Actualización de la base de datos

Business Components. Reglas, eventos y chequeos. Repaso

GeneXus™

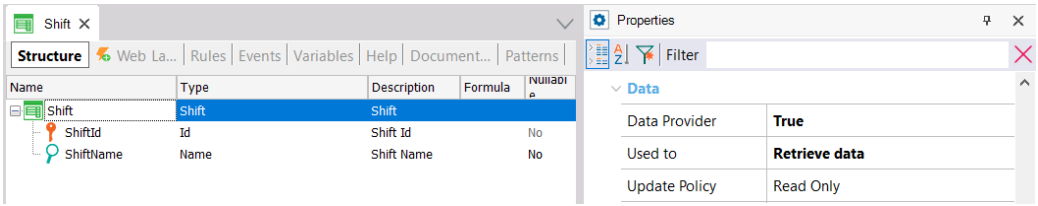
Insert, Update, Delete

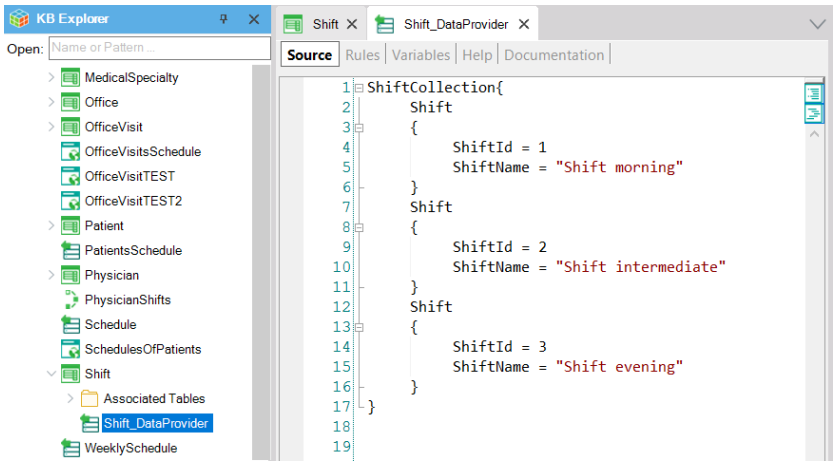
Business Component

KB Hospital

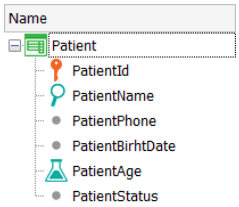
Vamos a integrar lo que sabemos de actualización de la base de datos a través de Business Components y a agregar algún conocimiento un poco más avanzado.

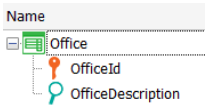
Para ello vamos a centrarnos en una KB que estamos desarrollando para un servicio de salud...

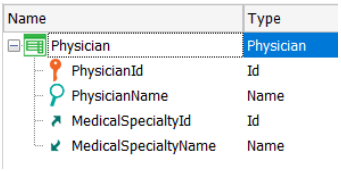


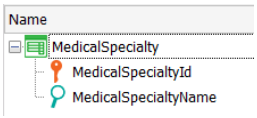


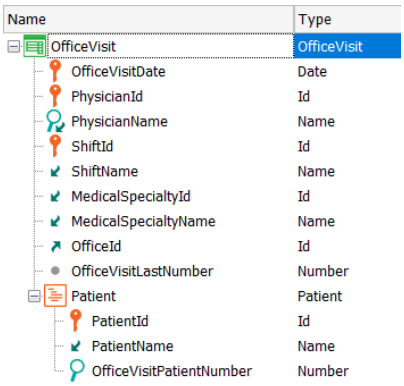
... que ofrece a sus pacientes la posibilidad de realizar consultas médicas en uno de tres turnos posibles: matutino, intermedio o vespertino. Para representar esos turnos tenemos la transacción Shift que es dinámica (vemos que prendimos la propiedad Data Provider y dijimos que era para recuperar datos, y solo recuperarlos, porque vemos que dejamos aquí Read Only. Aquí indicamos los 3 turnos.

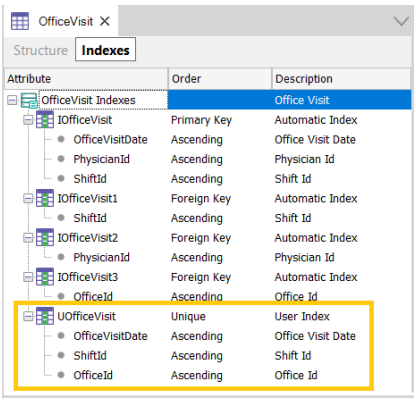








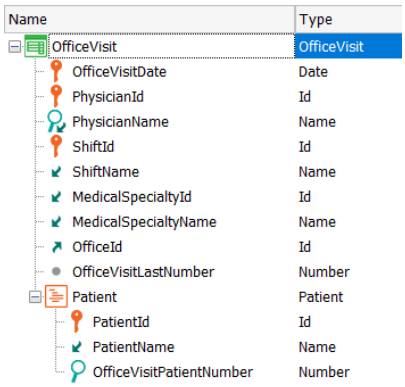


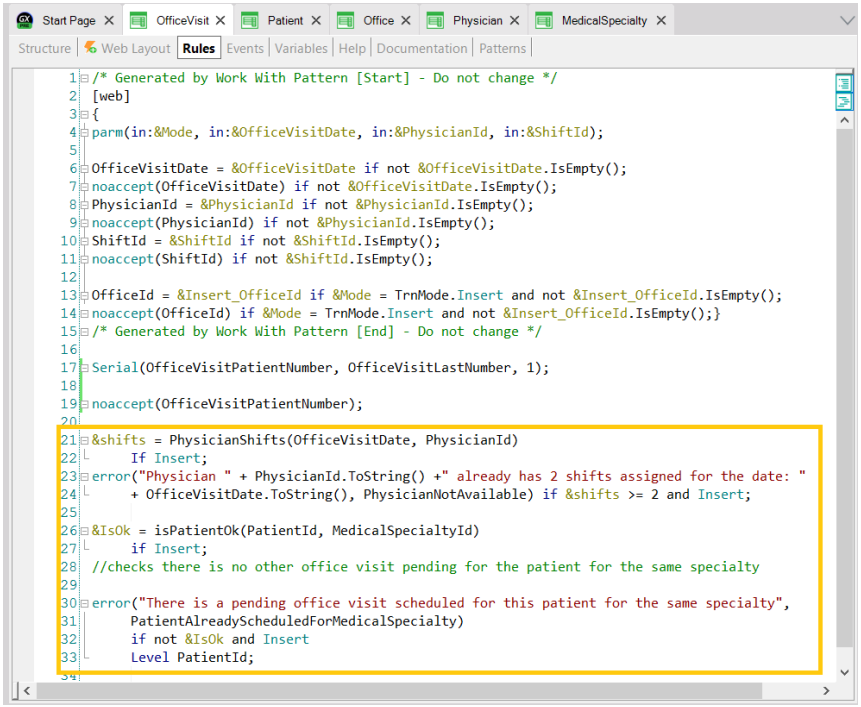


Por otro lado tenemos la transacción Patient para registrar a los pacientes, Office para registrar los consultorios en los que se realizarán esas consultas médicas, Physician para registrar la información de cada médico, que tiene una (y solo una) especialidad médica, y por último, y en esta nos vamos a centrar, la transacción OfficeVisit que es la que registra cada consulta en la que un médico atiende en una fecha y turno (matutino, intermedio o vespertino) a una lista de pacientes.

Como a los pacientes se les quiere asignar un número para ser atendidos, se agrega en el primer nivel un atributo que registra el último número dado, de manera de poder utilizar una regla serial cada vez que se ingrese un nuevo paciente a través de la transacción.

También es necesario indicar en qué consultorio será realizada la consulta. Ya podemos observar que si bien definimos que el trío "fecha de visita, médico y turno" no puede repetirse (constituye la clave primaria), tampoco deberían repetirse fecha de visita, turno y consultorio (es decir, no puede haber dos consultas en una misma fecha, turno y consultorio). Por tanto estos tres atributos forman una clave candidata. Por esta razón creamos un índice unique para representarlo (y asegurarlo).





Además querremos agregar dos reglas de negocio: un mismo doctor puede atender hasta en dos turnos en el mismo día, pero no tres.
Y por otro lado no se debe permitir ingresar un paciente que ya tenga una consulta pendiente para un médico de la misma especialidad.

Las chequeamos así:
Aquí estamos llamando a este procedimiento solo si se está en modo Insert; en él se cuenta la cantidad de consultas del médico en la fecha en la que estamos queriendo asignar una nueva consulta. Y dispararemos un error si lo que devuelve es 2 o más.

Aquí tenemos la regla serial para que cuando se está ingresando un nuevo paciente para la visita se le asigne el siguiente número para ser atendido. Y aquí deshabilitamos el campo, porque no querremos que el usuario lo modifique.

Y aquí estamos llamando a otro procedimiento que chequea si el paciente tiene alguna consulta pendiente para la misma especialidad de la consulta a la que se lo está intentando agendar. En ese caso se dispara un error.

Vemos que los procedimientos y las reglas de error solo se ejecutan cuando se está queriendo **insertar**: cabezal para un caso y línea para el otro.

Además, observemos que como le hemos aplicado el pattern WorkWith a la transacción, éste automáticamente agregó estas reglas (entre ellas, la

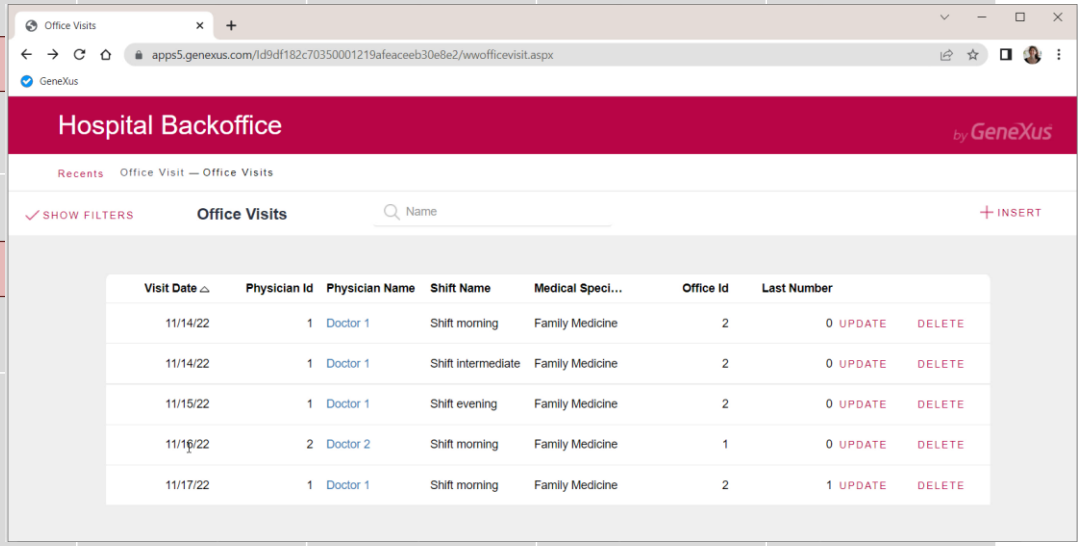
parm para recibir clave y modo).

Nuestro objetivo será repasar y profundizar en qué de todo esto se ejecuta cuando se hacen las operaciones a través del Business Component.

	Monday	Tuesday	Wednesday	Thursday	Friday
Shift 1 morning	Physician: 1 Office: 2		Physician: 2 Office: 1	Physician: 1 Office: 2 Patient: 2	
Shift 2 intermediate	Physician: 1 Office: 2				
Shift 3 evening		Physician: 1 Office: 2			

Asumamos que tenemos estos datos en la base de datos para la semana próxima, sin ningún paciente aún agendado salvo para la consulta del jueves, que tiene al paciente 2.

	Monday	Tuesday	Wednesday	Thursday	Friday
Shift 1 morning	Physician: Office:				
Shift 2 intermediate	Physician: Office:				
Shift 3 evening					



Aquí los vemos en el Work With.

Querremos insertar una consulta nueva para el lunes, y probaremos que funcione como esperamos en todos los casos.

