International Journal of Latest Engineering Research and Applications (IJLERA)
www.ijlera.com
Volume 1 – Issue 1 pp: 33-38

Cloud Data Management in Multi Tenant Architecture

Udhaya Chandrika Shanmugam
Research and Development Centre, Bharathiar University, Coimbatore.

Abstract: Cloud Data management in multi tenant architecture, needs Data Sharding, Scaling, filtering and applying data encryption to share data for a particular client. Tenants are groups of assessors who need a shared instance in Multitenant Architectural Environmental Application. Every tenant's data in cloud can be shared with proper configuration management, user management and tenant functional properties. A cloud data in multi tenant cloud architecture is sharing a single cloud instance data to multiple tenants. Sharding is possibly very transparent key technique to provide splited data to the tenant in Multitenant Application and accessing correct shards information. Filtering and Encryptions are common practices can be applied in Cloud Data Management for best performance in Multitenant Cloud Application Architecture.
Keywords: Data Scaling, Data Filtering, Encrypting.

Introduction

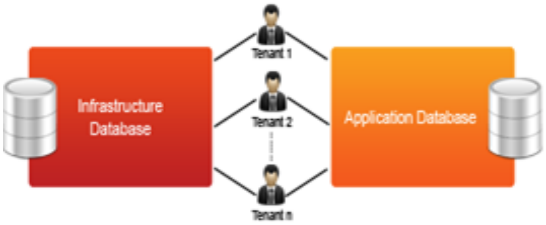
Managing cloud data by:

1. Cloud Data Sharding and Scaling in Multi Tenant Applications
2. Filtering Data using Tenant View filters in cloud databases.
3. Applying Encryptions for data Security while handling multiple Tenants.

Cloud Data Sharding and Scaling in Multi Tenant Applications

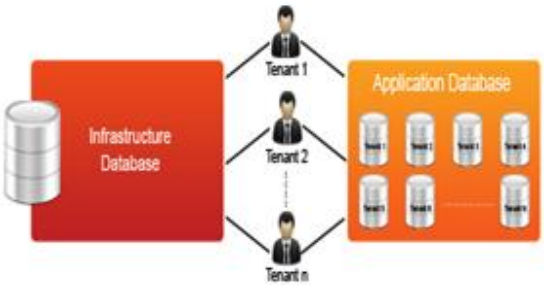
Cloud Data Sharding in detail: A Saas product which is developed for managing cloud data instance in multitenant environment, it should scalable without compromising the performance. Also it should be reliable. And data should be available to required tenant. The intention of building large scale cloud applications are to handle and serve multiple tenant requirements in cloud architecture. When the number tenant keep increase the performance and lookup for filtered data availability. **Cloud Data Sharding through different Types of Method are:**

Approach One: Managing Two different databases one for Tenants data storages and another one for Application Data.



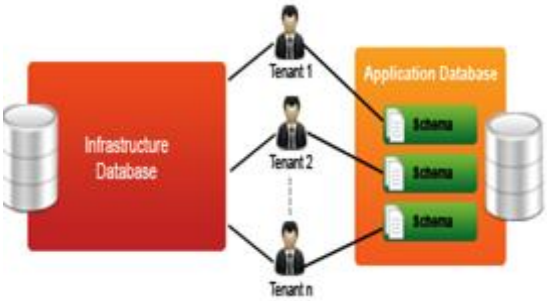
Shared Databases

Approach Two: Here tenant requesting for complete isolation of their secured data in the Database can be achieved by creating separate instance for the tenant can be created. Other tenant's data will reside in common database and transitions will happen in instance of db created for secured tenant.



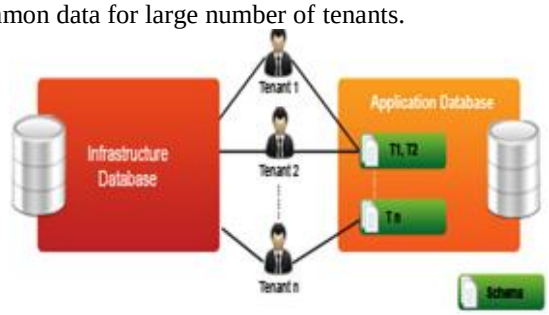
Tenant Based Database

Approach Three: Separating the tenants by separate schema modules. It gives high level of security, isolation and complete freedom to access the protected data in multiple tenant environments.



