

# Multi-Tenant Arquitectura de Datos

de junio de 2006

Frederick Chong, Gianpaolo Carraro, y Roger Wolter  
Microsoft Corporation

Se aplica a:

Arquitectura de la aplicación  
de software como servicio (SaaS)

**Resumen:** El segundo artículo de la serie sobre el diseño de aplicaciones para múltiples usuarios se identifican tres enfoques distintos para la creación de arquitecturas de datos. (25 páginas impresas)

## Expresiones de gratitud

Muchas gracias a Paul Henry por su ayuda en la redacción técnica.

**Para mayor referencia, un ARCast está disponible:**

[ARCast - Software As A Service](#)

## Contenido

[Introducción](#) [Tres estrategias para la gestión de datos multiusuario](#) [La elección de un enfoque](#) [Al darse cuenta de multi-Tenant Arquitectura de Datos](#) [Conclusión](#) [relacionadas con la orientación de votos](#)

## Introducción

La confianza , o la falta de ella, es el principal factor de bloqueo de la adopción de software como servicio (SaaS). Un caso podría ser que los datos son el activo más importante de cualquier negocio, los datos acerca de los productos, clientes, empleados, proveedores, y más. Y datos, por supuesto, está en el corazón de SaaS. Las aplicaciones SaaS proporcionan a los clientes un acceso centralizado, basado en la red de datos con menos sobrecarga que es posible cuando se utiliza una aplicación instalada localmente. Sin embargo, con el fin de aprovechar los beneficios de SaaS, una organización debe entregar un nivel de control sobre sus propios datos, confiando en que el proveedor de SaaS para mantenerlo a salvo y lejos de miradas indiscretas.

Para ganar esta confianza, una de las principales prioridades de un arquitecto de SaaS prospectivo está creando una arquitectura de datos SaaS que es a la vez robusta y segura suficiente para satisfacer a los inquilinos o clientes que se preocupan por entregar el control de los datos empresariales vitales a un tercero, y al mismo tiempo ser eficiente y rentable para administrar y mantener.

Este es el segundo artículo de la serie sobre el diseño de aplicaciones para múltiples usuarios. El primer artículo, [Arquitectura Estrategias para la captura de la larga cola](#) , introdujo el modelo SaaS a un alto nivel y discute sus retos y beneficios. Está disponible en MSDN. Otros artículos en la serie se centrarán en temas tales como el flujo de trabajo y el diseño de interfaz de usuario, la seguridad en general, y otros.

En este artículo, vamos a ver en el continuo entre datos aislados y datos compartidos, e identificamos tres enfoques distintos para la creación de arquitecturas de datos que caen en diferentes lugares a lo largo del continuo. A continuación, vamos a

explorar algunos de los factores técnicos y de negocio a considerar al decidir qué método utilizar. Por último, vamos a presentar patrones de diseño para garantizar la seguridad, la creación de un modelo de datos extensible, y la ampliación de la infraestructura de datos.

## Tres estrategias para la gestión de datos multiusuario

La distinción entre los datos compartidos y datos aislados no es binaria. En cambio, es más bien un proceso continuo, con muchas variaciones que son posibles entre los dos extremos.

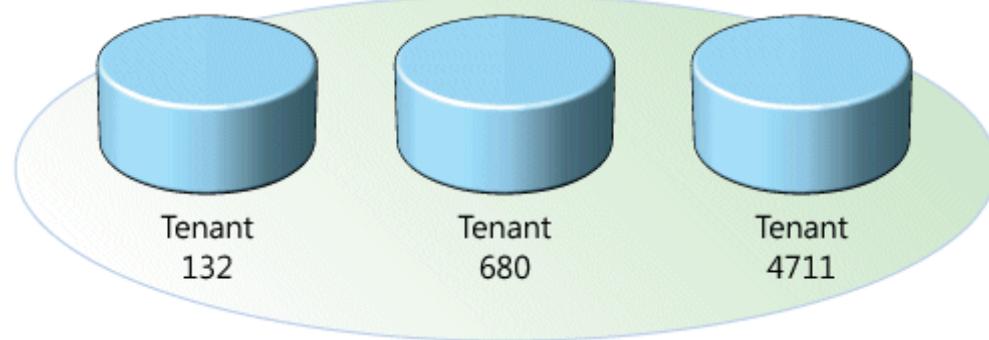


Arquitectura de datos es un área en la que el grado óptimo de aislamiento para una aplicación SaaS puede variar significativamente dependiendo de las consideraciones técnicas y de negocios. Los arquitectos de datos experimentados están acostumbrados a considerar un amplio espectro de opciones en el diseño de una arquitectura para cumplir con un conjunto específico de desafíos, y SaaS no es la excepción. Examinaremos tres grandes enfoques, cada uno de los cuales se encuentra en una ubicación diferente en el continuo entre el aislamiento y el compartir.



### Las bases de datos separadas

El almacenamiento de datos en bases de datos separadas inquilino es el método más sencillo para el aislamiento de datos.



**La figura 1. Este método utiliza una base de datos diferente para cada inquilino**

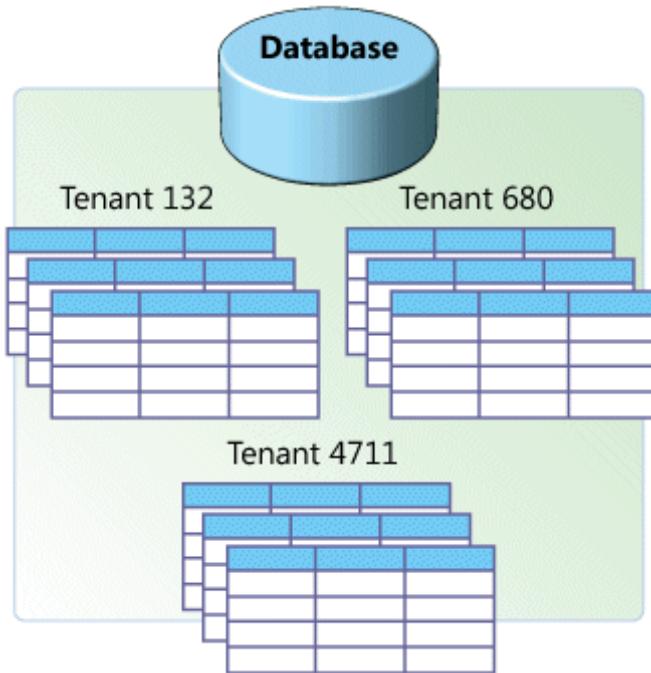
Los recursos informáticos y código de aplicación son generalmente compartida entre todos los inquilinos en un servidor, pero cada inquilino tiene su propio conjunto de datos que permanecen lógicamente aislado de datos que pertenecen a todos los otros inquilinos. Metadatos asociados a cada base de datos con el inquilino correcta, y la base de datos de seguridad impide cualquier inquilino tengan acceso a los datos de forma accidental o malintencionada otros inquilinos.

Dando a cada inquilino su propia base de datos hace que sea fácil de expandir el modelo de datos de la aplicación (explicado más adelante) para satisfacer las necesidades individuales de los inquilinos, y la restauración de los datos del inquilino de copias de seguridad en el caso de un fallo es un procedimiento relativamente simple. Por desgracia, este enfoque tiende a conducir a mayores costos para el mantenimiento de los equipos y realizar copias de seguridad de datos de inquilinos. Los costos de hardware también son más altos de lo que son en virtud de enfoques alternativos, como el número de inquilinos que puede ser alojada en un servidor de base de datos dada está limitado por el número de bases de datos que el servidor puede soportar. (El uso de autoclose para descargar las bases de datos de la memoria cuando no hay conexiones activas pueden hacer que una aplicación más escalable mediante el aumento del número de bases de datos de cada servidor puede soportar.)

La separación de los datos de inquilinos en bases de datos individuales es el enfoque "premium", y los relativamente altos requisitos y los costes de hardware y mantenimiento, resulta apropiada para los clientes que están dispuestos a pagar extra para mayor seguridad y capacidad de personalización. Por ejemplo, los clientes en campos como la banca o la gestión de registros médicos a menudo tienen muy fuertes requisitos de aislamiento de datos, y puede que ni siquiera piense en una aplicación que no proporciona a cada inquilino con su propia base de datos individual.

## Base de datos compartida, esquemas separados

Otro enfoque implica múltiples inquilinos de la vivienda en la misma base de datos, con cada inquilino tiene su propio conjunto de tablas que se agrupan en un esquema creado específicamente para el inquilino.