Checks	Error Id	Error Description
FK: PhysicianId	ForeingKeyNotFound	No matching 'Physician"
`error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: " + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;`	PhysicianNotAvailable	Physician... already has 2 shifts...
FK: ShiftId	ForeingKeyNotFound	No matching 'Shift'
FK: OfficeId	ForeingKeyNotFound	No matching 'Office'
CK: {OfficeVisitDate, ShiftId, OfficeId}		Office Visit Date, Shift Id, Office Id already exists
PK: {OfficeVisitDate, PhysicianId, ShitfId}	DuplicatePrimaryKey	Record already exists
FK: PatientId	ForeingKeyNotFound	No matching 'Patient'
`error("There is a pending office visit scheduled for this patient for the same specialty", PatientAlreadyScheduledForMedicalSpecialty) if not &IsOk and Insert Level PatientId;`	PatientAlreadyScheduledForMedicalSpecialty	There is a pending office visit scheduled...
PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}	DuplicatePrimaryKey	Record already exists

Resumamos todo lo que sabemos chequeará la transacción en ejecución cuando se intente insertar y luego veremos qué chequea el Business Component.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

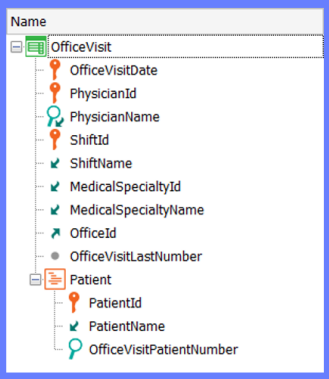
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



Por el orden de los atributos en la estructura de la transacción, el orden de los chequeos será el que mostramos.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

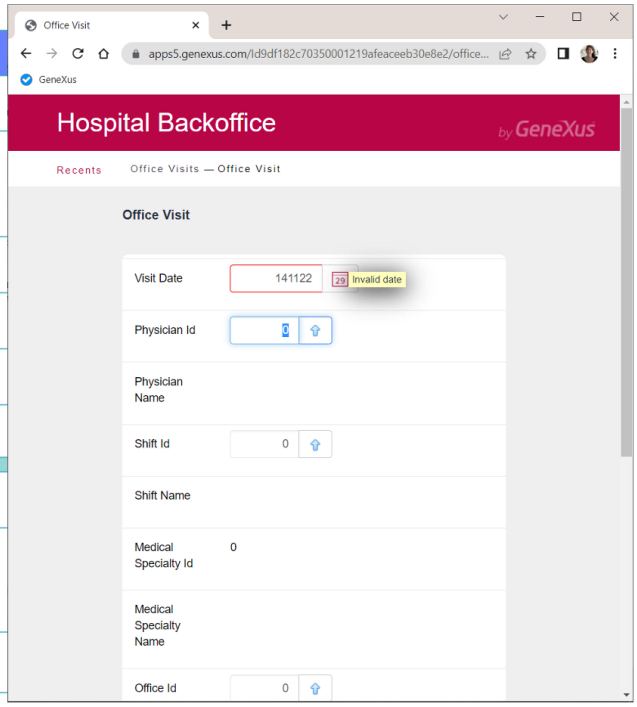
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



No agregamos los chequeos de los tipos de datos. Por ejemplo, para el primer campo claramente chequeará que la fecha sea válida.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

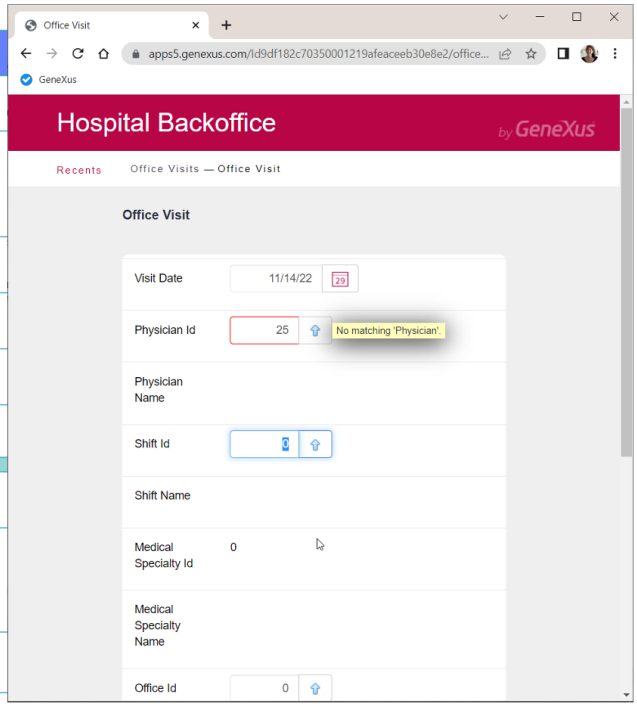
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



El siguiente chequeo será que exista un médico con ese Id en la tabla de médicos. De lo contrario veremos este error de integridad referencial. Coloquemos el médico 2.

Checks	Error Id	Error Description
FK: PhysicianId	ForeingKeyNotFound	No matching 'Physician"
`error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: " + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;`	PhysicianNotAvailable	Physician... already has 2 shifts...
FK: ShiftId	ForeingKeyNotFound	No matching 'Shift'
FK: OfficeId	ForeingKeyNotFound	No matching 'Office'
CK: {OfficeVisitDate, ShiftId, OfficeId}		Office Visit Date, Shift Id, Office Id already exists
PK: {OfficeVisitDate, PhysicianId, ShitfId}	DuplicatePrimaryKey	Record already exists
FK: PatientId	ForeingKeyNotFound	No matching 'Patient'
`error("There is a pending office visit scheduled for this patient for the same specialty", PatientAlreadyScheduledForMedicalSpecialty) if not &IsOk and Insert Level PatientId;`	PatientAlreadyScheduledForMedicalSpecialty	There is a pending office visit scheduled...
PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}	DuplicatePrimaryKey	Record already exists

Lo siguiente que se hará será invocar al procedimiento que cuenta la cantidad de consultas que ese médico ya tiene en ese día y si le da 2 o más, se disparará la regla de error, que muestra este mensaje en pantalla.