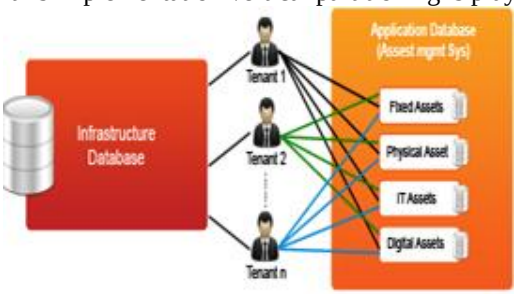
Schema Based Database

Approach Four: A single database can be shared to multiple tenants depends on the volume of data requirements, and sharing common data for large number of tenants.



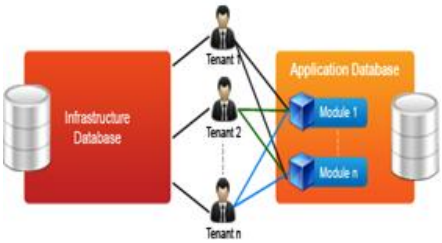
Scalable Database with Different Shared Schemas

Approach Five: Here different debases can be considered for different modules. Tenants can be access required module in particular database. In this implementation vertical partitioning is plays big role.



Module based Databases

Approach Six: Applying horizontal and vertical partitioning in Databases, based on the required modules which are stored in different databases. This data Sharding can be portioned by Tenant Identity.



Module based Shared Database and Shared Schemas

Filtering Data using Tenant View filters in cloud databases.

MS SQL Server recommends Tenant View Filtering in Multitenancy.

CREATE VIEW TenantAsEmployee AS
Select * from Employee WHERE TenID = cUser_cID()
The GROUPING in SQL will gather all of the rows together that contain data in the specified column(s) and will allow aggregate functions to be performed on the one or more columns. SQL Grouping allows accessing the objects in the group to create different group with different schema of objects

GRANT CONNECT TO TenantOne;
GRANT RESOURCE TO TenantOne;
GRANT GROUP TO TenantOne;

Now that the group is created as TenantOne, a user with permissions of create tables for other Tenants could create a table in the new schema by specifying the owner on the

CREATE TABLE command:

```
CREATE TABLE TenantOne.Employees
(
 EmployeeID    INTEGER    NOT NULL,
 ManagerID    INTEGER    NULL,
 Surname      CHAR(20)    NOT NULL,
 GivenName    CHAR(20)    NOT NULL,
 ST_DeptID  INTEGER    NOT NULL,
 Street       CHAR(30)    NOT NULL,
```

```
 City        CHAR(20)    NOT NULL,
 State       CHAR(16)    NULL,
 Country     CHAR(16)    NULL,
 CONSTRAINT EmployeesKey PRIMARY KEY (EmployeeID)
);
```

Just a few more commands to allow a given user to access the new table:

```
GRANT SELECT, INSERT, DELETE, UPDATE ON TenantOne.Employees TO TenantOne;
```

GRANT, SELECT, INSERT, DELETE, UPDATE commands can be used in TenantOne.Employees TO TenantOne Group.

```
GRANT CONNECT TO TUser IDENTIFIED BY Tpass;

GRANT MEMBERSHIP IN GROUP TenantOne TO  TUser;

 CONNECT AS TUser IDENTIFIED BY  Tpass;
 SELECT * FROM TenantOne;
```

Table VIEW creates Virtual Table to manipulate SQL queries and commands.

```
CREATE TABLE "INFRACTUREDB_TENANT".UserTenantMap (
TenantId          INTEGER NOT NULL,
ActiveTenants         CHAR(255) NOT NULL,
CONSTRAINT "UserTenantMapKey" PRIMARY KEY ("TenandId", "ActiveTenants" )
);
```