**Figura 2. En este enfoque, cada inquilino tiene su propio conjunto independiente de las tablas de una base de datos común**

Cuando un cliente se suscribe por primera vez al servicio, el subsistema de aprovisionamiento crea un conjunto discreto de mesas para el inquilino y lo asocia con el propio esquema del inquilino. Puede utilizar el comando CREATE SQL para crear un esquema y autorizar una cuenta de usuario para acceder a ella. Por ejemplo, en Microsoft SQL Server 2005:

```
CREATE SCHEMA ContosoSchema AUTORIZACIÓN Contoso
```

La aplicación puede crear y tablas de acceso dentro del esquema del inquilino mediante el **SCHEMANAME.TABLENAMES** convención:

```
CREAR TABLA (ContosoSchema.Resumes de empleado clave int identidad primaria,
Reanudar nvarchar (max))
```

Una vez creado el esquema, que se establece como el esquema predeterminado para la cuenta del arrendatario:

MODIFICAR USUARIO CON Contoso DEFAULT\_SCHEMA = ContosoSchema

Una cuenta de inquilino puede acceder a tablas dentro de su esquema predeterminado especificando sólo el nombre de la tabla, en lugar de utilizar el **SCHEMANAME.TABLENAMES** convención. De esta manera, un único conjunto de instrucciones SQL se puede crear para todos los inquilinos, que cada inquilino puede usar para acceder a sus propios datos:

\* SELECT DE CV

Al igual que el enfoque aislado, el enfoque de esquema separado es relativamente fácil de implementar, y los inquilinos pueden extender el modelo de datos tan fácilmente como con el enfoque de base de datos independiente. (Las tablas se crean a partir de un conjunto predeterminado estándar, pero una vez que se crean que ya no tienen que cumplir con el conjunto predeterminado, y los inquilinos pueden añadir o modificar columnas e incluso tablas si lo deseas.) Este enfoque ofrece un grado moderado de aislamiento de datos lógicos para los inquilinos preocupados por la seguridad, aunque no tanto como un sistema completamente aislado sería, y puede soportar un mayor número de inquilinos por servidor de base de datos.

Una desventaja significativa del enfoque de esquema separada es que los datos inquilino es más difícil de restaurar en el caso de un fallo. Si cada inquilino tiene su propia base de datos, la restauración de los datos de un solo inquilino significa simplemente la restauración de la base de datos de la copia de seguridad más reciente. Con una aplicación de esquema separado, la restauración de la base de datos significaría encima de los datos de todos los inquilinos en la misma base de datos con los datos de copia de seguridad, independientemente de que cada uno ha experimentado ninguna pérdida o no. Por lo tanto, para restaurar los datos de un solo cliente, el administrador de base de datos puede tener que restaurar la base de datos a un servidor temporal, y luego importar tablas del cliente en el servidor de aplicaciones para una tarea complicada y potencialmente mucho tiempo de producción.

El enfoque esquema separado es apropiado para las aplicaciones que utilizan un número relativamente pequeño de tablas de base de datos, en el orden de aproximadamente 100 tablas por inquilino o menos. Este enfoque normalmente tiene capacidad para más inquilinos por servidor que el enfoque de base de datos independiente puede, por lo que puede ofrecer la aplicación a un menor costo, siempre y cuando sus clientes aceptan que sus datos de ubicación conjunta con la de otros inquilinos.

## Base de datos compartida, esquema compartido

Un tercer enfoque consiste en utilizar la misma base de datos y el mismo conjunto de tablas de datos para alojar múltiples inquilinos. Una tabla dada puede incluir registros de varios inquilinos almacenados en cualquier orden; una columna de ID del cliente asocia cada registro con el inquilino adecuado.

TenantID	CustName	Address	
4	TenantID	ProductID	ProductName
1	4	TenantID	Shipment
6	1	4711	324965
4	6	132	2006-02-21
4	6	680	2006-04-08
		4711	654109
			2006-03-27
			324956
			2006-02-23

**Figura 3. En este enfoque, todos los inquilinos comparten el mismo conjunto de mesas, y una ID del cliente asociados a cada inquilino con las filas de los que es titular**

De los tres métodos explicados aquí, el enfoque de esquema compartido tiene el hardware más bajo y los costos de copia de seguridad, ya que le permite servir a un mayor número de inquilinos por servidor de base de datos. Sin embargo, debido a

múltiples inquilinos comparten las mismas tablas de bases de datos, este enfoque puede implicar el esfuerzo de desarrollo adicional en el área de seguridad, para garantizar que los inquilinos no pueden acceder a los datos de otros arrendatarios, incluso en el caso de errores inesperados o ataques.

El procedimiento para la restauración de los datos para un inquilino es similar a la del enfoque de esquema común, con la complicación adicional de que las filas individuales en la base de datos de producción se deben eliminar y, a continuación reinsertados de la base de datos temporal. Si hay un gran número de filas en las tablas afectadas, esto puede hacer que el rendimiento de sufrir notablemente para todos los inquilinos que la base de datos sirve.

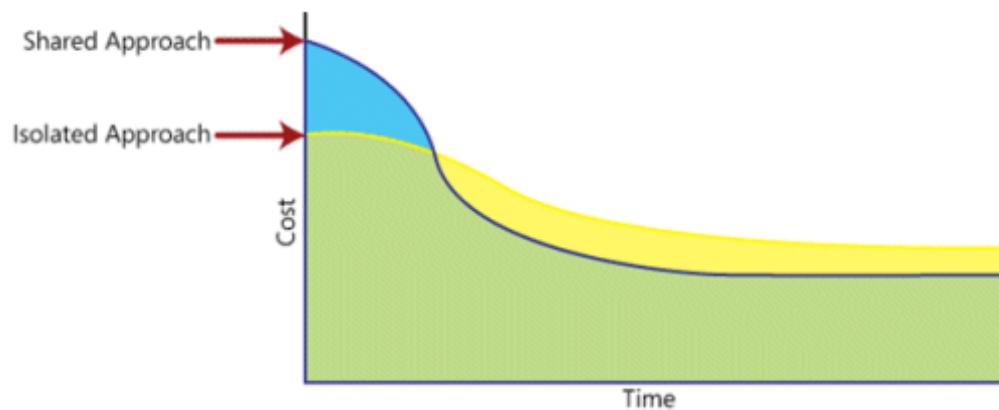
El enfoque de esquema compartido es apropiado cuando es importante que la aplicación sea capaz de servir a un gran número de inquilinos con un pequeño número de servidores y clientes potenciales están dispuestos a entregar el aislamiento de datos a cambio de los costos más bajos que este enfoque hace posible .

## La elección de un enfoque

Cada uno de los tres enfoques descritos anteriormente ofrece su propio conjunto de ventajas y las desventajas que lo hacen un modelo apropiado a seguir en algunos casos y en otros no, según lo determinado por una serie de consideraciones comerciales y técnicas. Algunas de estas consideraciones se enumeran a continuación.

### Consideraciones económicas

Aplicaciones optimizadas para un enfoque compartido tienden a requerir un esfuerzo de desarrollo más grande que las aplicaciones diseñadas utilizando un enfoque más aislado (a causa de la relativa complejidad de desarrollar una arquitectura compartida), resultando en mayores costos iniciales. Debido a que pueden apoyar más inquilinos por servidor, sin embargo, sus costos operativos en curso tienden a ser más bajos.



**Figura 4. Costo en el tiempo para un par hipotético de aplicaciones SaaS; se utiliza un enfoque más aislado, mientras que la otra utiliza un enfoque más compartido**

Su esfuerzo de desarrollo puede verse limitada por factores económicos y de negocios, que pueden influir en su elección del enfoque. El enfoque de esquema compartido puede llegar a ahorrar dinero en el largo plazo, pero requiere un esfuerzo de desarrollo inicial mayor antes de que pueda empezar a producir ingresos. Si usted es incapaz de financiar un esfuerzo de desarrollo de la dimensión necesaria para construir una aplicación del esquema compartido, o si usted tiene que traer su aplicación al mercado más rápidamente que un esfuerzo de desarrollo a gran escala permitiría, puede que tenga que considerar una más aislada enfoque.