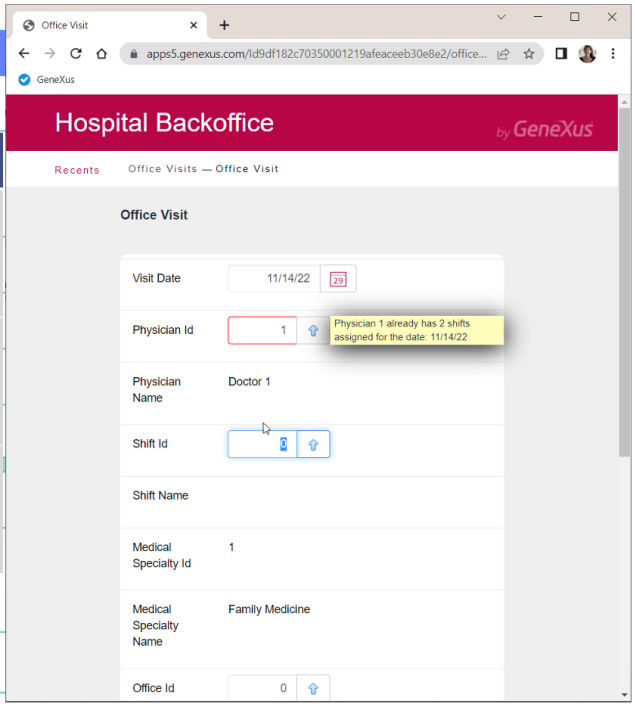
Checks

FK: PhysicianId

Monday	Tuesday	Wednesday	Thursday	Friday
Physician: 1 Office: 2		Physician: 2 Office: 1	Physician: 1 Office: 2 Patient: 2	
Physician: 1 Office: 2				
	Physician: 1 Office: 2			

```
if not &IsOk and Insert
Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



Con el médico 2 no nos dio error, pero si colocamos en cambio el médico 1, que sabemos que ya tiene 2 turnos ese día... sí vemos el error.

Checks	Error Id	Error Description
FK: PhysicianId	ForeingKeyNotFound	No matching 'Physician"
`error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: " + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;`	PhysicianNotAvailable	Physician... already has 2 shifts...
FK: ShiftId	ForeingKeyNotFound	No matching 'Shift'
FK: OfficeId	ForeingKeyNotFound	No matching 'Office'
CK: {OfficeVisitDate, ShiftId, OfficeId}		Office Visit Date, Shift Id, Office Id already exists
PK: {OfficeVisitDate, PhysicianId, ShitfId}	DuplicatePrimaryKey	Record already exists
FK: PatientId	ForeingKeyNotFound	No matching 'Patient'
`error("There is a pending office visit scheduled for this patient for the same specialty", PatientAlreadyScheduledForMedicalSpecialty) if not &IsOk and Insert Level PatientId;`	PatientAlreadyScheduledForMedicalSpecialty	There is a pending office visit scheduled...
PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}	DuplicatePrimaryKey	Record already exists

Este código interno del error corresponde al segundo parámetro de la regla error. Si no lo colocamos, el Id del error quedará vacío (lo que no afecta en nada el funcionamiento).

Volvamos a colocar al médico 2 para seguir.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

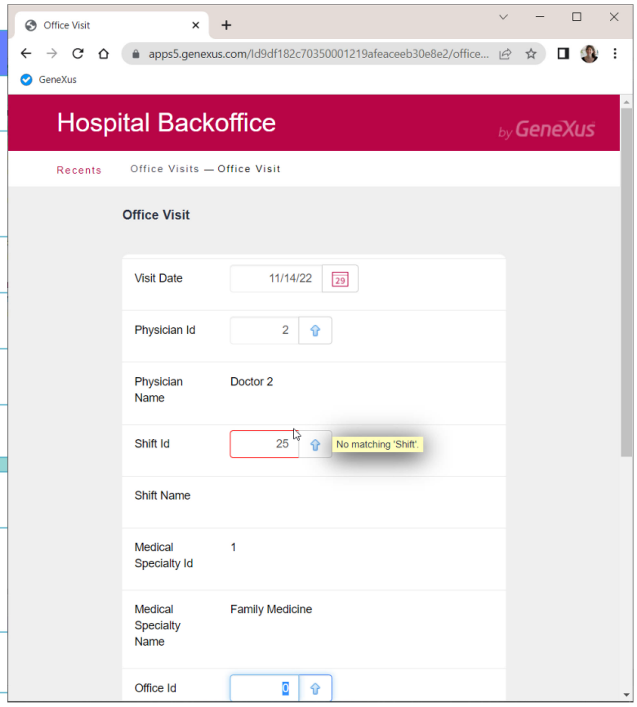
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



Lo siguiente será controlar la integridad referencial con ShiftId.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

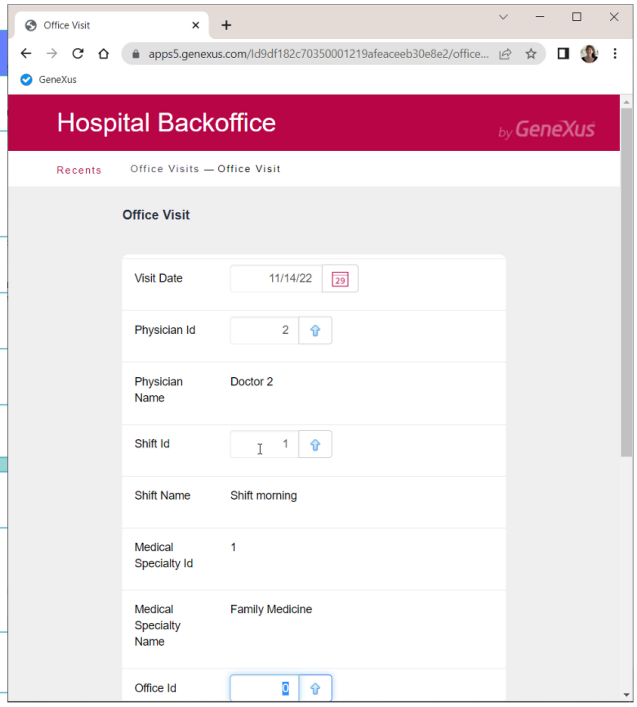
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



Asignemos ahora el Shift 1, Morning.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

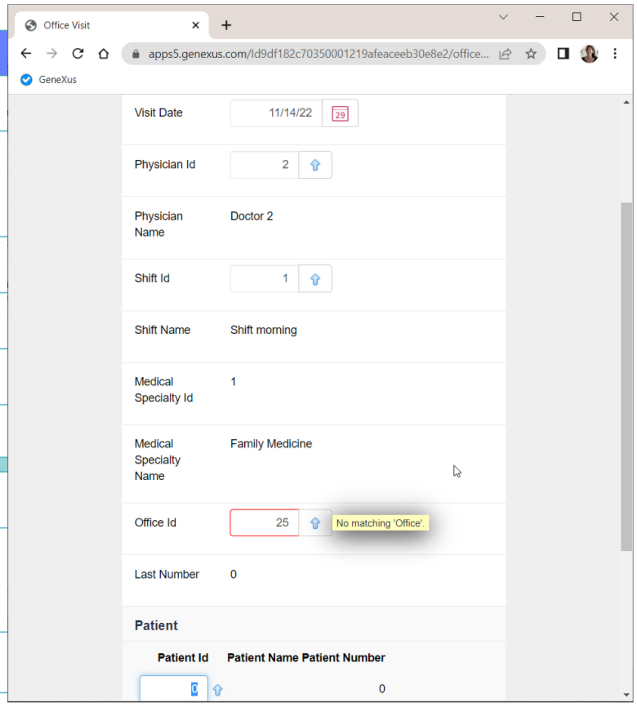
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



Luego controlará que el consultorio exista. Por ejemplo, colocamos el 25.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

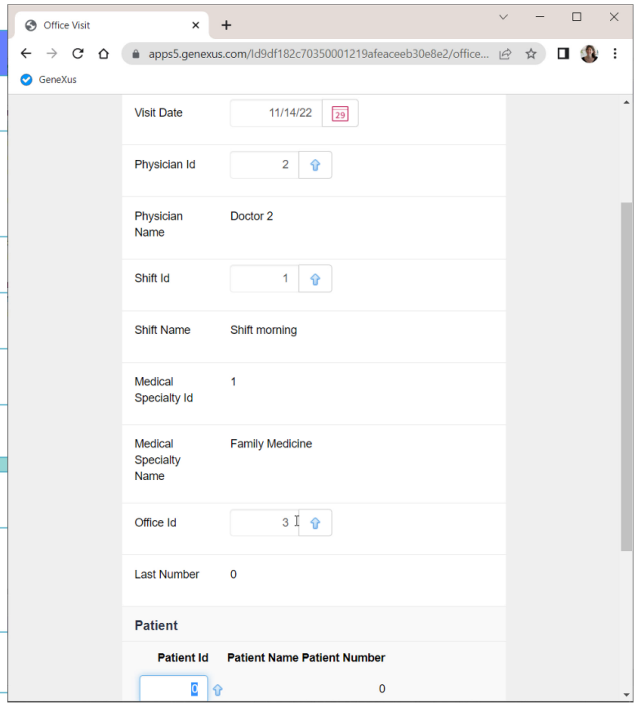
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



Y ahora este, que sí existe.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

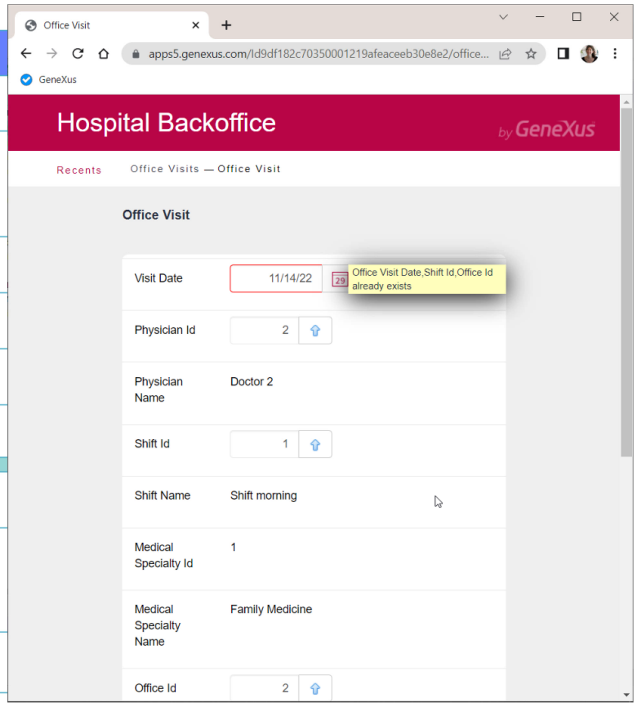
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



Al salir del campo OfficeId ya tiene para chequear la clave candidata. En este caso habíamos colocado consultorio 3, pero observemos qué pasaría si colocáramos el 2, que sabemos que ya está ocupado (por el médico 1). Nos muestra este mensaje de error. Para chequearlo es que utiliza el índice unique.

Volvamos a colocar el consultorio 3, que sabemos no está ocupado.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

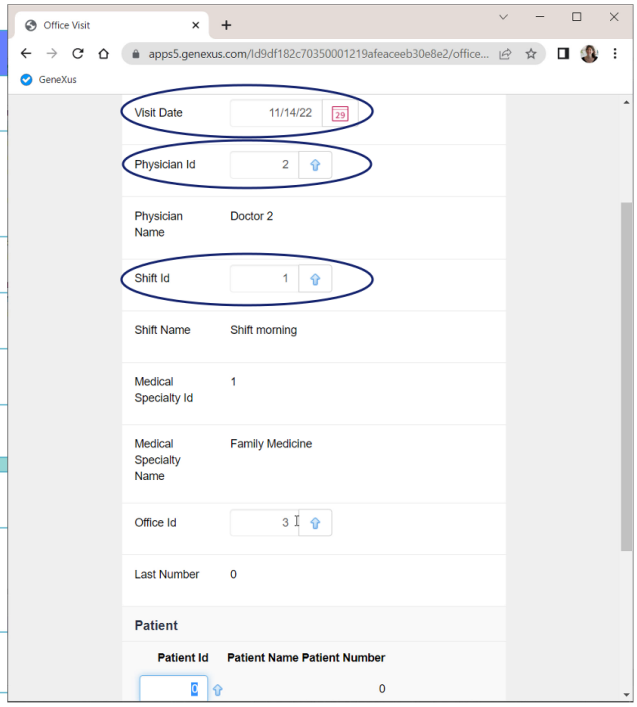
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

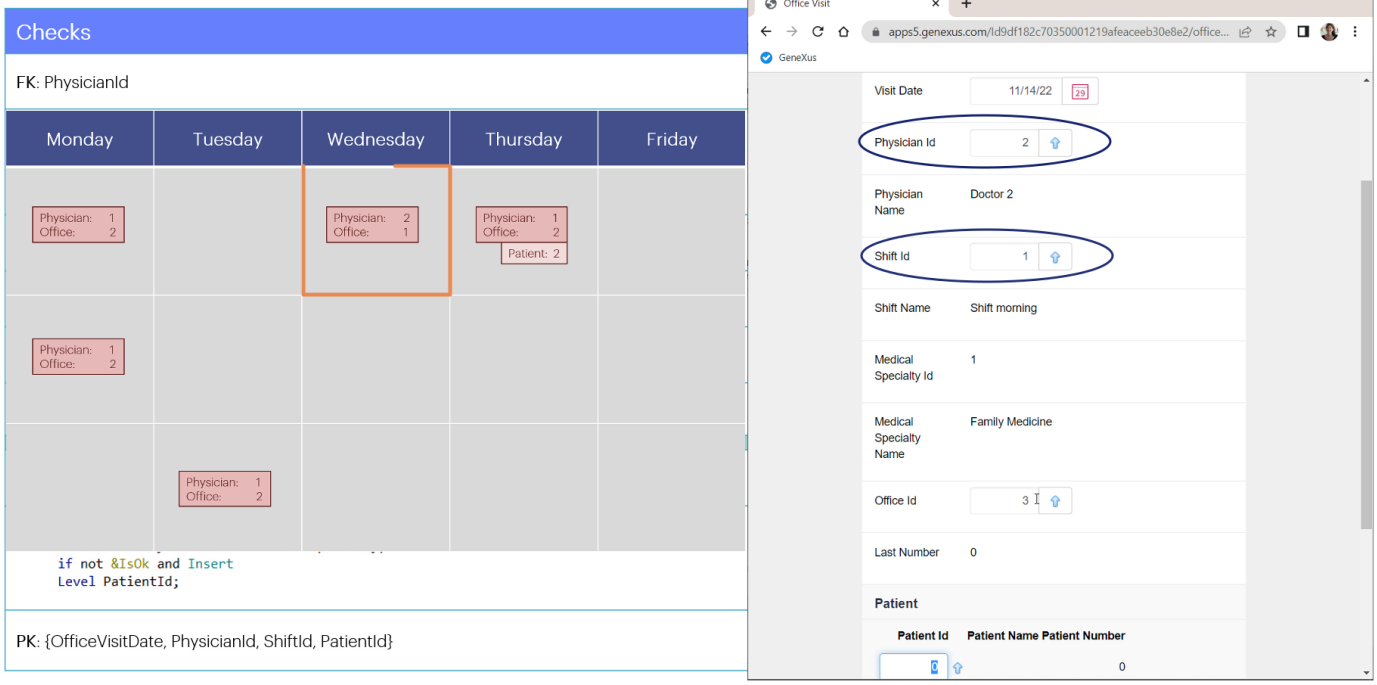
FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



En este momento se controla la unicidad del registro que se está queriendo insertar. En este caso no encontró que el registro ya existiera, por lo que no vemos nada.



Si hubiéramos intentado ingresar un registro que ya existía...por ejemplo observemos que para este médico y este turno, para el día miércoles ya teníamos una consulta agendada, entonces veamos qué pasaría si cambiamos la fecha para la nueva consulta que estamos queriendo asignar, y colocamos esta del miércoles.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

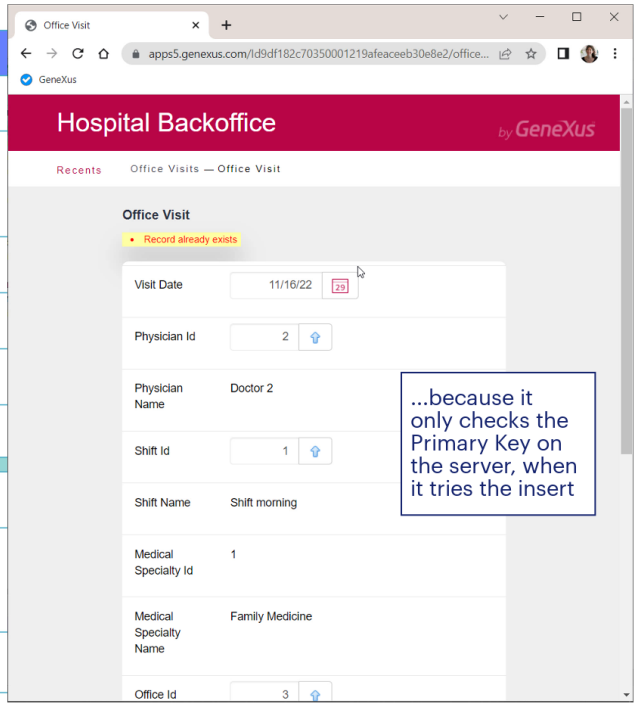
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



No nos lo está mostrando interactivamente, pero si presionamos confirm veremos el mensaje "**Record already exists**".

Volvamos a colocar la fecha del lunes.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

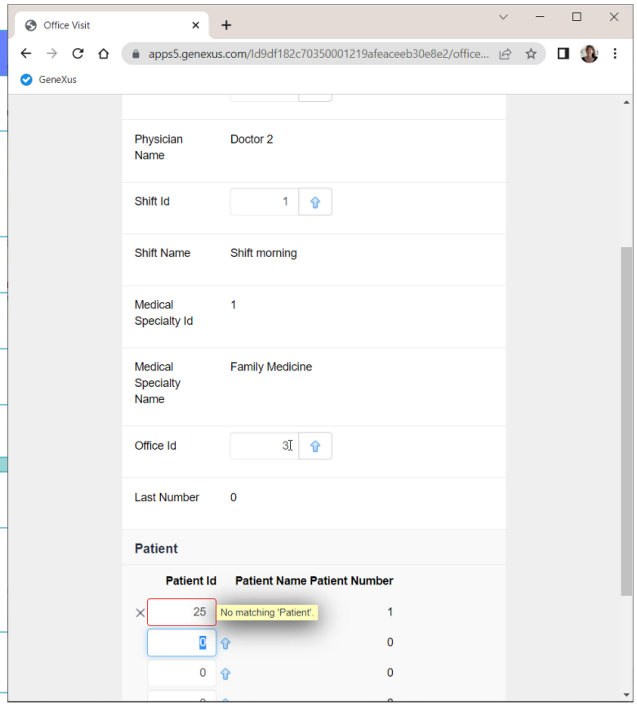
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



Con esto los chequeos del primer nivel terminaron.

Ahora se pasa a chequear cada línea del segundo nivel. Si colocamos un id de paciente inexistente, fallará la integridad referencial. Coloquemos uno existente, el 1.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

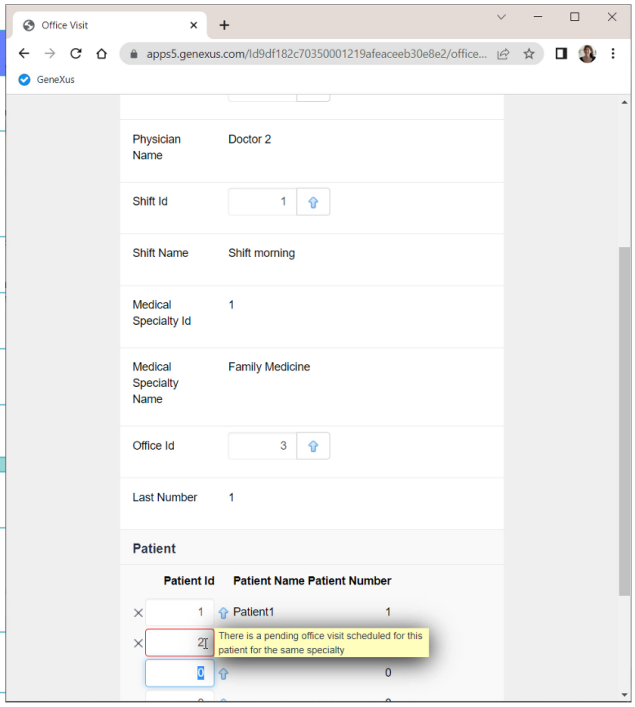
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}



Lo siguiente que se va a chequear es el error. Probemos con el paciente 2. Se está disparando la regla de error que habíamos declarado. ¿Por qué?

Porque si recordamos, ya teníamos al paciente 2 agendado para la consulta del jueves, para el médico 1 que es de la misma especialidad que el médico 2 (Family Medicine). Entonces no podemos agendarlo.

Checks

FK: PhysicianId

