEを呼び出します。作成した動的ロールにロールHR_ROLEを割り当てることにより、スキーマHRにある表EMP、DEPTへのアクセス権限を与えます。

```
begin
    sys.xs_principal.create_dynamic_role(
        name => 'EMPLOYEE'
      , scope => XS_PRINCIPAL.SESSION_SCOPE
    );
```

```
end;
/
grant hr_role to employee;
```

seminar200825-approle-emoployee.sql hosted with ❤ by GitHub view raw



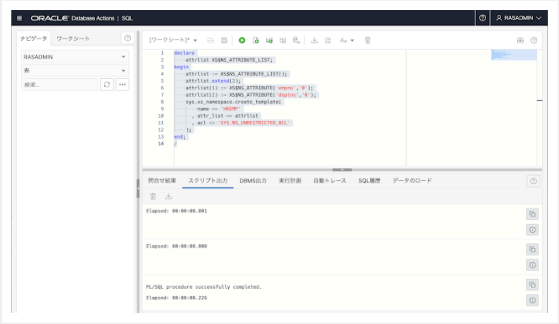
作成した動的ロールはビューDBA_XS_DYNAMIC_ROLES、動的ロールに与えられたロールはビューDBA_XS_ROLE_GRANTSより確認できます。

続いて、ネームスペース・テンプレート**HREMP**を作成します。仮想プライベート・データベースでのアプリケーション・コンテキストと同じ用途で使用します。サインインしたユーザーの従業員番号EMPNOを**empno**、部門番号DEPTNOを**deptno**として保持します。アプリケーション・コンテキストはファンクションSYS_CONTEXTを通して値を参照することに対して、RASのネームスペースではファンクションXS_SYS_CONTEXTが使われます。

ネームスペースの作成にはプロシージャXS_NAMESPACE.CREATE_TEMPLATEを呼び出します。

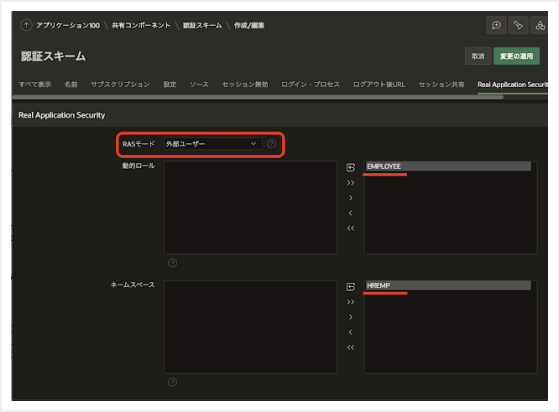
```
declare
    attrlist XS$NS_ATTRIBUTE_LIST;
begin
    attrlist := XS$NS_ATTRIBUTE_LIST();
    attrlist.extend(2);
    attrlist(1) := XS$NS_ATTRIBUTE('empno','0');
    attrlist(2) := XS$NS_ATTRIBUTE('deptno','0');
    sys.xs_namespace.create_template(
        name => 'HREMP'
      , attr_list => attrlist
      , acl => 'SYS.NS_UNRESTRICTED_ACL'
    );
end;
/
```

seminar200825-nstemp-hremp.sql hosted with ❤ by GitHub view raw



作成されたネームスペース・テンプレートはビューDBA_XS_NS_TEMPLATESおよびDBA_XS_NS_TEMPLATE_ATTRIBUTESより確認できます。

Real Application Security側での準備ができたので、APEXアプリケーションの認証スキームを変更します。アプリケーションの**共有コンポーネント**より、認証スキーム**従業員による認証**を開き、**Real Application Security**の**RASモード**を**外部ユーザー**に変更します。**動的ロールEMPLOYEE**および**ネームスペース**の**HREMP**をシャトルの右側に移し、有効にします。