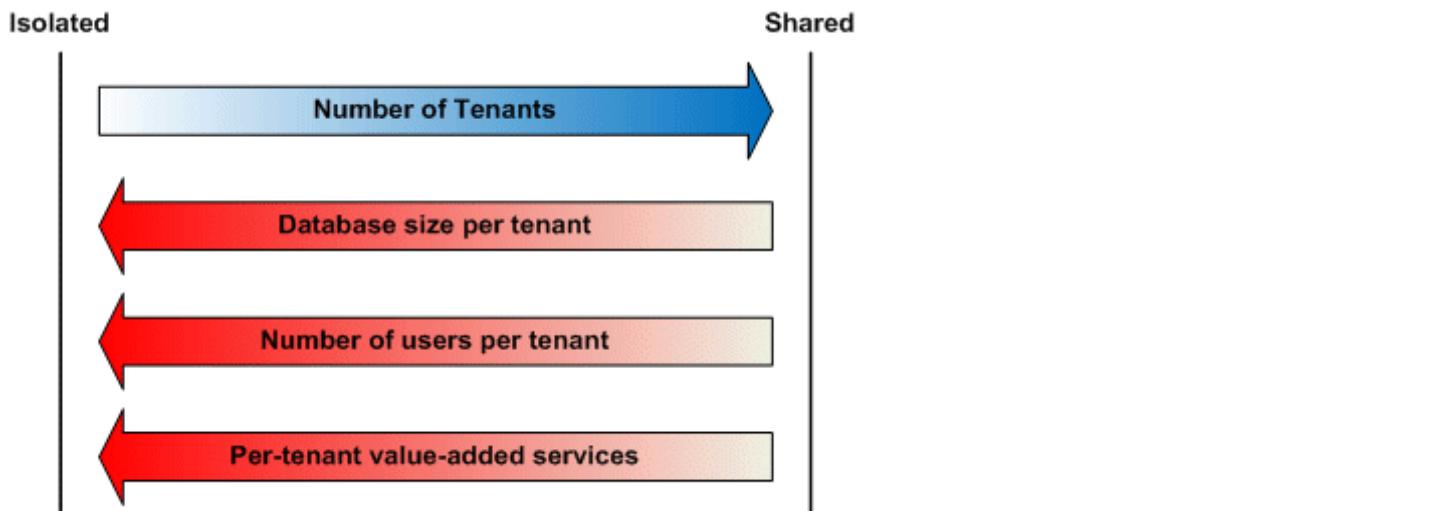
### Consideraciones de Seguridad

A medida que su aplicación va a almacenar los datos sensibles de inquilinos, clientes potenciales tendrán altas expectativas sobre la seguridad, y sus acuerdos de nivel de servicio (SLA) tendrán que proporcionar fuertes garantías de seguridad de datos. Un error común sostiene que sólo el aislamiento físico puede proporcionar un nivel adecuado de seguridad. De hecho, los datos almacenados utilizando un enfoque compartido también pueden proporcionar una fuerte seguridad de los datos, pero requiere el uso de patrones de diseño más sofisticados.

## Consideraciones de inquilinos

El número, la naturaleza y las necesidades de los inquilinos que esperan servir todo esto afecta su decisión arquitectura de datos de diferentes maneras. Algunas de las siguientes preguntas puede que el sesgo hacia un enfoque más aislados, mientras que otros pueden usted sesgo hacia un enfoque más común.

- ¿Cuántas posibles inquilinos qué se puede esperar para apuntar? Es posible que en ninguna parte cerca de ser capaz de estimar el uso prospectivo con autoridad, pero pensar en términos de órdenes de magnitud: ¿Está construyendo una solicitud de cientos de inquilinos? ¿Miles? ¿Decenas de miles? ¿Más? Cuanto mayor sea usted espera que su base de inquilinos a ser, más probable es que tendrá que considerar un enfoque más común.
- La cantidad de espacio de almacenamiento de datos se puede esperar del inquilino promedio para ocupar? Si espera que algunos o todos los inquilinos para almacenar grandes cantidades de datos, el enfoque de base de datos independiente es probablemente el mejor. (De hecho, los requisitos de almacenamiento de datos puede obligar a adoptar un modelo de base de datos independiente de todos modos. Si es así, será mucho más fácil diseñar la aplicación de esa manera desde el principio que para pasar a un enfoque de base de datos separada más adelante.)
- ¿Cuántos usuarios concurrentes final se puede esperar que el inquilino promedio de apoyo? Cuanto mayor sea el número, más un enfoque más aislado más apropiado será para satisfacer las necesidades de los usuarios finales.
- Qué esperas para ofrecer cualquier servicio por inquilinos de valor añadido, tales como copia de seguridad por cada inquilino y capacidad de restauración? Tales servicios son más fáciles de ofrecer a través de un enfoque más aislado.



**Figura 5. Factores relacionadas con el inquilino y cómo afectan "aislados frente compartidos" decisiones de arquitectura de datos**

## Consideraciones reguladoras

Empresas, organizaciones y gobiernos a menudo están sujetos a la ley de regulación que pueden afectar a sus necesidades de seguridad y almacenamiento de registros. Investigar los entornos regulatorios que sus clientes potenciales ocupan en los mercados en los que usted espera para operar, y determinar si presentan consideraciones que afectarán su decisión.

## Consideraciones conjunto de habilidades

El diseño de instancia única, arquitectura multi-inquilino es todavía muy nueva habilidad, por lo expertos en la materia puede ser difícil de conseguir. Si los arquitectos y personal de apoyo no tienen una gran cantidad de aplicaciones SaaS experiencia en la construcción, que tendrán que adquirir los conocimientos necesarios, o que tendrán que contratar a personas que ya cuentan con ella. En algunos casos, un enfoque más aislado puede permitir a su personal para aprovechar más en sus conocimientos existentes de desarrollo de software tradicional que un enfoque más común haría.

## Al darse cuenta de multi-Tenant Arquitectura de Datos

El resto de este artículo se detalla una serie de patrones que pueden ayudar a planificar y construir su aplicación SaaS. Como ya

comentamos en [nuestro artículo de introducción](#), una aplicación SaaS bien diseñado se distingue por tres cualidades: *escalabilidad*, *configurabilidad* y *eficiencia de múltiples usuarios*. La siguiente tabla muestra los patrones apropiados para cada uno de los tres enfoques, divididos en secciones que representan estas tres cualidades.

La optimización de la eficiencia de múltiples usuarios en un entorno compartido no debe poner en peligro el nivel de seguridad de la salvaguardia de acceso a datos. Los patrones de seguridad que figuran a continuación muestran cómo se puede diseñar una aplicación con el "aislamiento virtual" a través de mecanismos tales como permisos, vistas SQL, y el cifrado.

Configurabilidad permite SaaS inquilinos para alterar la apariencia de la aplicación y se comporta sin necesidad de una instancia de solicitud por separado para cada inquilino individual. Los patrones de extensibilidad describen posibles formas de implementar un modelo de datos que los inquilinos pueden ampliar y configurar individualmente para satisfacer sus necesidades.

El método que elija para su arquitectura de datos de la aplicación SaaS afectará a las opciones disponibles para usted para aumentar la escala para dar cabida a más inquilinos o uso pesado. Los patrones de escalabilidad frente a los diferentes retos que plantea la ampliación de bases de datos compartidas y bases de datos dedicados.

**Tabla 1. Patrones apropiados para SaaS Aplicación**

Enfoque	Los patrones de seguridad	Patrones de extensibilidad	Patrones de escalabilidad
Las bases de datos separadas	<ul style="list-style-type: none"> <li>• Conexiones de base de confianza</li> <li>• Tablas de base de datos segura</li> <li>• Cifrado de datos inquilino</li> </ul>	<ul style="list-style-type: none"> <li>• Columnas personalizadas</li> </ul>	<ul style="list-style-type: none"> <li>• Single Tenant ScaleOut</li> </ul>
Base de datos compartida, esquemas separados	<ul style="list-style-type: none"> <li>• Conexiones de base de confianza</li> <li>• Tablas de base de datos segura</li> <li>• Cifrado de datos inquilino</li> </ul>	<ul style="list-style-type: none"> <li>• Columnas personalizadas</li> </ul>	<ul style="list-style-type: none"> <li>• La partición horizontal basada en inquilinos</li> </ul>
Base de datos compartida, esquema compartido	<ul style="list-style-type: none"> <li>• Conexiones de base de confianza</li> <li>• Arrendatario filtro de vista</li> <li>• Cifrado de datos inquilino</li> </ul>	<ul style="list-style-type: none"> <li>• Los campos preasignado</li> <li>• Pares nombre-valor</li> </ul>	<ul style="list-style-type: none"> <li>• La partición horizontal basada en inquilinos</li> </ul>

## Los patrones de seguridad

La construcción de una seguridad adecuada en todos los aspectos de la aplicación es una tarea de suma importancia para cualquier arquitecto SaaS. La promoción de software como servicio, básicamente, significa pedir a los clientes potenciales den una mayor libertad de sus datos de negocio. Dependiendo de la aplicación, esto puede incluir información extremadamente sensible acerca de las finanzas, secretos comerciales, datos de los empleados, y más. Una aplicación SaaS segura es aquella que

proporciona una *defensa en profundidad*, utilizando múltiples niveles de defensa que se complementan entre sí para proporcionar protección de datos de diferentes maneras, en diferentes circunstancias, contra las amenazas internas y externas.