```
error("Physician " + PhysicianId.ToString() +" already has 2 shifts assigned for the date: "
     + OfficeVisitDate.ToString(), PhysicianNotAvailable) if &shifts >= 2 and Insert;
```

FK: ShiftId

FK: OfficeId

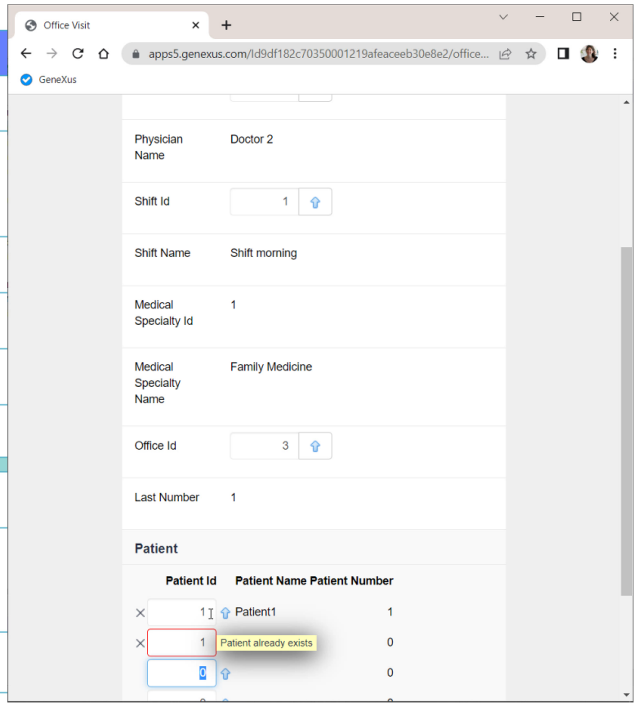
CK: {OfficeVisitDate, ShiftId, OfficeId}

PK: {OfficeVisitDate, PhysicianId, ShitfId}

FK: PatientId

```
error("There is a pending office visit scheduled for this patient for the same specialty",
     PatientAlreadyScheduledForMedicalSpecialty)
     if not &IsOk and Insert
     Level PatientId;
```

PK: {OfficeVisitDate, PhysicianId, ShiftId, PatientId}

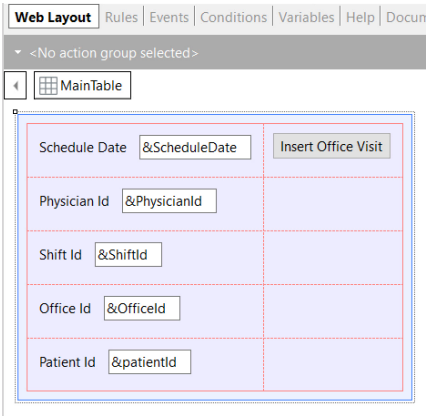


Luego, se chequeará la unicidad de la clave primaria de este segundo nivel. Como no tenemos ningún otro paciente ingresado, no fallará. Pero probemos con otra línea. Volvamos a colocar el 1. Aquí vemos que se está controlando la unicidad, efectivamente.

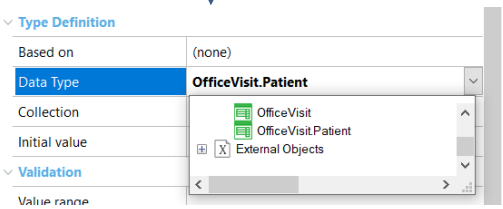
Si ahora presionamos Confirm, vemos ingresada la consulta.

	Monday	Tuesday	Wednesday	Thursday	Friday
Shift 1 morning	Physician: 1 Office: 2 Physician: 2 Office: 3 Patient: 1		Physician: 2 Office: 1	Physician: 1 Office: 2 Patient: 2	
Shift 2 intermediate	Physician: 1 Office: 2				
Shift 3 evening		Physician: 1 Office: 2			

Ahora vamos a intentar repetir lo mismo, pero insertando a través del Business Component. Borremos entonces la consulta recién ingresada para volver a intentar.




```
Event 'Insert Office Visit'
    &officeVisit = new()
    &officeVisit.OfficeVisitDate = &ScheduleDate
    &officeVisit.PhysicianId = &PhysicianId
    &officeVisit.ShiftId = &ShiftId
    &officeVisit.OfficeId = &OfficeId
    If not &patientId.IsEmpty()
        &officeVisitPatient = new()
        &officeVisitPatient.PatientId = &patientId
        &officeVisit.Patient.Add(&officeVisitPatient)
    endif
    if &officeVisit.Insert()
        Commit
    endif
    &messages = &officeVisit.GetMessages()
Endevent
```



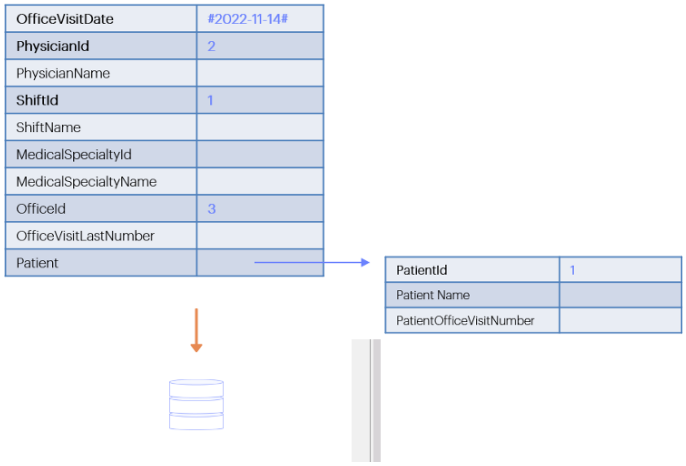
Para hacerlo bien fácil hemos creado un web panel con 5 variables de entrada: para los 4 atributos relevantes del cabezal de la consulta, y para ingresar un paciente. Primero intentaremos insertar a través del Business Component. Para ello, por supuesto, hemos prendido la propiedad Business Component de la transacción.

A partir de allí es que pudimos crear esta variable del tipo de datos el Business Component, a través de la cuál insertaremos.

Como buena práctica, pedimos antes que nada espacio de memoria nuevo para la variable, para asegurarnos de que esté limpia. Asignamos valores a los 4 elementos del cabezal, a partir de las variables de la pantalla.

Luego, si el campo en la pantalla para Patient Id no se dejó vacío, entonces intentaremos agregar un paciente. Tenemos esta variable del tipo de datos de cada línea, la limpiamos, con new, le asignamos valor solo al elemento PatientId y lo agregamos a la colección de patients del Business Component.

```
Event 'Insert Office Visit'
    &officeVisit = new()
    &officeVisit.OfficeVisitDate = &ScheduleDate
    &officeVisit.PhysicianId = &PhysicianId
    &officeVisit.ShiftId = &ShiftId
    &officeVisit.OfficeId = &OfficeId
    If not &patientId.IsEmpty()
        &officeVisitPatient = new()
        &officeVisitPatient.PatientId = &patientId
        &officeVisit.Patient.Add(&officeVisitPatient)
    endif
    if &officeVisit.Insert()
        Commit
    endif
    &messages = &officeVisit.GetMessages()
Endevent
```



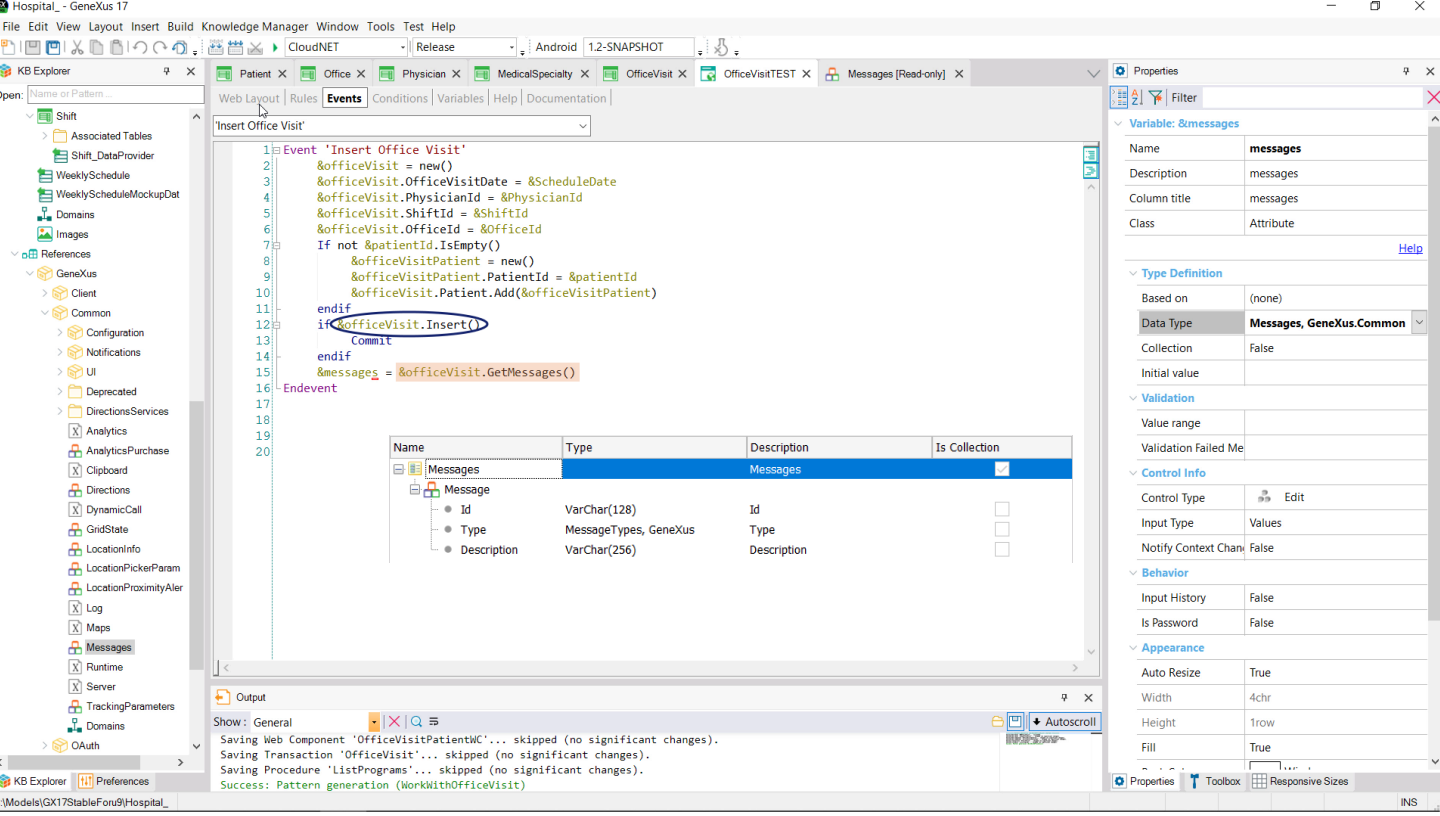
Con eso hemos cargado la estructura de la variable &officeVisit. Ahora tenemos que dar la orden de intentar la inserción...