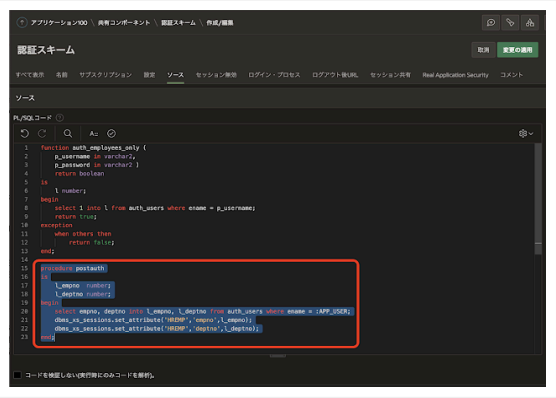


認証スキームの**ソース**に、認証後に実行するプロシージャとして、ネームスペースHREMPを初期化する**postauth**を追加します。追加するコードは以下になります。

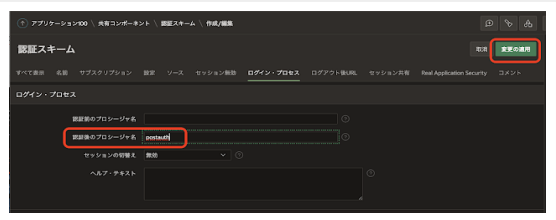
```
procedure postauth
is
    l_empno  number;
    l_deptno number;
begin
    select empno, deptno into l_empno, l_deptno from auth_users where ename = :APP_USER;
    dbms_xs_sessions.set_attribute('HREMP','empno',l_empno);
    dbms_xs_sessions.set_attribute('HREMP','deptno',l_deptno);
end;
```

seminar210825-postauth_hremp.sql hosted with ❤ by GitHub view raw

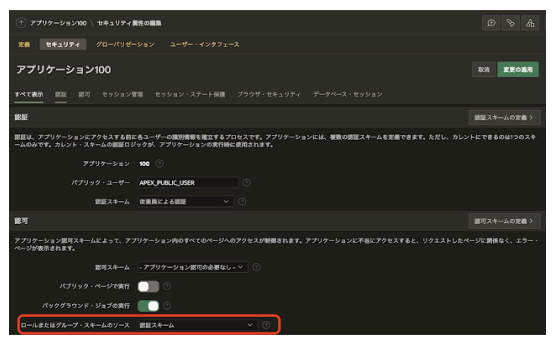
ソースの**PL/SQLコード**にすでに記載されているファンクションauth_employees_onlyに続けて記述します。



ログイン・プロセスの**認証後のプロシージャ名**に**postauth**を指定します。**変更の適用**をクリックします。



アプリケーション定義属性の**セキュリティ**を開きます。**認可**の**ロールまたはグループ・スキームのソース**を**認証スキーム**に変更します。RASモードの動的ロールを有効にする、必須の設定です。



仮想プライベート・データベースの構成で使用していた、**データベース・セッション**の**初期化PL/SQLコード**と**PL/SQLコードのクリーンアップ**のプロシージャを削除し、**変更の適用**をクリックします。

以上で認証スキームがRASモードに変更されました。RASモードは外部ユーザーなので、RASのユーザーの作成は不要です。

認証スキームの動作確認を行います。アプリケーションを実行し、ユーザーSCOTTにて**サインイン**してみます。

エラーが発生せず正常にサインインできれば、Real Application Securityは有効になっています。

Real Application Securityによる保護の設定

APEXアプリケーションからサインインしたセッションに、Real Application Securityのアプリケーション・ユーザーとロールが割り当たるようになりました。このユーザーとロールによるデータの保護を構成します。