La construcción de la seguridad en una aplicación SaaS significa mirar a la aplicación en los diferentes niveles y pensando acerca de dónde se encuentran los riesgos y la forma de abordarlos. Los patrones de seguridad discutidos en esta sección se basan en tres patrones subyacentes para proporcionar el tipo correcto de la seguridad en los lugares adecuados:

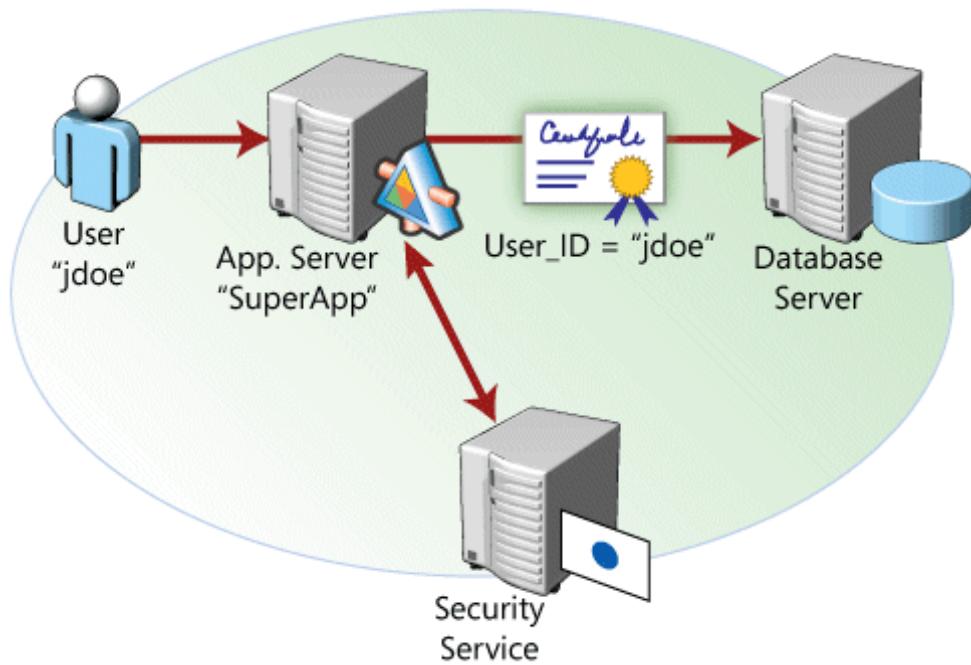
- **Filtrado:** El uso de una capa intermedia entre un inquilino y una fuente de datos que actúa como un colador, haciendo que parezca que el inquilino como si sus datos son los únicos datos en la base de datos.
- **Permisos:** El uso de listas de control de acceso (ACL) para determinar quién puede acceder a los datos en la aplicación y lo que pueden hacer con él.
- **Encriptación:** Oscurecimiento de los datos críticos de todos los inquilinos por lo que seguirá siendo inaccesibles a personas no autorizadas, incluso si entran en posesión de ella.

Mantener estos patrones en cuenta al leer el resto de esta sección.

### Conexiones de base de confianza

En múltiples niveles arquitectos de aplicaciones de entorno de aplicaciones tradicionalmente utilizar dos métodos para asegurar el acceso a los datos almacenados en bases de datos: *la suplantación*, y una *cuenta de subsistema de confianza*.

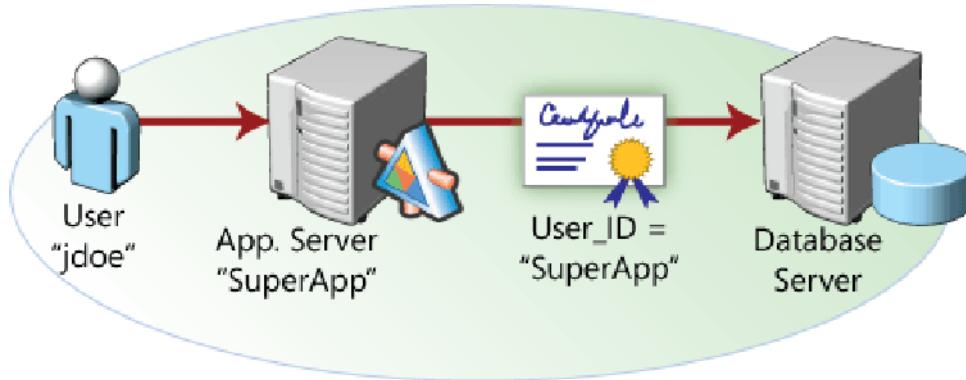
Con el método de acceso suplantación, la base de datos está configurado para permitir a los usuarios individuales para acceder a diferentes tablas, vistas, consultas, procedimientos almacenados y otros objetos de la base. Cuando un usuario final realiza una acción que requiere directa o indirectamente una llamada a una base de datos, la aplicación se presenta a la base de datos *como ese usuario*, literalmente, hacerse pasar por el usuario a efectos de acceder a la base de datos. (En términos técnicos, la aplicación emplea del usuario *contexto de seguridad*). Un mecanismo como delegación Kerberos se puede utilizar para permitir que el proceso de aplicación se conecte a la base de datos en nombre del usuario.



**Figura 6. Una aplicación se conecta a una base de datos utilizando la suplantación**

Con el método de acceso de subsistema de confianza, la aplicación siempre se conecta a la base de datos utilizando su propia identidad proceso de aplicación, independiente de la identidad del usuario; a continuación, el servidor concede el acceso a las aplicaciones de los objetos de la base de que la aplicación puede leer o manipular. Ninguna seguridad adicional debe ser implementada dentro de la propia aplicación para evitar que los usuarios finales individuales tengan acceso a todos los objetos de base de datos que no deben ser expuestos a ellos. Este enfoque hace más fácil la gestión de seguridad, lo que elimina la necesidad de configurar el acceso a los objetos de base de datos en una base por usuario, pero eso significa renunciar a la

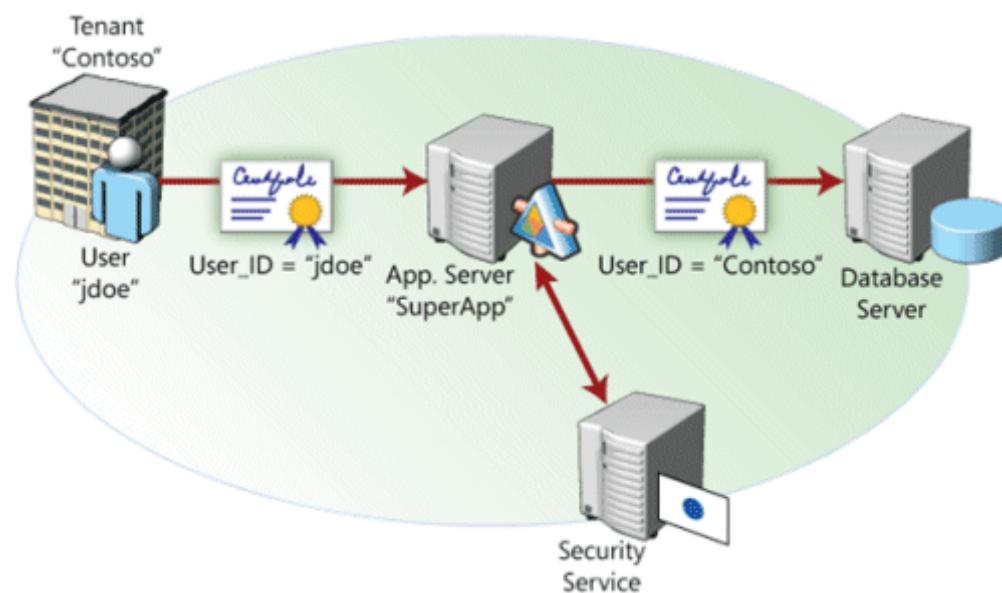
capacidad de asegurar los objetos de base de datos para los usuarios individuales.



**Figura 7. Una aplicación se conecta a una base de datos como un subsistema de confianza**

En una aplicación SaaS, el concepto de "usuarios" es un poco más complicado que en las aplicaciones tradicionales, debido a la distinción entre un inquilino y un usuario final. El inquilino es una organización que utiliza la aplicación para acceder a su propio almacén de datos, que es lógicamente aislado de almacenes de datos que pertenecen a otros inquilinos. Que otorga cada uno de inquilinos el acceso a la aplicación a uno o más usuarios finales, lo que les permite acceder a una parte de los datos del inquilino utilizando cuentas de usuario final controlados por el inquilino.

En este escenario, se puede utilizar un enfoque híbrido de acceso de datos que combina aspectos tanto de la suplantación y métodos de acceso subsistema de confianza. Esto le permite tomar ventaja de los mecanismos de seguridad nativas del servidor de base de datos para hacer cumplir el máximo aislamiento lógica de los datos del arrendatario sin crear un modelo de seguridad compleja unworkably.