```
Event 'Insert Office Visit'
    &officeVisit = new()
    &officeVisit.OfficeVisitDate = &ScheduleDate
    &officeVisit.PhysicianId = &PhysicianId
    &officeVisit.ShiftId = &ShiftId
    &officeVisit.OfficeId = &OfficeId
    If not &patientId.IsEmpty()
        &officeVisitPatient = new()
        &officeVisitPatient.PatientId = &patientId
        &officeVisit.Patient.Add(&officeVisitPatient)
    endif
    if &officeVisit.Insert()
        Commit
    endif
    &messages = &officeVisit.GetMessages()
Endevent
```

OfficeVisitDate	#2022-11-14#
PhysicianId	2
PhysicianName	Doctor 2
ShiftId	1
ShiftName	Shift morning
MedicalSpecialtyId	1
MedicalSpecialtyName	Family Medicine
OfficeId	3
OfficeVisitLastNumber	1
Patient	

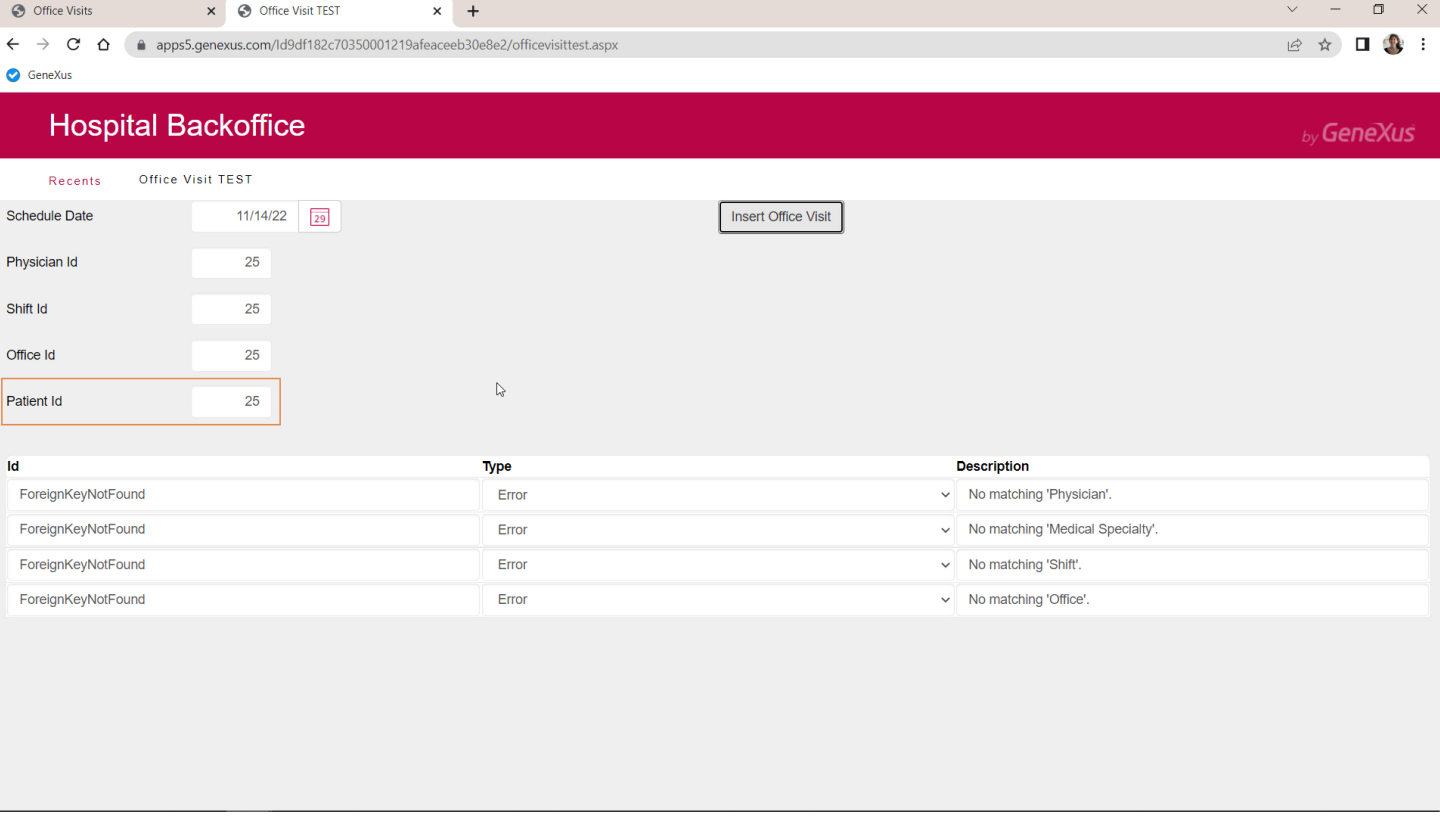
PatientId	1
Patient Name	Patient 1
PatientOfficeVisitNumber	1

Y si es exitosa, commitear.