データベース・アクションにユーザー**RASADMIN**で接続します。**開発**の**SQL**を開きます。

最初にセキュリティ・クラス**emp_priv**を作成します。プロシージャXS_SECURITY_CLASS.CREATE_SECURITY_CLASSを呼び出します。**parent_list**として**sys.dml**を指定することで、標準で定義されている権限のリスト("SELECT","UPDATE","INSERT","DELETE")を引き継いでいます。セキュリティ・クラスemp_privが持つ特別な権限は**view_sal**で、これは列SALとCOMMへのアクセスを制御します。

```
begin
    sys.xs_security_class.create_security_class(
        name => 'emp_priv'
      , parent_list => xs$name_list('sys.dml')
      , priv_list => xs$privilege_list(xs$privilege('view_sal'))
    );
end;
/
```

seminar210825-security_class_emp_priv.sql hosted with ❤ by GitHub view raw

作成されたセキュリティ・クラスはビューUSER_XS_SECURITY_CLASSESより確認できます。

２つのACL、**emp_acl**と**mgr_acl**を作成します。プロシージャXS_ACL.CREATE_ACLを呼び出します。emp_aclはサインインしたユーザーと同じ部門の従業員にアクセスを限定するために使用します。mgr_aclはサインインしたユーザーがマネージャーである従業員の列SALとCOMMにアクセスを限定するために使用します。

作成したACLは、セキュリティ・クラス**emp_priv**へ紐付けます。また、アクセス制御エントリ(ACE)の**principal_name**として動的ロール**employee**を指定しています。そのため、動的ロールEMPLOYEEを持っているセッションで、これらのACLで保護されているアクセスが許可されます。

```
declare
    aces xs$ace_list := xs$ace_list();
begin
    aces.extend(1);

    aces(1) := xs$ace_type(
        privilege_list => xs$name_list('select','insert','update','delete')
      , principal_name => 'employee'
    );

    sys.xs_acl.create_acl(
        name => 'emp_acl'
      , ace_list => aces
      , sec_class => 'emp_priv'
    );

    aces(1) := xs$ace_type(
        privilege_list => xs$name_list('select','insert','update','delete','view_sal')
      , principal_name => 'employee'
    );

    sys.xs_acl.create_acl(
        name => 'mgr_acl'
      , ace_list => aces
      , sec_class => 'emp_priv'
    );
end;
/
```

seminar200825-create_acls.sql hosted with ❤️ by GitHub view raw

作成されたACLはビューUSER_XS_ACLSより確認できます。ACLに紐づいているACEはビューUSER_XS_ACESより確認できます。

データ・セキュリティ・ポリシー**employees_ds**を作成します。プロシージャXS_DATA_SECURITY.CREATE_POLICYを呼び出します。

ACLの**emp_acl**を適用する**realm**の設定として、**deptno = xs_sys_content('HREMP','deptno')**を指定します。結果として、サインインした従業員と同じ部門の従業員の情報にアクセスが制限されま

す。また、**mgr_aclのrealmはmgr = xs_sys_content('HREMP','empno')**とします。結果として、サインインした従業員がマネージャーである従業員にアクセスが制限されます。

```
declare
    realms xs$realm_constraint_list := xs$realm_constraint_list();
    cols   xs$column_constraint_list := xs$column_constraint_list();
begin
    realms.extend(2);
    realms(1) := xs$realm_constraint_type(
        realm => 'deptno = xs_sys_context(''HREMP'',''deptno'')'
      , acl_list => xs$name_list('emp_acl')
    );
    realms(2) := xs$realm_constraint_type(
        realm => 'mgr = xs_sys_context(''HREMP'',''empno'')'
      , acl_list => xs$name_list('mgr_acl')
    );

    cols.extend(1);
    cols(1) := xs$column_constraint_type(
        column_list => xs$list('SAL','COMM')
      , privilege => 'view_sal'
    );

    sys.xs_data_security.create_policy(
        name => 'employees_ds'
      , realm_constraint_list => realms
      , column_constraint_list => cols
    );
end;
/
```

seminar200825-data_security_policy.sql hosted with ❤ by GitHub view raw

作成されたデータ・セキュリティ・ポリシーはビューUSER_XS_POLICIES、セキュリティ・ポリシーに含まれるレルムの構成はUSER_XS_REALM_CONSTRAINTS、USER_XS_COLUMN_CONSTRAINTS等から確認できます。