**Figura 8. A la aplicación SaaS se conecta a una base de datos utilizando una combinación de la suplantación y enfoques subsistema de confianza**

Este enfoque implica la creación de una cuenta de acceso a base de datos para cada inquilino, y el uso de las ACL para conceder cada uno de estos inquilino cuentas de acceso a la base de datos de objetos se permite que el arrendatario de usar. Cuando un usuario final realiza una acción que requiere directa o indirectamente una llamada a una base de datos, la aplicación utiliza credenciales asociadas con la *cuenta del arrendatario*, en lugar de credenciales asociadas con el usuario final. (Una forma de aplicación para obtener las credenciales apropiadas es a través de la suplantación, en conjunción con un sistema de credenciales como Kerberos. Un segundo enfoque es utilizar un servicio de token de seguridad que devuelve un conjunto real de las credenciales de inicio de sesión cifradas establecidos para el inquilino, que la proceso de aplicación puede presentar a la base de datos.) el servidor de base de datos no distingue entre las solicitudes procedentes de diferentes usuarios finales asociados a un

mismo arrendatario, y subvenciones todas las solicitudes acceso a los datos del inquilino. Dentro de la propia aplicación, código de seguridad impide que los usuarios finales reciban y la modificación de los datos que no tienen derecho a acceder.

Por ejemplo, considere un usuario final de una aplicación de gestión de relaciones con los clientes (CRM) que realiza una operación que realiza consultas a la base de datos de registros de clientes que coincidan con una determinada cadena. La aplicación envía la consulta a la base de datos utilizando el contexto de seguridad del inquilino, por lo que en lugar de devolver todos los registros coincidentes en la base de datos, la consulta sólo recupera las filas coincidentes de las tablas se permite que el inquilino de acceso. Hasta ahora, todo bien, pero supongamos que la función del usuario final sólo le permite acceder a los registros de los clientes localizados dentro de una determinada región geográfica. (Para obtener más información sobre las funciones, consulte la sección "Autorización" en [Arquitectura Estrategias para la captura de la larga cola](#), el primer artículo de esta serie.) La solicitud debe interceptar los resultados de la consulta y sólo presentar al usuario los registros que tiene derecho a ver.

### Tablas de base de datos segura

Para asegurar una base de datos en el nivel de tabla, utilice el comando GRANT de SQL para conceder a un usuario acceso a la cuenta inquilino a una mesa u otro objeto de base de datos:

```
GRANT SELECT, UPDATE, INSERT, DELETE ON [TableName] para [nombre de usuario]
```

Esto agrega la cuenta de usuario a la ACL para la tabla. Si se utiliza el enfoque híbrido para el acceso de base de datos se discutió anteriormente, en el que los usuarios finales están asociados con los contextos de seguridad de sus respectivos inquilinos, esto sólo hay que hacer una vez, durante el proceso de aprovisionamiento arrendatario; las cuentas de usuario final creado por el inquilino podrán acceder a la tabla.

Este modelo es adecuado para su uso con las bases de datos-independiente y separada de esquema de enfoques. En el enfoque de base de datos independiente, puede aislar de datos con sólo restringir el acceso a nivel de toda la base de datos para el inquilino asociado con esa base de datos, aunque también se puede utilizar este modelo en el nivel de tabla para crear otro nivel de seguridad.

### Arrendatario filtro de vista

SQL **vistas** pueden utilizarse para conceder acceso inquilinos individualmente a algunos de las filas de una tabla dada, mientras que lo que les impide acceder a otras filas.

En SQL, una vista es una tabla virtual definido por los resultados de una consulta SELECT. La vista resultante puede ser consultada y se utiliza en procedimientos almacenados como si fuese una tabla de base de datos real. Por ejemplo, la siguiente instrucción SQL crea una vista de una tabla llamada **Los empleados**, que ha sido filtrada de manera que sólo las filas que pertenecen a un solo arrendatario son visibles:

```
CREATE VIEW TenantEmployees AS  
SELECT * FROM Employees WHERE TenantID = SUSER_SID()
```

Esta declaración obtiene el identificador de seguridad (SID) de la cuenta de usuario para acceder a la base de datos (que, como se recordará, es una cuenta que pertenece al **arrendatario**, no el usuario final) y lo utiliza para determinar qué filas deben ser incluidos en la vista. (El ejemplo se supone que el número de ID del cliente único es idéntico al SID del inquilino. Si este no es el caso, uno o más pasos adicionales sería necesario asociar a cada inquilino las filas correctas.) Cuenta de acceso a los datos de cada inquilino individual sería el permiso concedido para utilizar el **TenantEmployees** vista, pero no otorga permisos al **Empleados** misma tabla de origen. Usted puede construir consultas y procedimientos compartidos para aprovechar las vistas, que proporciona los inquilinos con la aparición de aislamiento de datos, incluso dentro de una base de datos de múltiples usuarios.

Este patrón es ligeramente más complejo que el modelo de base de datos Tablas seguro, pero es una manera apropiada para asegurar los datos de inquilinos en una aplicación de esquema común, en el que varios clientes comparten el mismo conjunto

de tablas.

## Cifrado de datos inquilino

Una manera de proteger aún más los datos inquilino es mediante *el cifrado* dentro de la base de datos, para que los datos queden bien cerrados, incluso si cae en las manos equivocadas.

Los métodos criptográficos se clasifican como sea *simétrica* o *asimétrica*. En la criptografía simétrica, una *clave* se genera que se utiliza para cifrar y descifrar datos. Los datos cifrados con una clave simétrica se pueden descifrar con la misma clave. En la criptografía asimétrica (también llamada criptografía de clave pública), se utilizan dos claves, designan la *clave pública* y la *clave privada*. Los datos que se cifrados con una clave pública dada sólo pueden ser descifrados con la clave privada correspondiente, y viceversa. En general, las claves públicas se distribuyen a cualquier y todas las partes interesadas en la comunicación con el titular de la clave, mientras que las claves privadas se llevan a cabo segura. Por ejemplo, si Alice desea enviar un mensaje cifrado a Bob, obtiene la clave pública de Bob a través de algún modo reconocido de medios, y lo utiliza para cifrar el mensaje. El mensaje cifrado resultante, o *texto cifrado*, sólo pueden ser descifrados por alguien en posesión de la clave privada de Bob (en la práctica, esto debería ser sólo Bob). De esta manera, Bob nunca tiene que compartir su clave privada con Alice. Para enviar un mensaje a Bob mediante el cifrado simétrico, Alice tendría que enviar la clave simétrica por separado, que corre el riesgo de que la clave podría ser interceptada por un tercero durante la transmisión.

Criptografía de clave pública requiere significativamente más potencia de cálculo de la criptografía simétrica; un fuerte par de claves puede tener cientos o incluso miles de veces más tiempo para cifrar y descifrar datos como una clave simétrica de calidad similar. Para aplicaciones SaaS en el que se cifra cada pedazo de datos almacenados, la sobrecarga de procesamiento resultantes pueden volver criptografía de clave pública no factible como solución global. Un mejor enfoque consiste en utilizar una *envoltura clave* sistema que combina las ventajas de ambos sistemas.

Con este enfoque, tres claves se crean para cada inquilino como parte del proceso de aprovisionamiento: una clave simétrica y un par de claves asimétricas que consiste en una clave pública y una clave privada. La clave simétrica más eficientes se utiliza para cifrar los datos críticos del inquilino para su almacenamiento. Para añadir otra capa de seguridad, un par de claves pública / privada se utiliza para cifrar y descifrar la clave simétrica, para mantenerlo a salvo de cualquier intruso potencial.

Cuando un usuario final inicia sesión, la aplicación utiliza suplantación para acceder a la base de datos utilizando el contexto de seguridad del inquilino, que concede el acceso al proceso de solicitud de la clave privada del arrendatario. La aplicación (aún hacerse pasar por el inquilino, por supuesto) a continuación, puede utilizar la clave privada del inquilino para descifrar la clave simétrica del inquilino y usarlo para leer y escribir datos.

Este es otro ejemplo del principio de defensa en profundidad en acción. La exposición accidental o maliciosa de los datos del arrendatario a otros inquilinos de un escenario de pesadilla para el SaaS preocupados por la seguridad y el proveedor se impidió en múltiples niveles. La primera línea de defensa, a nivel de base de datos, evita que los usuarios finales tengan acceso a los datos privados de otros inquilinos. Si un insecto o un virus en el servidor de base de datos eran para provocar una fila incorrecta para ser entregado al inquilino, el contenido cifrado de la fila sería inútil sin acceso a la clave privada del arrendatario.

La importancia de cifrado aumenta cuanto más cerca una aplicación SaaS es hasta el final "compartida" de la serie continua aislada / compartido. El cifrado es especialmente importante en situaciones que involucran datos de alto valor o preocupaciones de privacidad, o cuando varios inquilinos comparten el mismo conjunto de tablas de base de datos.