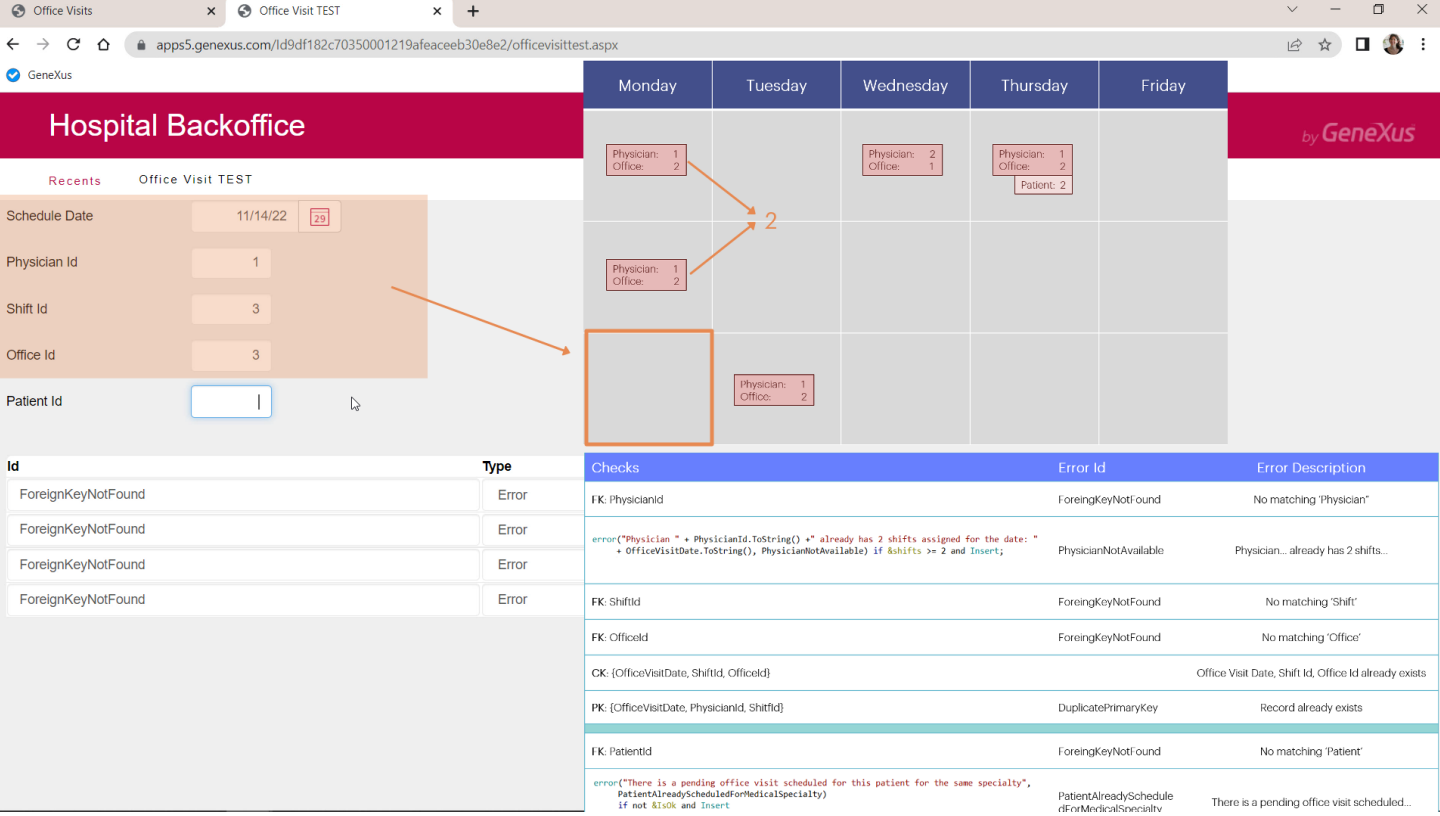


En la variable &messages del tipo de datos el SDT que viene predefinido en el módulo GeneXus, obtenemos todos los mensajes que hayan resultado de la ejecución del método Insert. Y la insertamos en pantalla para poder ver todos esos mensajes.

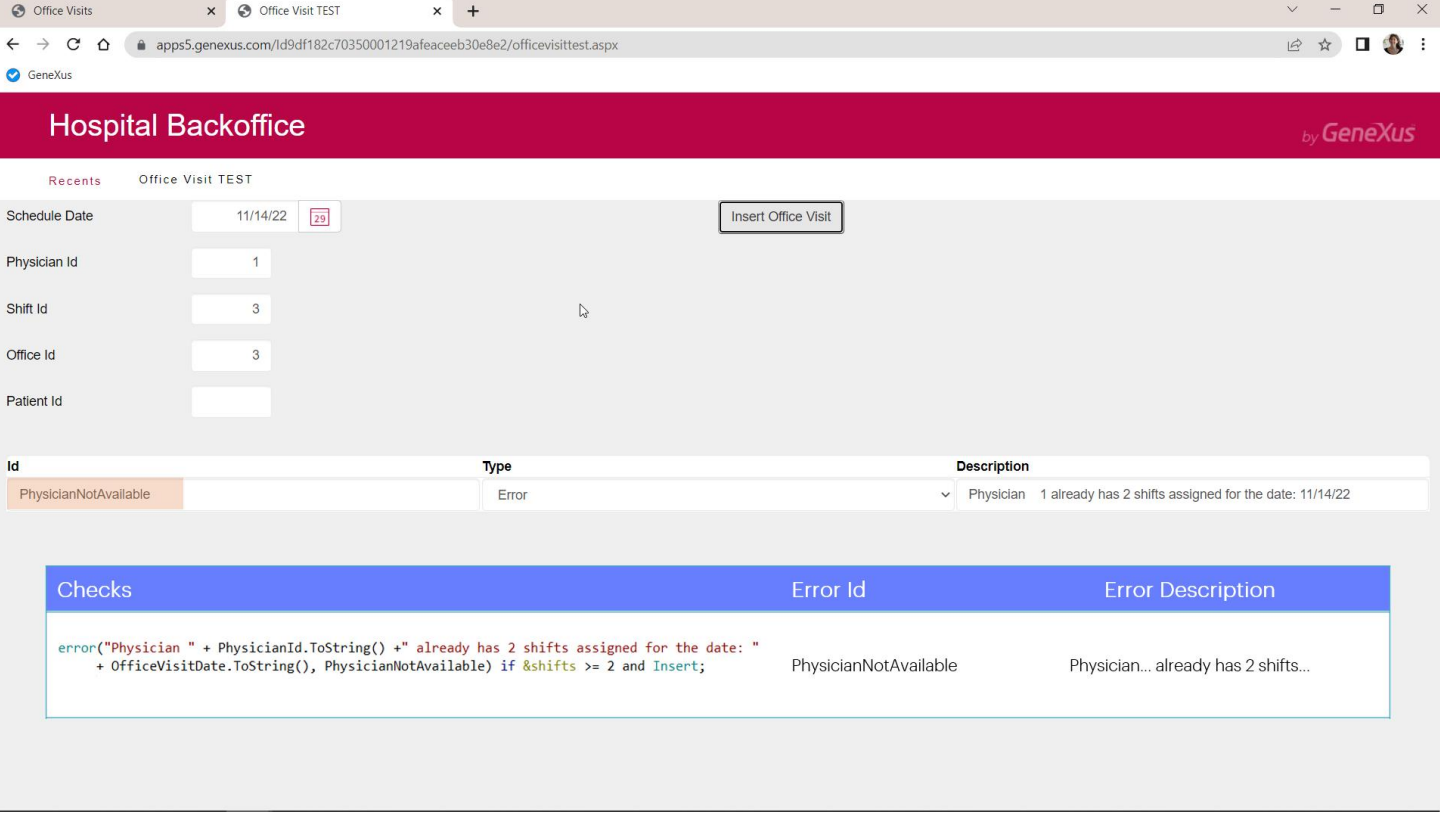
Entonces ejecutemos para repetir todo lo que hicimos a través de la transacción, pero ahora por esta vía.



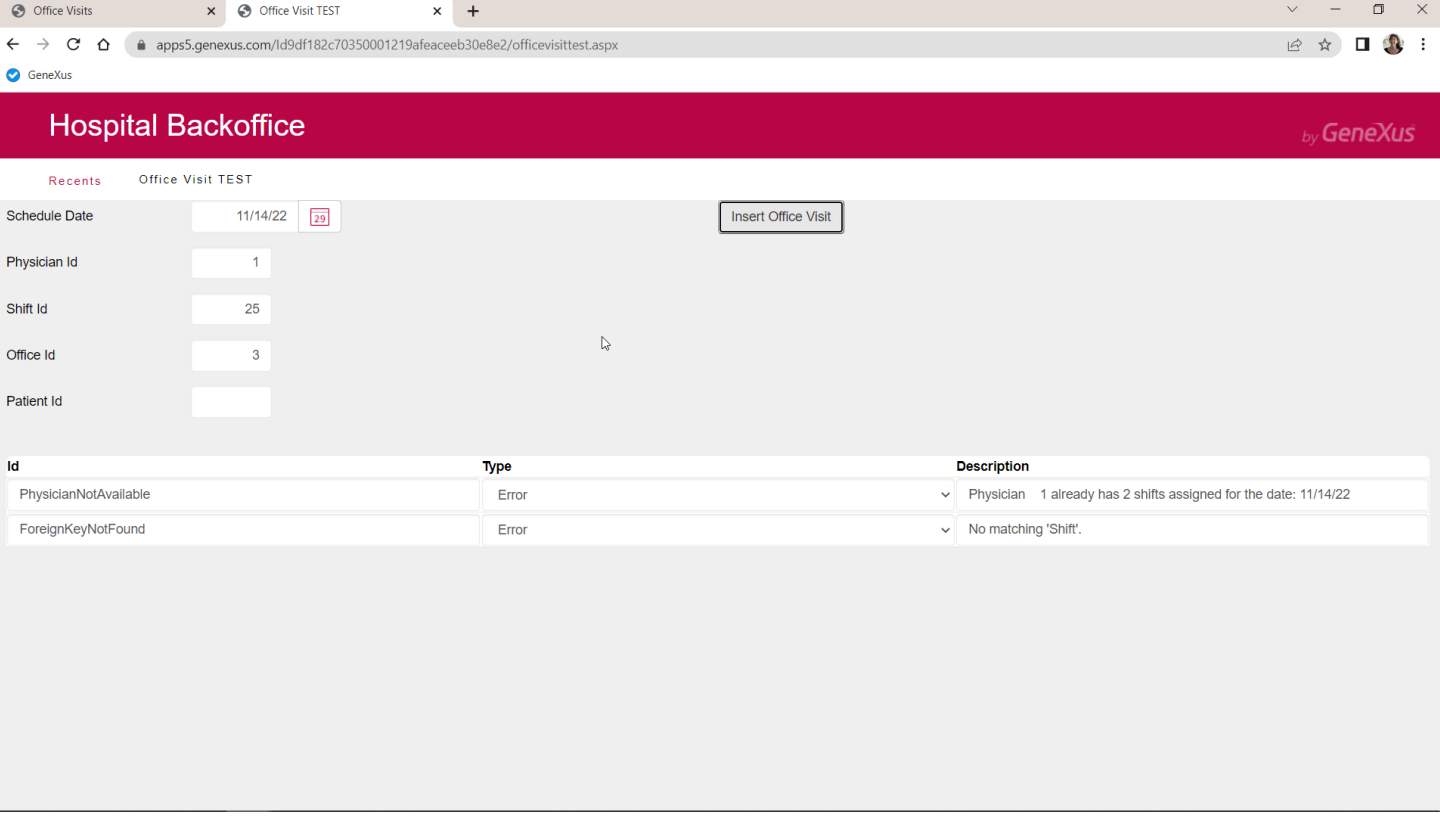
Elijamos como fecha la del lunes que queríamos. Coloquemos un médico inexistente, un turno inexistente, un consultorio inexistente y hasta un paciente inexistente, para ver todos los mensajes que obtendremos al intentar insertar.
Vemos todos los errores de integridad referencial. A Patient ni llegó porque eso solo ocurriría si pudo insertar el cabezal.



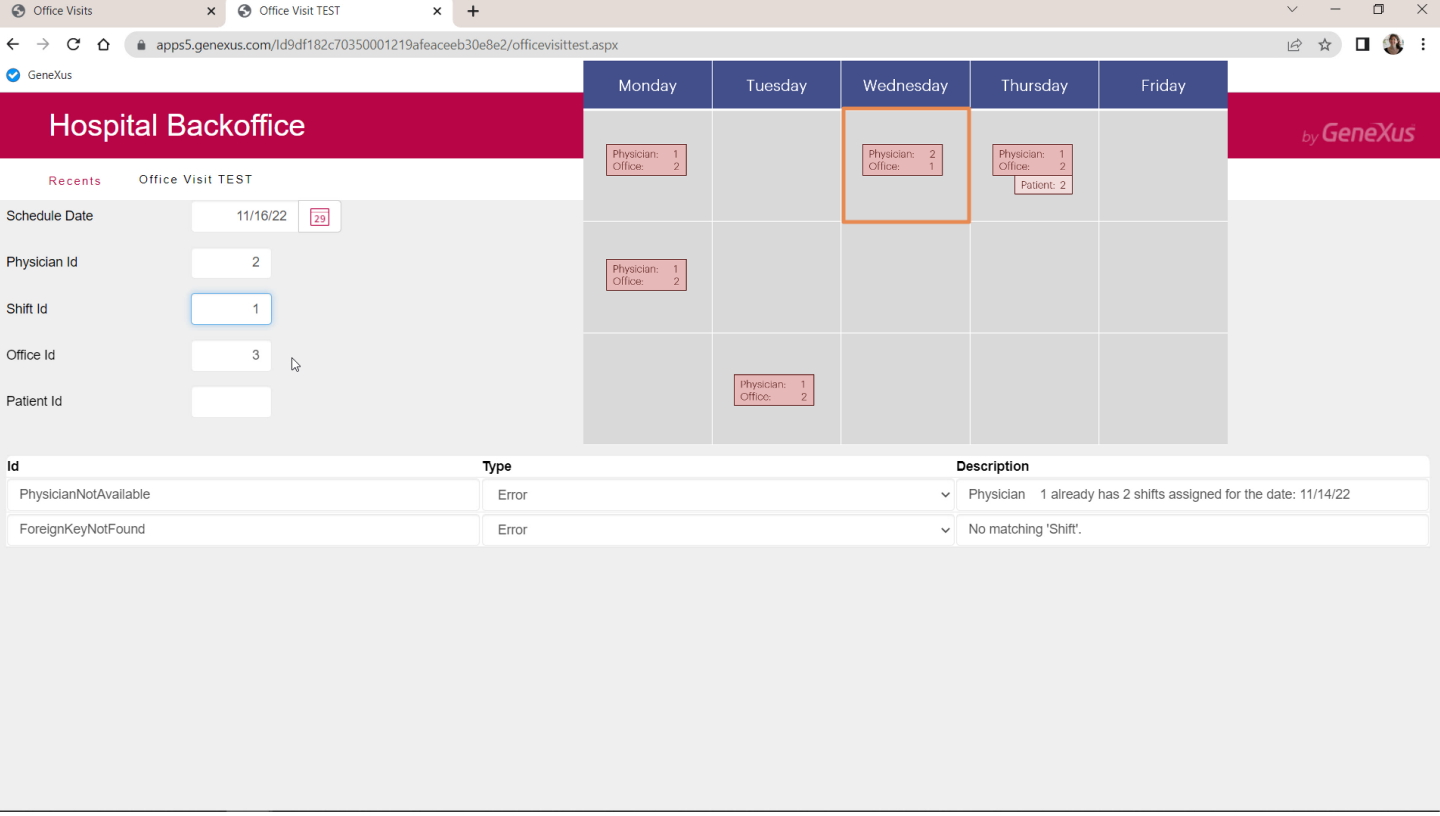