作成したデータ・セキュリティ・ポリシー**employees_ds**を表**HR.EMPに適用**します。プロシージャXS_DATA_SECURITY.APPLY_OBJECT_POLICYを呼び出します。

```
begin
    sys.xs_data_security.apply_object_policy(
        policy => 'employees_ds'
      , schema => 'hr'
      , object => 'emp'
    );
end;
/
```

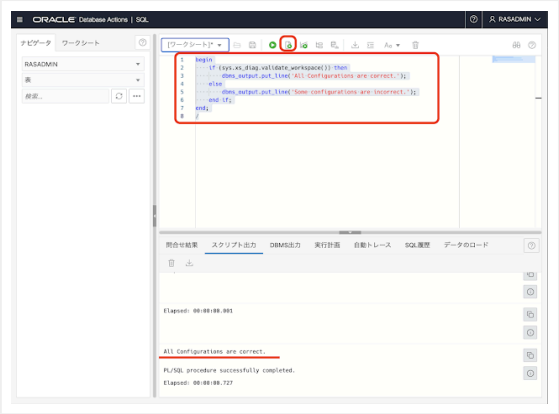
seminar200825-apply_object_policy.sql hosted with ❤ by GitHub view raw

以上でReal Application Securityによる表HR.EMPの保護ができました。適用されたデータ・セキュリティ・ポリシーはビューALL_XS_APPLIED_POLICIESより確認できます。

設定を検証するために、プロシージャXS_DIAG.VALIDATE_WORKSPACEを呼び出します。All Configurations are correct.と表示されると、設定に問題はありません。

```
begin
    if (sys.xs_diag.validate_workspace()) then
        dbms_output.put_line('All Configurations are correct.');
    else
        dbms_output.put_line('Some configurations are incorrect.');
    end if;
end;
/
```

seminar200825-validate_workspace.sql hosted with ❤ by GitHub view raw



問題がある場合は、ビュー**XS$VALIDATION_TABLE**より検証結果をレポートします。

```
select * from xs$validation_table order by 1,2,3,4;
```