Porque no se puede índice de cifrado columnas, la selección de las columnas de las tablas para cifrar implica la realización de un compromiso entre la seguridad de los datos y el rendimiento. Piense acerca de los usos y la sensibilidad de los distintos tipos de datos en el modelo de datos al tomar decisiones sobre el cifrado.

## Patrones de extensibilidad

Tal como fue diseñado, su solicitud incluirá, naturalmente, una configuración de base de datos estándar, con tablas predeterminadas, campos, consultas y relaciones que sean apropiadas a la naturaleza de su solución. Sin embargo, diferentes organizaciones tienen sus propias necesidades únicas que una, no extensible modelo de datos predeterminado rígida no será capaz de hacer frente. Por ejemplo, un cliente de un sistema de trabajo de seguimiento de SaaS podría tener que almacenar una cadena de código de clasificación generada externamente con cada registro para integrar totalmente el sistema con sus otros procesos. Un cliente puede tener diferentes sin necesidad de un campo de cadena de clasificación, pero podría requerir apoyo

para el seguimiento de un número de identificación de la categoría, un número entero. Por lo tanto, en muchos casos, tendrá que desarrollar e implementar un método por el cual los clientes pueden ampliar su modelo de datos por defecto para satisfacer sus necesidades, sin afectar el modelo de datos que utilizan otros clientes.

### Los campos preasignado

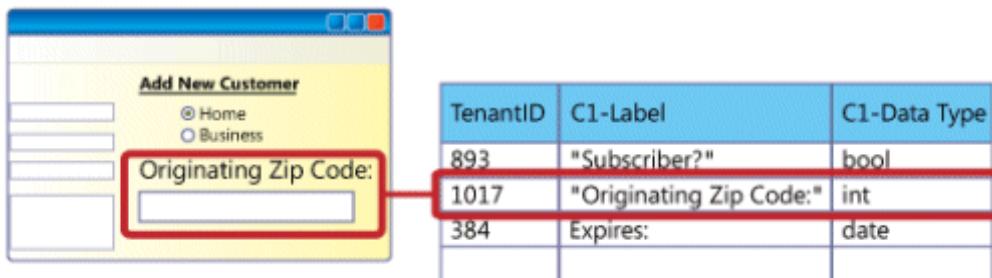
Una forma de hacer que su modelo de datos extensible es simplemente crear un número predeterminado de campos personalizados en cada tabla que desea permitir que los inquilinos se extiendan.

TenantID	FirstName	BirthDate	C1	C2	C3
345	Ted	1970-07-02	null	"Paid"	null
777	Kay	1956-09-25	"66046"	null	null
1017	Mary	1962-12-21	null	null	null
345	Ned	1940-03-08	null	"Paid"	null
438	Pat	1952-11-04	null	"San Francisco"	"Yes"

**Figura 9. Una mesa con un conjunto predeterminado de campos personalizados, etiquetado C1 a C3**

En la figura anterior, los registros de diferentes clientes se entremezclan en un solo cuadro; un campo ID del cliente asocia cada registro con un inquilino individual. Además del conjunto estándar de los campos, se proporcionan una serie de campos personalizados, y cada cliente puede elegir qué usar estos campos y cómo los datos serán recogidos por ellos.

¿Qué pasa con los tipos de datos? Usted simplemente puede elegir un tipo de datos común para cada campo personalizado se crea, pero los clientes es probable encontrar este enfoque innecesariamente restrictiva, ¿y si un cliente tiene una necesidad de tres campos de cadena adicionales y sólo ha proporcionado un campo de cadena, un entero campo, y un campo booleano? Una manera de proporcionar este tipo de flexibilidad es utilizar el tipo de datos de cadena para cada campo personalizado, y el uso de metadatos para realizar un seguimiento del tipo de datos "real" del inquilino deseada utilizar.



The diagram illustrates a 'Customer' entity with fields for FirstName, LastName, BirthDate, and a custom field 'Originating Zip Code'. A red box highlights the 'Originating Zip Code' input field. To its right is a 'C1' metadata table:

TenantID	C1-Label	C1-Data Type
893	"Subscriber?"	bool
1017	"Originating Zip Code:"	int
384	Expires:	date

A red box highlights the row for TenantID 1017, indicating it is the definition for the custom field.

**Figura 10. Un campo personalizado en una página web, que se define por una entrada en una tabla de metadatos**

En el ejemplo anterior, el inquilino ha utilizado las características de extensibilidad de la aplicación para agregar un cuadro de texto llamado "Código postal de origen" a una pantalla de entrada de datos, y se asigna el cuadro de texto a un campo personalizado llamado C1. Al crear el cuadro de texto, la lógica de validación utilizado inquilino (no se muestra) para exigir que el cuadro de texto contiene un número entero. Tal como se aplica, este campo personalizado se define por un registro en una tabla de metadatos que incluye el número del inquilino ID único (1017), la etiqueta que el inquilino haya elegido para el campo ("originario Código postal"), y el tipo de datos que el inquilino quiere para usar para el campo ("int").

Puede realizar un seguimiento de las definiciones de campo para todos los campos personalizados de la aplicación en una sola tabla de metadatos, o utilizar una tabla separada para cada campo personalizado; por ejemplo, una mesa "C1" definiría C1 campo personalizado para todos los inquilinos que lo utiliza, una mesa "C2" haría lo mismo para C2 campo personalizado, y así sucesivamente.

TableID	C1-Label	C1-DataType	C2-Label	C2-DataType
893	"Subscriber?"	bool	"Subscription Code"	string
1017	"Originating Zip Code:"	int	null	null
564	"Expires"	date	"Auto Review?"	bool

Table: C1ExtensionTable

TableID	C1-Label	C1-DataType
893	"Subscriber?"	bool
1017	"Originating Zip Code:"	int
564	"Expires"	date

Table: C2ExtensionTable

TableID	C2-Label	C2-DataType
893	"Subscription Code"	string
564	"Auto Review?"	bool

**Figura 11. Almacenamiento de definiciones de campo en una sola tabla de metadatos, parte superior, y en tablas separadas para cada campo personalizado**

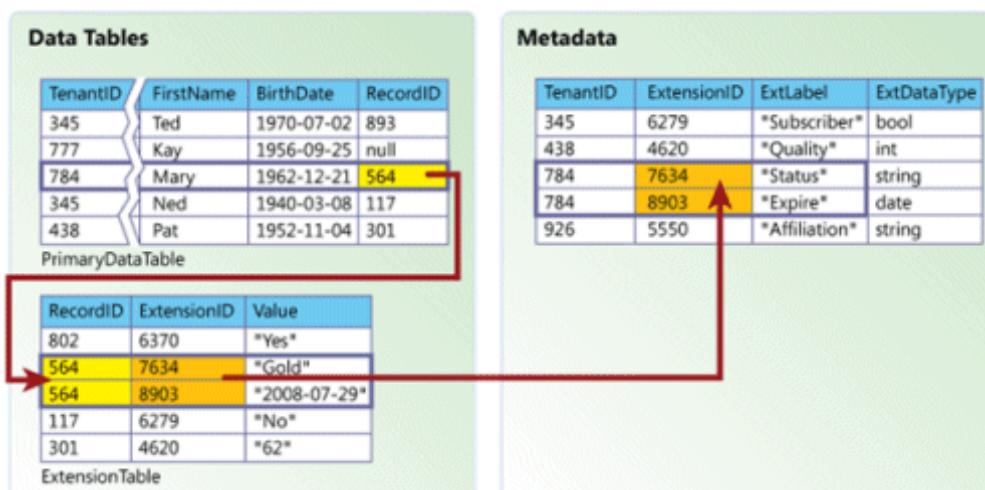
La principal ventaja de utilizar tablas independientes es que cada tabla específica de los campos sólo contiene filas para los inquilinos que utilizan ese campo, lo que ahorra espacio en la base de datos. (Con el enfoque de una sola mesa, todos los inquilinos que utiliza al menos un campo personalizado para crear una fila de la tabla combinada, con campos nulos que representan los campos personalizados disponibles que el inquilino no ha utilizado). La desventaja de usar tablas separadas es que aumenta la complejidad de las operaciones de campo personalizado, lo que requiere el uso de SQL JOIN declaraciones para inspeccionar todas las definiciones de campo personalizado para un solo usuario.

Cuando un usuario escribe finales una cantidad en el campo y guarda el registro, la aplicación arroja el valor de originar Código postal a una cadena antes de crear o actualizar el registro en la base de datos. Siempre que la aplicación recupera el registro, se comprueba la tabla de metadatos para el tipo de datos que desea utilizar y pone el valor en el campo personalizado de nuevo a su tipo original.