Ahora coloquemos al médico 1, en el turno 3, en el consultorio 3, sin paciente. Sabemos que nos tiene que fallar por la regla de error.



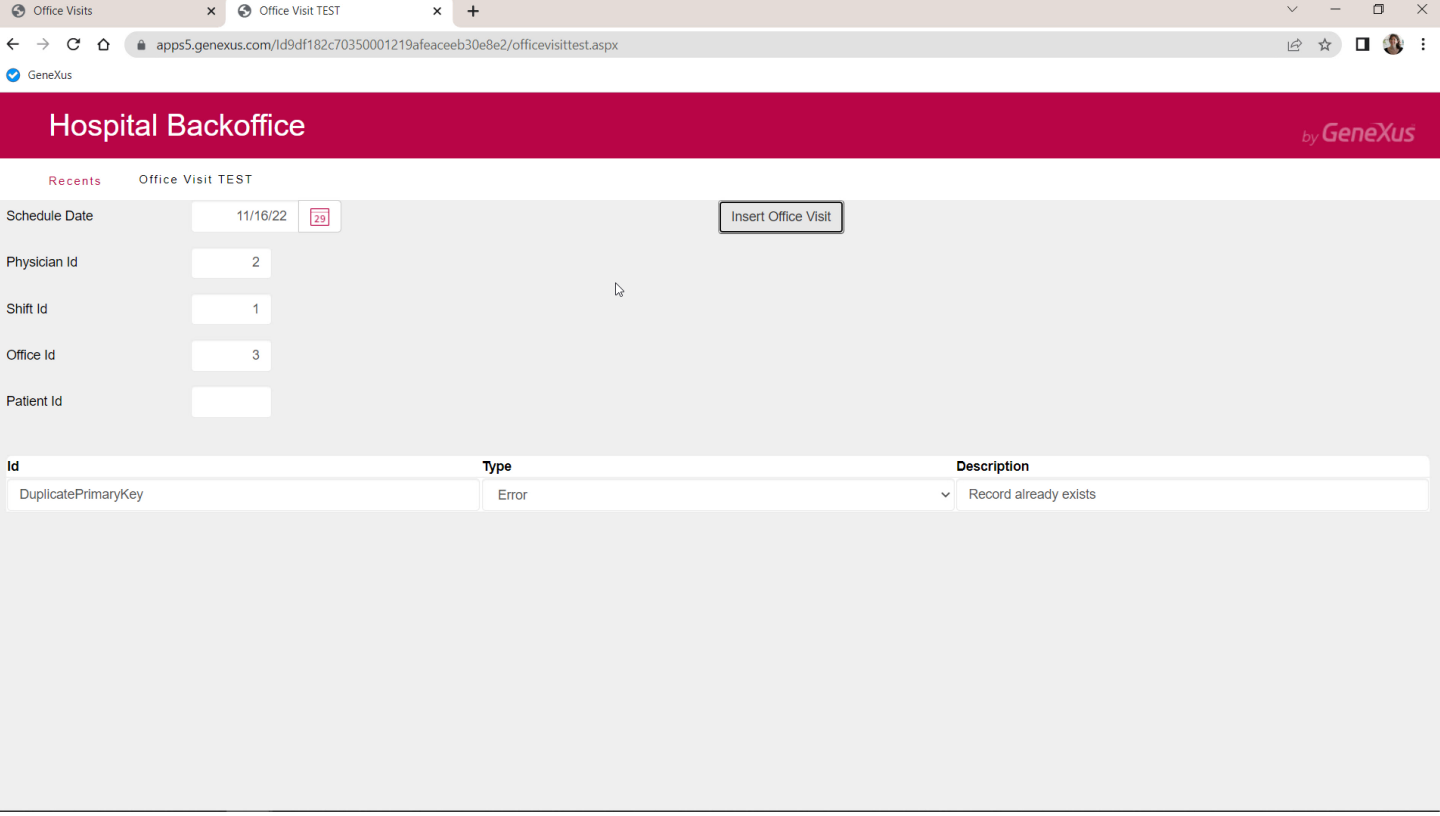
Vemos que el id del mensaje de error es el que colocamos en la regla.



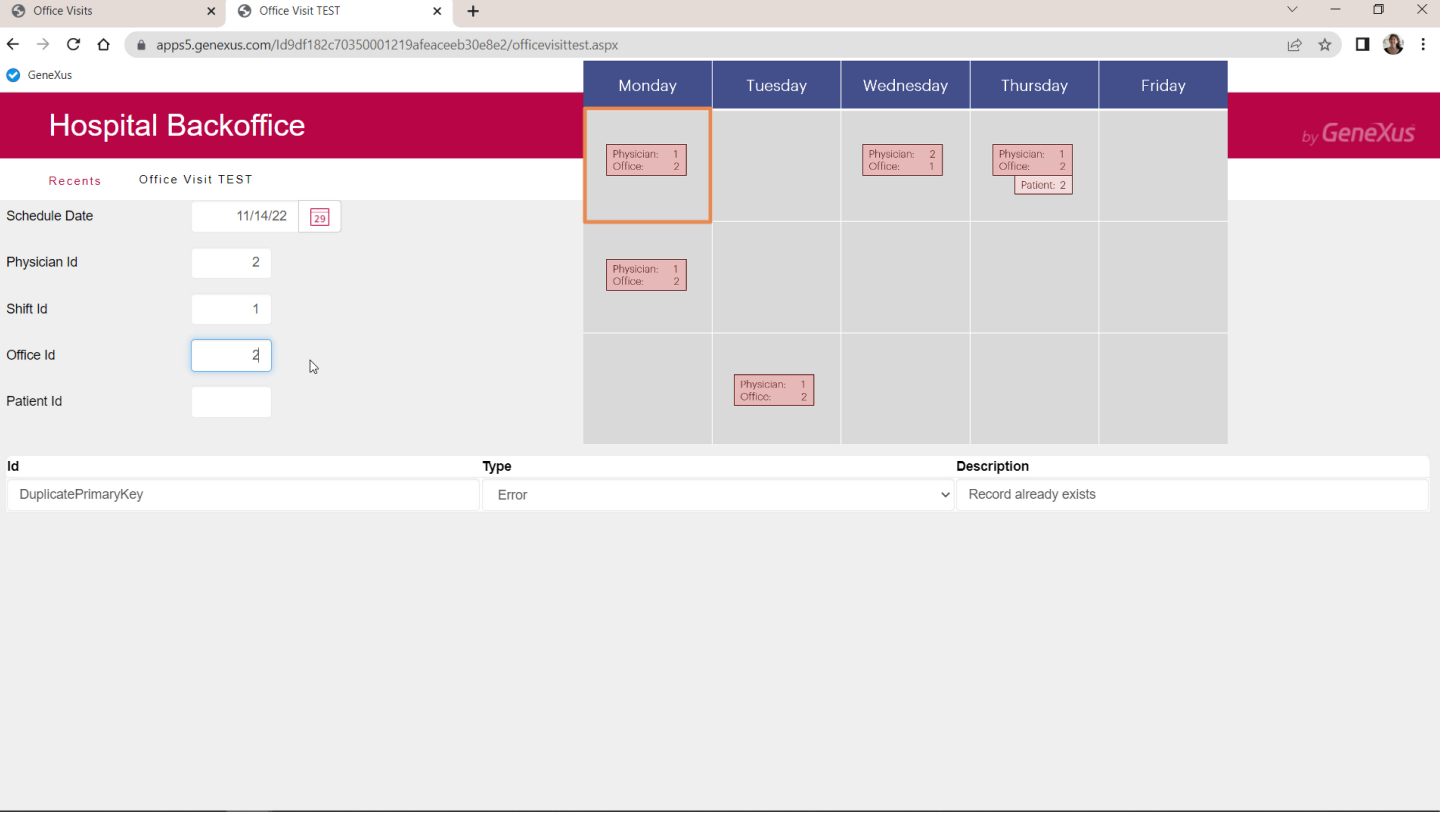
Podemos violar más chequeos, y lo veremos en la colección de mensajes. Por ejemplo, coloquemos el shift 25, que no existe, y vemos los dos mensajes de error.



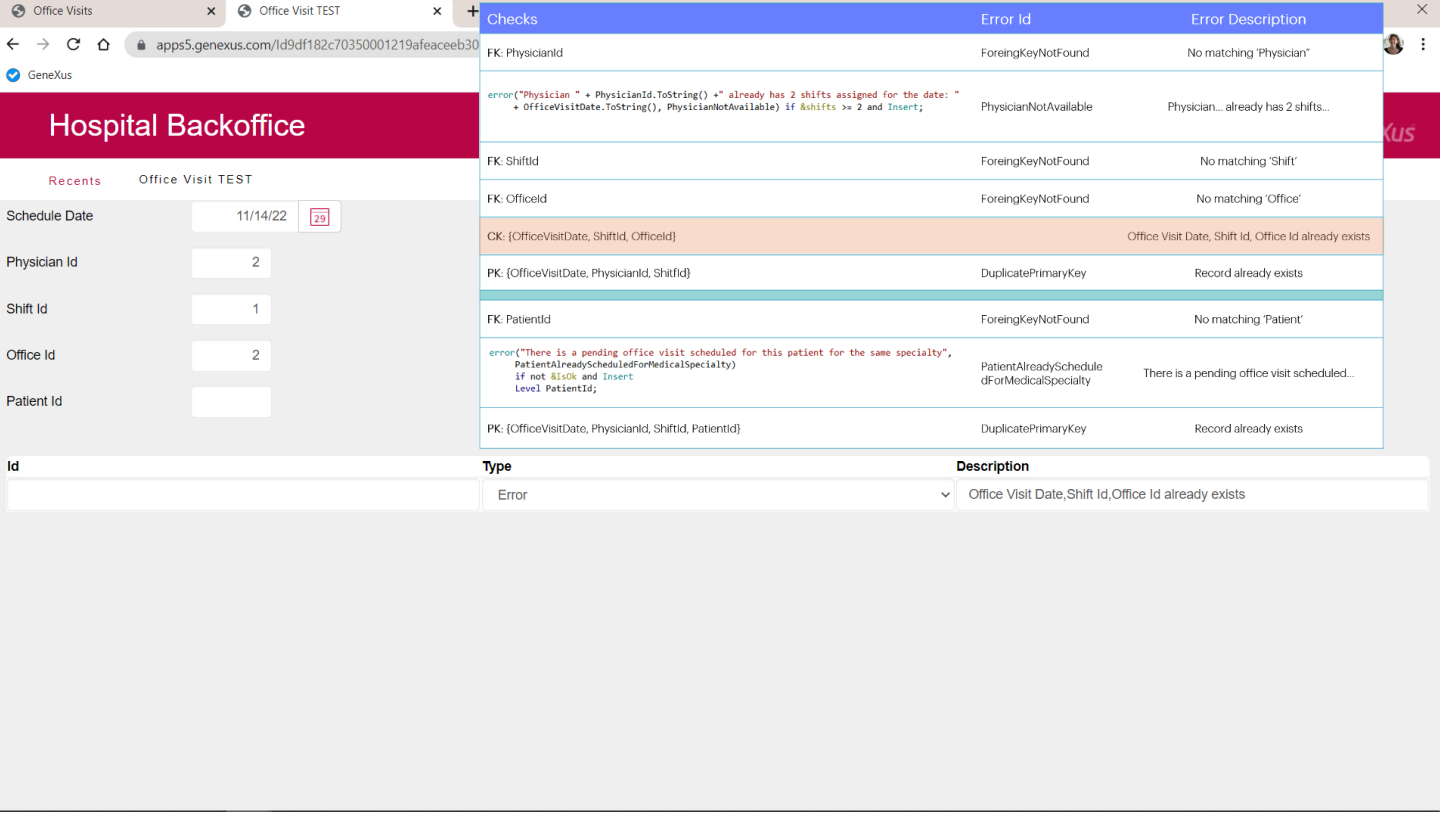
Ahora probemos el chequeo de unicidad de clave primaria. Intentemos insertar un registro para el miércoles, médico 2, turno 1, consultorio 3.



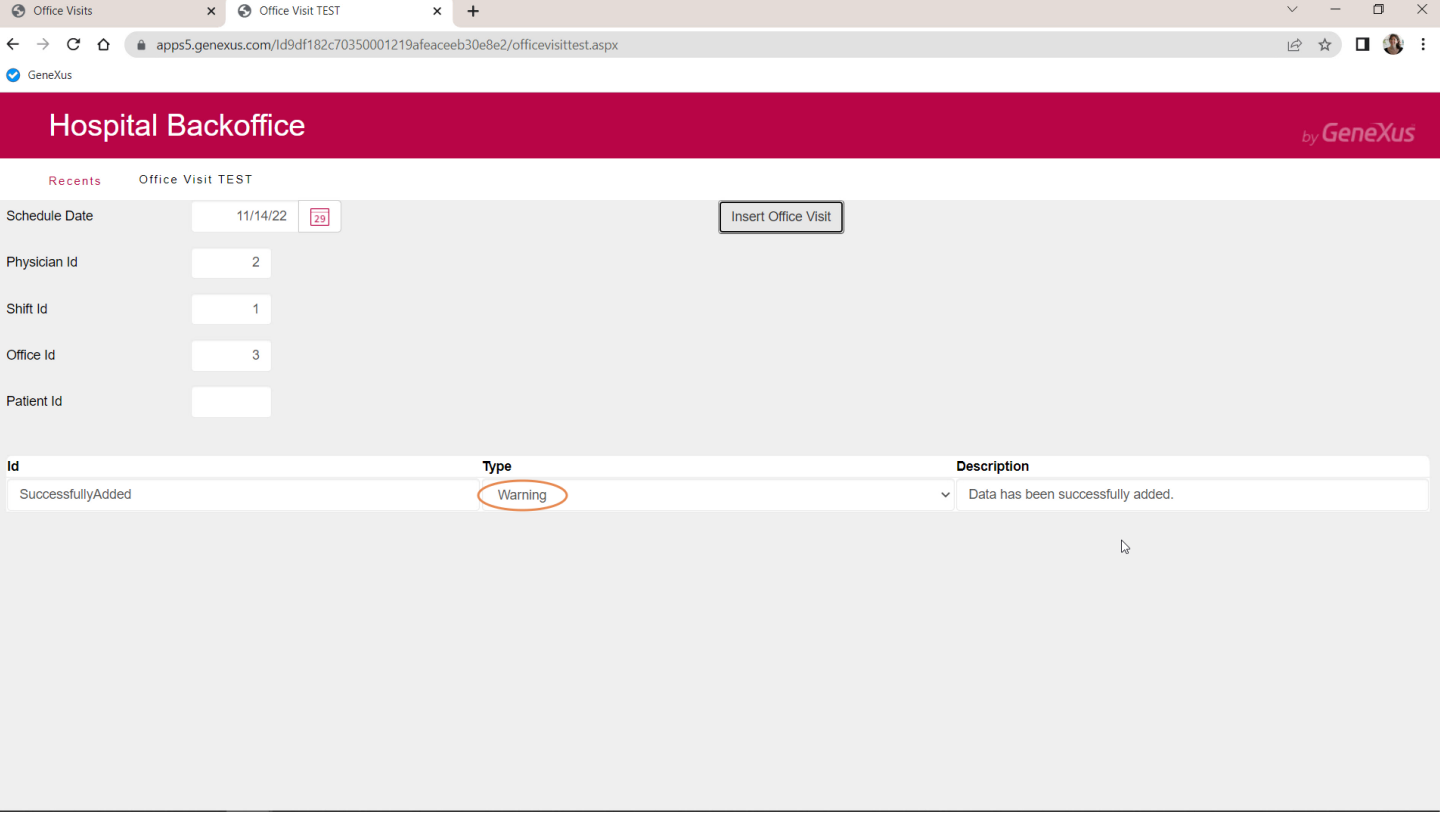
Bien, era lo que esperábamos.



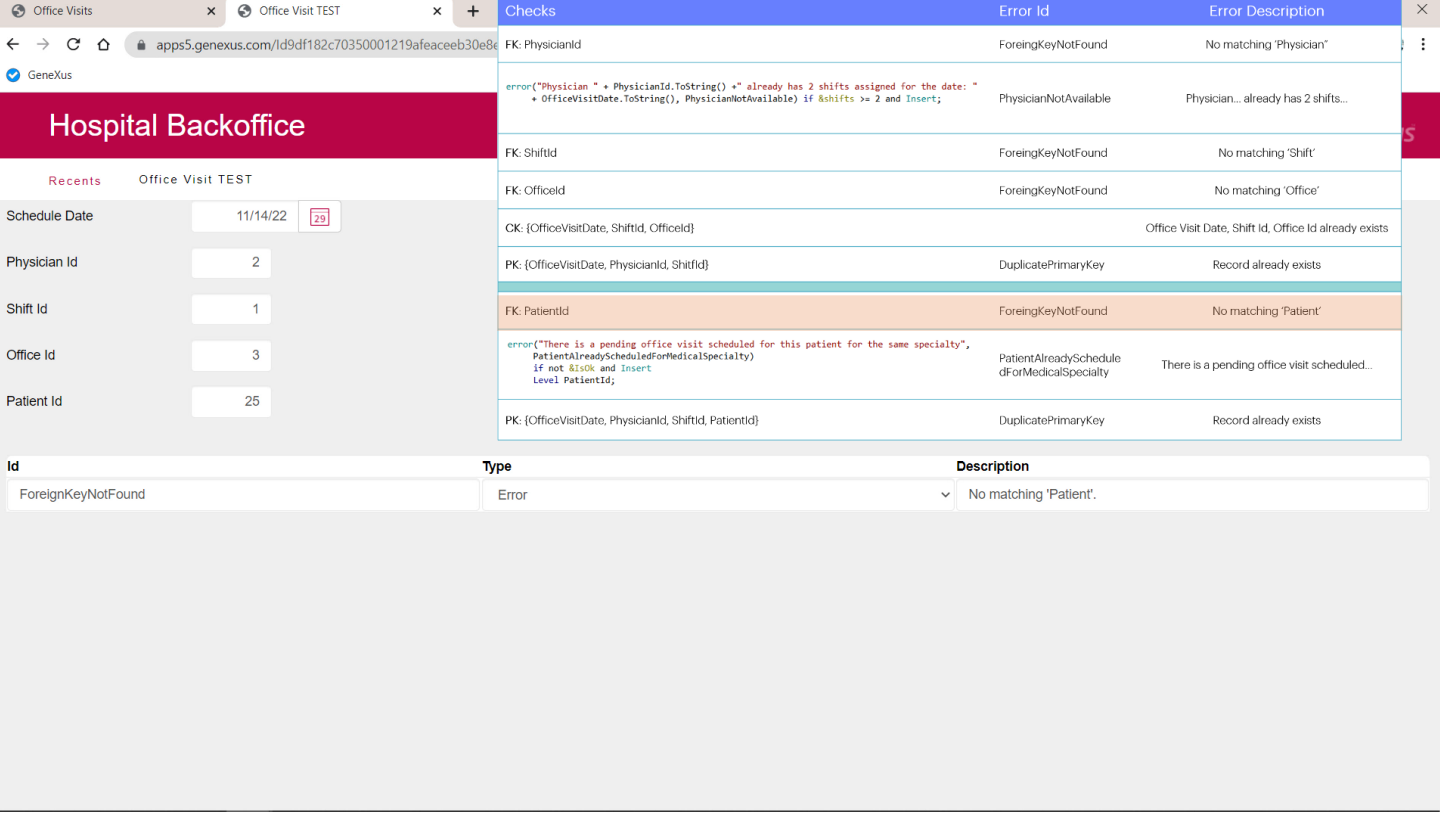
Ahora probemos insertar la consulta para el lunes en el turno 1, pero para consultorio ya ocupado, o sea, el 2.