Next, the base table is defined with a TenantIdcolumn, and a VIEW is defined, joining with the UserTenantMap table:

```
CREATE TABLE "INFRACTUREDB_TENANT"."TenantModules" (
   "TenandId"  INTEGER NOT NULL                    DEFAULT ( CAST( '1' AS  INTEGER ) ),
   "ST_DeptID"              INTEGER NOT NULL,
   "ST_DeptName"            CHAR(40) NOT NULL,
   "ST_DeptHeadID"          INTEGER NULL,
   CONSTRAINT " TenantModules Key" PRIMARY KEY ("TenandId","ST_DeptID") );

CREATE VIEW "TENANTVIEW"."SharedTableDept" (
   "ST_DeptID",
   "ST_DeptName",
   "ST_DeptHeadID" ) AS SELECT
   "ST_DeptID",
   "ST_DeptName",
   "ST_DeptHeadID"
FROM "INFRACTUREDB_TENANT"."TenantModules"
JOIN "INFRACTUREDB_TENANT"."UserTenantMap"
ON "TenantModules"."TenandId" = "UserTenantMap"."TenandId"
WHERE
"UserTenantMap"."ActiveTenants" = CURRENT USER;
```

The application can be coded to simply SELECT from the SharedTableDept table as before. Inserts or Deletes can be handled using an INSTEAD OF trigger defined on the view. INSTEAD OF triggers allow alternate actions to be performed, rather than the DML that caused the trigger to fire. Read more about SQL Anywhere's implementation of INSTEAD OF triggers here. Here is an INSTEAD OF trigger to handle inserts on the SharedTableDept VIEW:

```
 CREATE OR REPLACE PROCEDURE "INFRACTUREDB_TENANT"."InsertST_DeptProc"
(IN new_ST_DeptID  INTEGER,
  IN new_ST_DeptName  CHAR(40),
  IN new_ST_DeptHeadID INTEGER,
  IN inserting User  CHAR(255)
)
BEGIN
  DECLARE new_TenandId  INTEGER;
  SELECT TenantId INTO new_TenandId
  FROM "INFRACTUREDB_TENANT"."UserTenantMap"
  WHERE ActiveTenants = insertingUser;
  INSERT INTO "INFRACTUREDB_TENANT"."TenantModules" (
  TenandId,
  ST_DeptID,
  ST_DeptName,
  ST_DeptHeadID )
  VALUES (
    new_TenandId,
    new_ST_DeptID,
    new_ST_DeptName,
    new_ST_DeptHeadID
  );
END
;


GRANT EXECUTE ON "INFRACTUREDB_TENANT"."InsertST_DeptProc" TO "TENANTVIEW";
CREATE OR REPLACE TRIGGER Insert_SharedTableDept
INSTEAD OF INSERT ON "TenantView"."SharedTableDept"
REFERENCING NEW AS new_row
FOR EACH ROW

BEGIN

  CALL "INFRACTUREDB_TENANT"."InsertST_DeptProc" (
  new_row.ST_DeptID,
  new_row.ST_DeptName,
  new_row.ST_DeptHeadID,
  CURRENT USER   );

END
;
```

Tenant Data Encryption

An RSA public-key / private-key pair can be generated by the following steps:

1. Generate a pair of large, random primes p and q.
2. Compute the modulus n as n = pq.
3. Select an odd public exponent e between 3 and n-1 that is relatively prime to p-1 and q-1.
4. Compute the private exponent d from e, p and q.
5. Output (n, e) as the public key and (n, d) as the private key.

The encryption operation in the RSA cryptosystem is exponentiation to the e th power modulo n:

$$c = \text{ENCRYPT}(m) = m^e \bmod n$$

The input m is the message; the output c is the resulting cipher text. In practice, the message m is typically some kind of appropriately formatted key to be shared. The actual message is encrypted with the shared key using a traditional encryption algorithm. This construction makes it possible to encrypt a message of any length with only one exponentiation. The decryption operation is exponentiation to the dth power modulo n:

$$m = \text{DECRYPT}(c) = c^d \bmod n \quad .$$

The relationship between the exponents e and d ensures that encryption and decryption are inverses, so that the decryption operation recovers the original message m. Without the private key (n, d) (or equivalently the prime factors p and q), it's difficult to recover m from c. Consequently, n and e can be made public without compromising security, which is the basic requirement for a public-key cryptosystem.

Conclusion

In this paper we discussed about Scaling and Sharding to meet the changing demands of cloud data with highly-available, high-performance services in Multitenant Architecture. Typically, data filtering in multi tenant environment in cloud application will involve taking out information for a particular tenant. Also Discussed about encrypting of cloud data in Multitenant application and securing data in cloud data management using RSA algorithm.