### Pares nombre-valor

El patrón preasignados Los campos que se explica en la sección anterior es una forma sencilla de proporcionar un mecanismo para que los arrendatarios ampliar y personalizar el modelo de datos de la aplicación. Sin embargo, este enfoque tiene ciertas limitaciones. Decidir el número de campos personalizados para proporcionar en una tabla dada implica la realización de una solución de compromiso. Muy pocos campos personalizados, y los inquilinos se sientan restringido y limitado por la aplicación; demasiados, y la base de datos se vuelve escasa y derrochador, con muchos campos no utilizados. En casos extremos, ambos pueden ocurrir, con algunos inquilinos menores de los que utilizan los campos personalizados y otros exigiendo aún más.

Una forma de evitar estas limitaciones es permitir a los clientes extender el modelo de datos de forma arbitraria, el almacenamiento de datos personalizados en una tabla separada y el uso de metadatos para definir etiquetas y tipos de datos para los campos personalizados de cada inquilino.

**Figura 12. Una mesa de extensión permite que cada inquilino para definir un número arbitrario de campos personalizados**

A continuación, una tabla de metadatos almacena información importante acerca de cada campo personalizado definido por todos los inquilinos, incluyendo el nombre del campo (etiqueta) y el tipo de datos. Cuando un usuario final guarda un registro con un campo personalizado, suceden dos cosas. En primer lugar, el propio registro se crea o se actualiza en la tabla de datos primarios; Los valores se guardan para todos los campos predefinidos, pero no el campo personalizado. En su lugar, la aplicación crea un identificador único para el registro y la guarda en el campo ID de registro. En segundo lugar, una nueva fila se crea en la tabla de extensión que contiene los siguientes elementos de información:

- El ID del registro asociado en la tabla de datos principal.
- El ID de extensión asociada con la definición de campo personalizado correcto.
- El valor del campo personalizado en el registro que está siendo salvado, echado en una cadena.

Este enfoque permite a cada inquilino para crear tantos campos especiales como sea necesario para satisfacer sus necesidades de negocio. Cuando la aplicación recupera un registro de cliente, se realiza una búsqueda en la tabla de extensión, selecciona todas las filas correspondientes a la ID de registro, y devuelve un valor para cada campo personalizado utilizado. Para asociar estos valores con los campos personalizados correctas y las arrojó sobre los tipos de datos correctos, la aplicación busca la información de campos personalizados en los metadatos utilizando los ID de extensión asociados a cada valor de la tabla de extensión.

Este enfoque hace que el modelo de datos arbitrariamente extensible al tiempo que conserva los beneficios de costo de la utilización de una base de datos compartida. La principal desventaja de este enfoque es que añade un nivel de complejidad de las funciones de base de datos, tales como la indexación, consulta, y los registros de actualización. Este suele ser el mejor enfoque a adoptar si desea utilizar una base de datos compartida, pero también anticipan que sus clientes requieren un alto grado de flexibilidad para extender el modelo de datos predeterminado.

### Columnas personalizadas

El tipo más simple de modelo de datos extensible es una en la que las columnas se pueden añadir a las tablas de los inquilinos directamente.

EmployeeID	FirstName	BirthDate	LastReview	Branch	401k
653	Pat	1952-11-04	null	"San Francisco"	true
1310	Tom	1949-12-14	2006-01-30	"London"	null
280	Surendra	1973-09-12	2005-11-08	"Bangalore"	null
985	Christine	1981-03-26	2006-06-09	"San Francisco"	false
1701	Gordon	1964-08-20	null	"Toronto"	null

**Figura 13. filas personalizadas se pueden agregar a una tabla dedicada sin alterar el modelo de datos para otros inquilinos**

Este patrón es apropiado para aplicaciones de base de datos separado o separada de esquema, ya que cada inquilino tiene su propio conjunto de tablas que se pueden modificar de forma independiente de los que pertenecen a otros clientes. Desde el punto de vista del modelo de datos, este es el más simple de los tres patrones de extensibilidad, ya que no requiere realizar un seguimiento de extensiones de datos por separado. En el lado arquitectura de la aplicación, sin embargo, este patrón puede a veces ser más difíciles de aplicar, ya que permite a los inquilinos para variar el número de columnas en una tabla. Incluso si el patrón de columnas personalizadas está disponible para usted, usted puede considerar el uso de una variación en el patrón preasignados campos o pares nombre-valor para reducir el esfuerzo de desarrollo, que le permite escribir código de la aplicación que puede asumir un número conocido e invariable de campos de cada mesa.

### El uso de las extensiones del Modelo de Datos

Sea cual sea el método que utilice para crear un modelo de datos extensible, que debe estar emparejado con un mecanismo para integrar los campos adicionales en la funcionalidad de la aplicación. Cualquier campo personalizado implementado por un cliente requiere una modificación correspondiente a la lógica de negocio (por lo que la aplicación puede utilizar los datos personalizados), la lógica de presentación (para que los usuarios tienen la posibilidad de insertar los datos personalizados como entrada y recibirla como de salida), o ambos. Por tanto, la interfaz de configuración se presente al cliente debe proporcionar

formas de modificar los tres, preferentemente en forma integrada. (Proporcionar mecanismos mediante los cuales los clientes pueden modificar la lógica de negocio y la interfaz de usuario se abordará en un próximo artículo de esta serie).

## Patrones de escalabilidad

Software de empresa a gran escala está destinado a ser utilizado por miles de personas al mismo tiempo. Si tiene experiencia en aplicaciones de construcción de la empresa de este tipo, que conocer de primera mano los desafíos de crear una arquitectura escalable. Para una aplicación SaaS, la escalabilidad es aún más importante, ya que tendrá que soportar datos pertenecientes a todos sus clientes. Para los proveedores de software independientes (ISV), acostumbrados a la construcción de software de la empresa en las instalaciones, el apoyo a este tipo de base de usuarios es como pasar de las ligas menores a los mayores: las reglas pueden ser familiares, pero el juego se juega en un nivel completamente diferente. En lugar de una, aplicación empresarial crítica para el negocio ampliamente desplegada, en realidad está la construcción de un sistema a escala de Internet que necesita para apoyar activamente una base de usuarios potenciales de numeración en los millones.

Las bases de datos se pueden ampliar (moviendo a un servidor más grande que utiliza procesadores más potentes, más memoria y unidades de disco más rápidas) y escalado a cabo (mediante la partición de una base de datos en varios servidores). Diferentes estrategias son apropiadas cuando se escala una base de datos compartida frente a la ampliación de bases de datos dedicados. (Al desarrollar una estrategia de ampliación, es importante distinguir entre la ampliación de su *aplicación* (el aumento de la carga total de trabajo de la aplicación puede acomodar) y la ampliación de sus *datos* (el aumento de su capacidad para almacenar y trabajar con datos). En este artículo se centra en la ampliación de datos específica.)

## Técnicas de escalamiento

Las dos herramientas principales para utilizar cuando se escala a cabo una base de datos a cabo son *la replicación* y *partición*. La replicación incluye la copia de la totalidad o parte de una base de datos a otra ubicación, y luego mantener la copia o copias sincronizadas con el original. La replicación de maestro único, en el que sólo el original (o *maestro de replicación*) se puede escribir, es mucho más fácil de manejar que la replicación con varios maestros, en los que todas o algunas de las copias puede ser anotado en y se usa algún tipo de mecanismo de sincronización para reconciliar los cambios entre las diferentes copias de los datos.

Particionamiento implica subconjuntos de poda de los datos de una base de datos y mover los datos podados a otras bases de datos u otras tablas en la misma base de datos. Puede dividir una base de datos mediante la reubicación de toda tablas, o mediante el fraccionamiento de una o más tablas arriba en tablas más pequeñas *horizontalmente* o *verticalmente*. Partición horizontal significa que la base de datos se divide en dos o más bases de datos más pequeñas que utilizan el mismo esquema y estructura, pero con un menor número de filas en cada tabla. Partición vertical significa que una o más tablas individuales se dividen en tablas más pequeñas con el mismo número de filas, pero con cada tabla que contiene un subconjunto de las columnas de la original. La replicación y la partición se utilizan a menudo en combinación uno con el otro cuando se escala bases de datos.

## La partición horizontal basada en inquilinos

Una base de datos compartida se debe reducir cuando ya no puede cumplir con las métricas de rendimiento de línea de base, como cuando hay demasiados usuarios intentando tener acceso a la base de datos al mismo tiempo o el tamaño de la base de datos está causando consultas y actualizaciones para tener demasiado tiempo para ejecutar, o cuando el mantenimiento operativo tareas comienzan a afectar a la disponibilidad de datos.