seminar210825-report_validate.sql hosted with ❤ by GitHub view raw

以上でReal Application Securityによる表HR.EMPの保護は完了です。

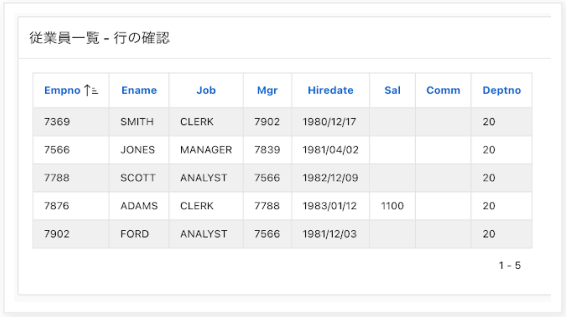
動作確認

テスト用アプリケーションを実行し、今までに設定したReal Application Securityによる保護を確認します。

従業員名に以下を指定します。

SCOTT' or '1' = '1

仮想プライベート・データベースによる保護と同様に、サインインしたユーザーSCOTTと同じ部門の従業員の一覧に制限されています。

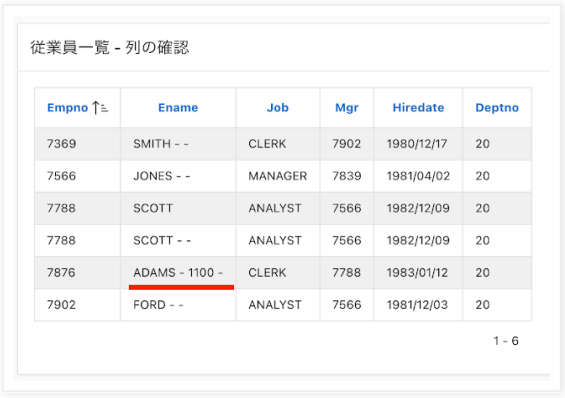

従業員一覧 - 行の確認

Empno	Ename	Job	Mgr	Hiredate	Sal	Comm	Deptno
7369	SMITH	CLERK	7902	1980/12/17			20
7566	JONES	MANAGER	7839	1981/04/02			20
7788	SCOTT	ANALYST	7566	1982/12/09			20
7876	ADAMS	CLERK	7788	1983/01/12	1100		20
7902	FORD	ANALYST	7566	1981/12/03			20

1 - 5

従業員名に以下を指定します。

SCOTT' UNION SELECT EMPNO, ENAME || ' - ' || SAL || ' - ' || COMM ENAME, JOB, MGR, HIREDATE, DEPTNO FROM HR.EMP WHERE '1'='1


従業員一覧 - 列の確認

Empno	Ename	Job	Mgr	Hiredate	Deptno
7369	SMITH - -	CLERK	7902	1980/12/17	20
7566	JONES - -	MANAGER	7839	1981/04/02	20
7788	SCOTT	ANALYST	7566	1982/12/09	20
7788	SCOTT - -	ANALYST	7566	1982/12/09	20
7876	ADAMS - 1100 -	CLERK	7788	1983/01/12	20
7902	FORD - -	ANALYST	7566	1981/12/03	20

1 - 6

列EnameにSALの情報が表示されますが、サインインしたユーザーSCOTTがマネージャーである従業員に限定されています。

以上で作業は完了です。

Real Application Securityによって保護されたままだと、後続の作業に支障があるため、保護の設定を削除します。

```
begin
    sys.xs_security_class.delete_security_class(
        sec_class => 'emp_priv'
      , delete_option => xs_admin_util.cascade_option
    );

    sys.xs_acl.delete_acl(
        acl => 'emp_acl'
      , delete_option => xs_admin_util.cascade_option
    );
```

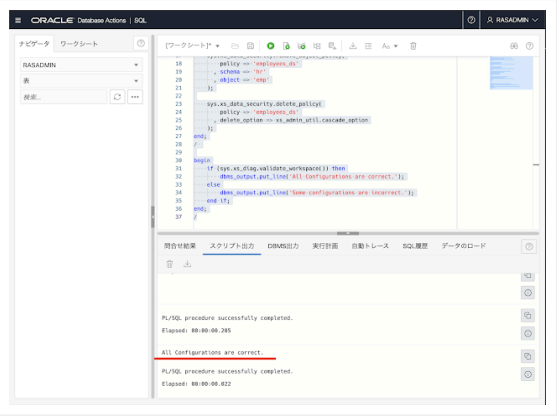
```
    sys.xs_acl.delete_acl(
        acl => 'mgr_acl'
      , delete_option => xs_admin_util.cascade_option
    );

    sys.xs_data_security.remove_object_policy(
        policy => 'employees_ds'
      , schema => 'hr'
      , object => 'emp'
    );

    sys.xs_data_security.delete_policy(
        policy => 'employees_ds'
      , delete_option => xs_admin_util.cascade_option
    );
end;
/

begin
    if (sys.xs_diag.validate_workspace()) then
        dbms_output.put_line('All Configurations are correct.');
    else
        dbms_output.put_line('Some configurations are incorrect.');
    end if;
end;
/
```

seminar200825-delete_ras.sql hosted with ❤️ by GitHub view raw



セキュリティ・クラス、ACL、データ・セキュリティ・ポリシーの削除、およびオブジェクトへのデータ・セキュティ・ポリシーの適用を解除を行い、設定の検証を実行しています。

All Configuration are correct.と出力されれば、保護の設定は正常に削除されています。

続く

Yuji N. 時刻: 17:50

共有

‹ ホーム ›

ウェブ バージョンを表示

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示