La forma más sencilla de ScaleOut una base de datos compartida es a través de la partición horizontal (en hilera) basado en ID del cliente. SaaS bases de datos compartidas están bien adaptados a la partición horizontal, ya que cada inquilino tiene su propio conjunto de datos, por lo que puede llegar fácilmente a los datos de inquilinos individuales y moverlo.

Sin embargo, no asuma que si usted tiene 100 inquilinos y desea particionar las cinco formas de bases de datos, sólo tiene que contar fuera 20 inquilinos a la vez y moverlos. Diferentes inquilinos pueden colocar radicalmente diferentes demandas de una aplicación, y es importante planificar con cuidado para evitar la simple creación de más pequeño, pero aún sobrecargado, particiones, mientras que otras particiones van infráutilizada.

Si usted está experimentando problemas de rendimiento de aplicaciones debido a demasiados usuarios finales acceden a la base de datos al mismo tiempo, tenga en cuenta la partición de la base de datos para igualar el número total de cuentas de usuario

final activas en cada servidor. Por ejemplo, si su base de datos existente sirve inquilinos A y B con 600 usuarios activos cada uno, y los inquilinos C, D y E con 400 usuarios activos cada uno, se puede particionar la base de datos moviendo los inquilinos C, D, y E a un nuevo servidor ; Ambas bases de datos servirían entonces 1200 usuarios cada uno.

Si experimenta problemas relacionados con el tamaño de la base de datos, tales como la longitud de tiempo que se necesita para realizar consultas, un método de partición más eficaz podría ser la de Tamaño de destino base de datos en su lugar, la asignación de los inquilinos a los servidores de base de datos de una manera tal que se más o menos igualar la cantidad de datos en cada una.

El método de partición que elija puede tener un impacto significativo en el desarrollo de aplicaciones. Independientemente del método que elija, es importante que se puede estudiar con precisión e informar sobre cualquier métrica que va a utilizar para tomar decisiones de partición. La construcción de apoyo para el seguimiento en su aplicación le ayudará a obtener una visión precisa de los patrones de uso y las necesidades de sus inquilinos. Además, es probable que usted tendrá que volver a crear particiones sus datos periódicamente, ya que sus inquilinos evolucionar y cambiar su forma de trabajar. Elija una estrategia de partición que se puede ejecutar cuando sea necesario, sin afectar indebidamente a los sistemas de producción.

De vez en cuando, un inquilino puede tener suficientes usuarios o utilizar datos suficientes para justificar el inquilino se mueve a una base de datos específica propia. Consulte la siguiente sección, "Single Tenant ScaleOut," en busca de ayuda la realización de más de escalamiento.

El patrón de partición horizontal basada en inquilinos es adecuada para su uso con aplicaciones de esquema compartido, que imponen algunas restricciones inusuales en la tarea familiar de escalar una base de datos. Proporciona una manera de escalar una base de datos compartida, evitando acciones que rompa el rendimiento de la aplicación o el daño (como, por ejemplo, la división de los datos de un inquilino a través de dos o más servidores sin darse cuenta o innecesariamente).

## Single Tenant ScaleOut

Si algunos o todos los inquilinos almacenar y utilizar una gran cantidad de datos, bases de datos de inquilinos pueden crecer lo suficiente como para justificar dedicar un servidor completo a una sola base de datos que sirve a un solo arrendatario. Los desafíos de escalabilidad en este escenario son similares a aquellas que enfrentan los arquitectos de aplicaciones tradicionales de un solo inquilino. Con una gran base de datos en un servidor dedicado, la ampliación es la forma más fácil para acomodar el crecimiento continuo.

Si la base de datos sigue creciendo, con el tiempo ya no será rentable para moverlo a un servidor más potente, y usted tendrá que *escalar* mediante la partición de la base de datos a uno o más servidores adicionales. Escalar hacia fuera una base de datos dedicado es diferente que el escalado una responsabilidad compartida. Con una base de datos compartida, el método más eficaz de escalamiento implica mover conjuntos completos de datos inquilino de una base de datos a otro, por lo que la naturaleza del modelo de datos que se utiliza no es particularmente relevante. Cuando se escala una base de datos que está dedicada a un solo inquilino, se hace necesario analizar los tipos de datos que se están almacenando para determinar el mejor enfoque.

El artículo [escalado horizontal de SQL Server 2005](#) contiene una guía y sugerencias sobre el análisis de datos de escalado horizontal adicional. El artículo explica los datos de referencia, datos de actividad y datos de recursos en detalle, da algunas pautas para la replicación y la partición de datos, y explica algunos factores adicionales que afectan ScaleOut. Algunas de las pautas ScaleOut a tener en cuenta:

- **Utilice la replicación para crear copias de sólo lectura de datos que no cambia muy a menudo.** Algunos tipos de datos cambian rara vez o nunca después de introducir los datos, tales como números de referencia o números de seguridad social de los empleados. Otros tipos de datos están sujetos a cambio activo durante un periodo de tiempo determinado y luego archivados, tales como órdenes de compra. Este tipo de datos son candidatos ideales para la replicación de ida a cualquier base de datos de las que podrían ser referenciados.
- **Ubicación, ubicación, ubicación.** Mantenga los datos cerca de otros datos que hace referencia a ella. ("Cerrar" en este sentido significa generalmente lógicamente próxima vez que físicamente próximas, a pesar de la proximidad lógica a menudo implica la proximidad física también.) Tenga en cuenta las relaciones entre los diferentes tipos de datos para decidir si debe separarlos, y utilizar la replicación para distribuir sólo lectura copias de los datos de referencia entre las diferentes bases de datos cuando sea apropiado.

Por ejemplo, si el acto de recuperar un registro de cliente implica habitualmente la selección de las últimas órdenes de compra del cliente a partir de una tabla diferente, trate de mantener las dos tablas en la misma base de datos o utilizar la replicación para crear copias de tipos apropiados de datos. Tratar de encontrar divisiones naturales en los datos que reduzca al mínimo la cantidad de comunicación entre bases de datos que necesita para tomar su lugar. Por ejemplo, los datos asociados con lugares particulares a menudo pueden dividirse geográficamente.

- **Identificar los datos que no deben ser particionado.** Datos de recursos, como los niveles de inventario de almacén, por lo general son buenos candidatos para la replicación o partición. Utilizar técnicas de ScaleOut para mover otros datos fuera del servidor, dejando sus datos de recursos más espacio para crecer. Si ha movido todos los datos que pueda, pero experimentar problemas, considere la ampliación a un servidor más grande para los datos de recursos.
- **Utilizar la replicación de un solo maestro siempre que sea posible.** Cambios a sincronizarse con múltiples copias de los mismos datos es difícil, por lo que evitar el uso de la replicación multi-master si es posible. Cuando se deben cambiar los datos replicados, sólo permiten que los cambios se escriben en la copia maestra.

Este patrón se puede aplicar a los tres enfoques, pero sólo entra en juego cuando las necesidades de datos de un inquilino individual no pueden ser acomodados por un único servidor. Con el enfoque de base de datos independiente, si las necesidades de almacenamiento de datos de los inquilinos son modestos, cada servidor individual podría albergar docenas de bases de datos; en ese caso escalar un servidor en particular implica simplemente moviendo una o más bases de datos a un nuevo servidor y modificar los metadatos de la aplicación para reflejar la nueva ubicación de los datos.

## Conclusión

Los enfoques de diseño y los patrones que hemos discutido en este artículo deben ayudar a crear la capa base de la confianza que es vital para el éxito de su aplicación SaaS. El diseño de una arquitectura de datos SaaS que concilie los beneficios y las demandas de uso compartido y aislamiento de la competencia no es una tarea trivial, pero estos enfoques y modelos debería ayudar a identificar y resolver muchas de las cuestiones críticas que se enfrentará. Las ideas y recomendaciones que aquí se presentan difieren en los detalles, pero todos ellos ayudan a aprovechar los principios de capacidad de configuración, la escalabilidad y la eficiencia de múltiples usuarios para diseñar una arquitectura de datos segura y extensible para una aplicación SaaS.

Este artículo es de ninguna manera la última palabra en una sola instancia, la arquitectura de datos de múltiples usuarios. Más adelante en esta serie, vamos a ver formas en que puede ayudar a los inquilinos poner sus extensiones modelo de datos para un buen uso a través de la presentación y la personalización del flujo de trabajo.

## Orientación relacionada

[Desarrollo de aplicaciones multi-arrendatario de la nube de Windows Azure](#)

## Realimentación

Los autores dan la bienvenida con gusto sus comentarios acerca de este documento. Envíe un correo electrónico a todos los comentarios que [fredch@microsoft.com](mailto:fredch@microsoft.com) , [gianpc@microsoft.com](mailto:gianpc@microsoft.com) , o [rwlter@microsoft.com](mailto:rwlter@microsoft.com) . Gracias.

© 2016 Microsoft