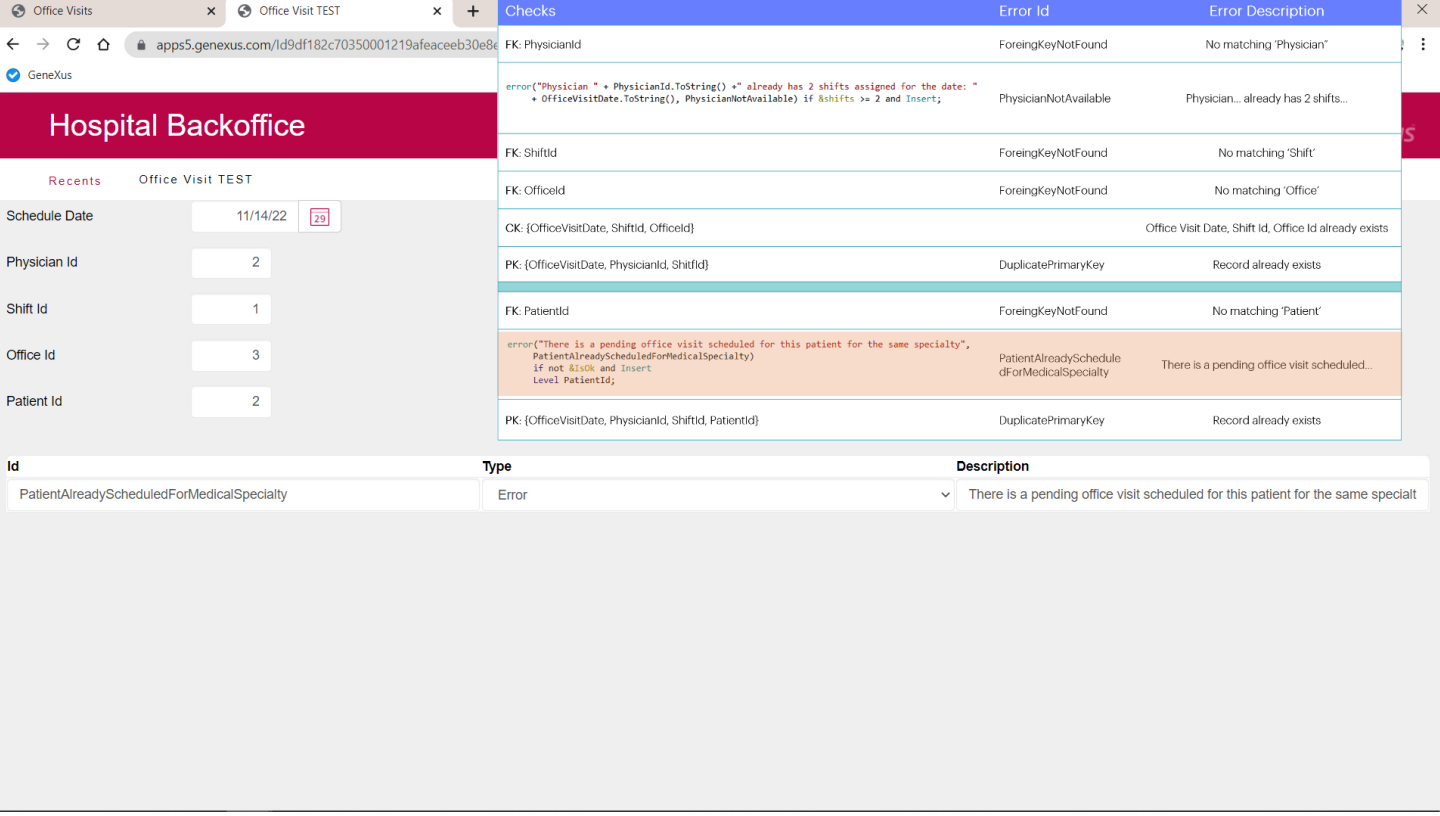
Vemos que chequeó unicidad de clave candidata.



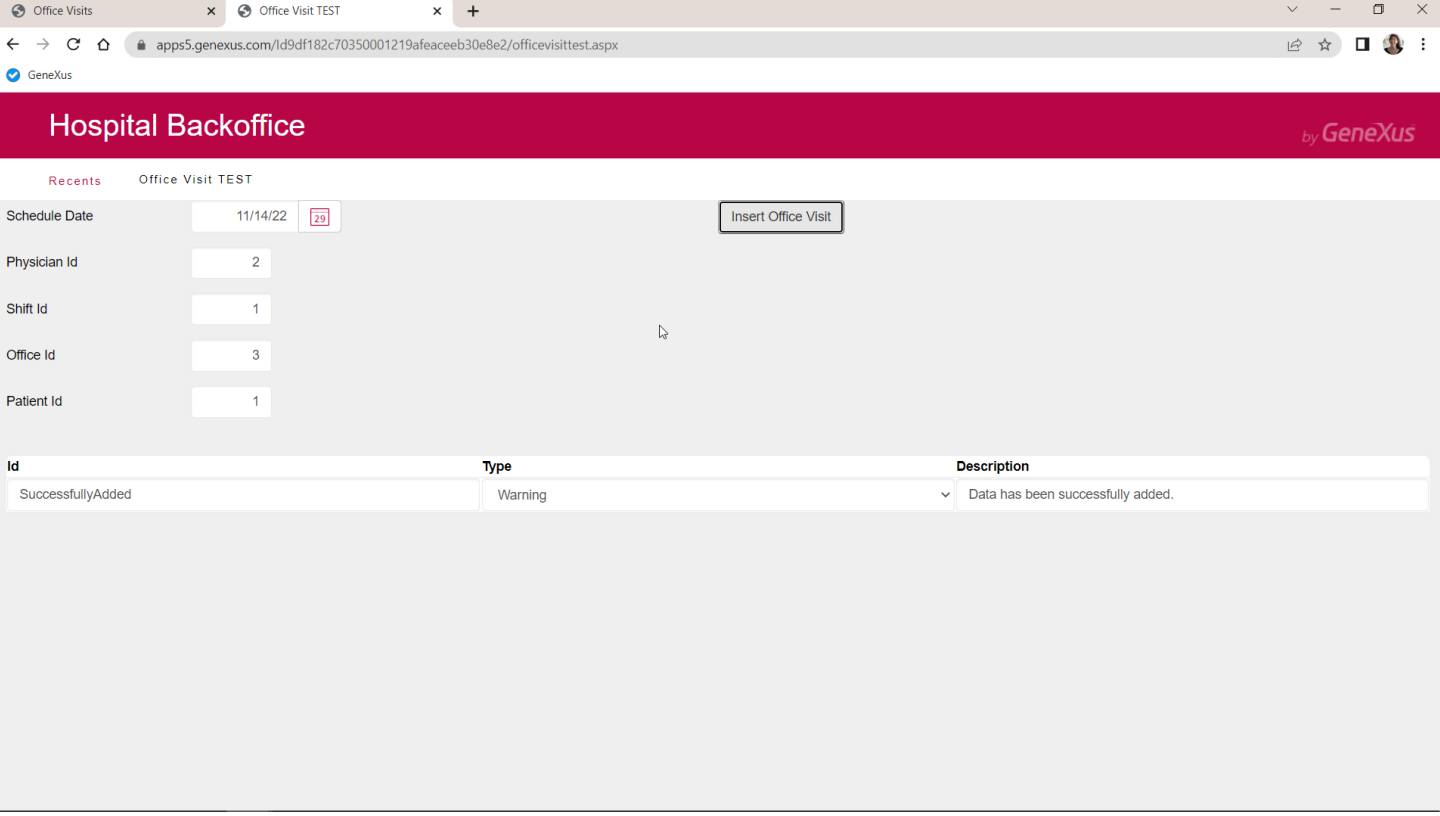
Ahora sí, coloquemos el consultorio correcto. Si dejamos vacío Patient Id, insertará el cabezal exitosamente. Vemos que el mensaje es de tipo Warning y es el que nos aparece en la transacción cuando no se le aplica el pattern, es decir, cuando no retorna al objeto llamador.



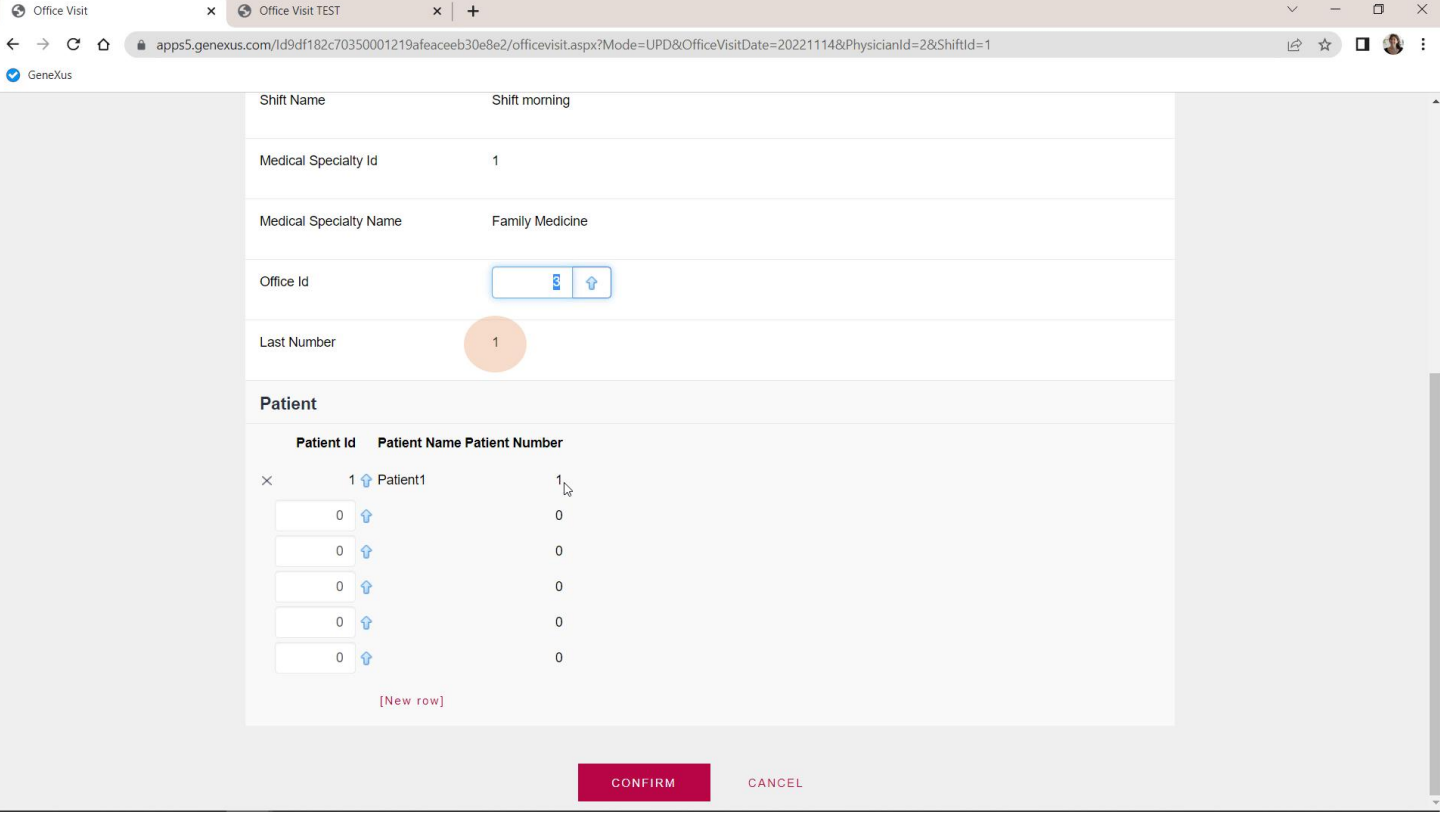
Eliminemos el registro e intentemos de vuelta, esta vez con paciente.
Colocamos paciente inexistente y vemos el error de integridad referencial.
Por lo que el cabezal no queda ingresado.



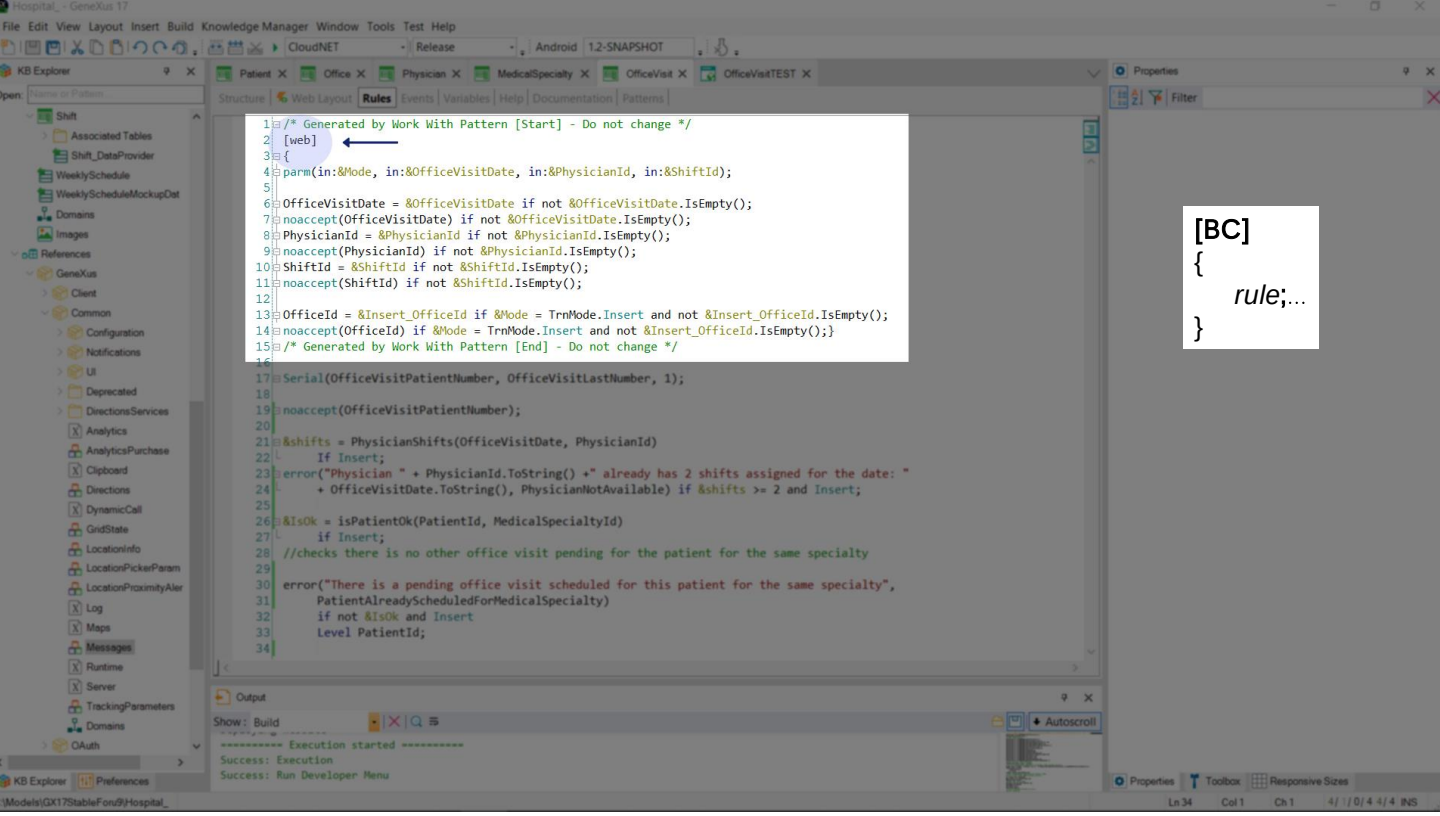
Si ahora intentamos colocar el paciente 2, que ya tiene una consulta pendiente para la misma especialidad... también nos arroja el error y tampoco graba.



Ahora sí, coloquemos el paciente 1. ¡Success!

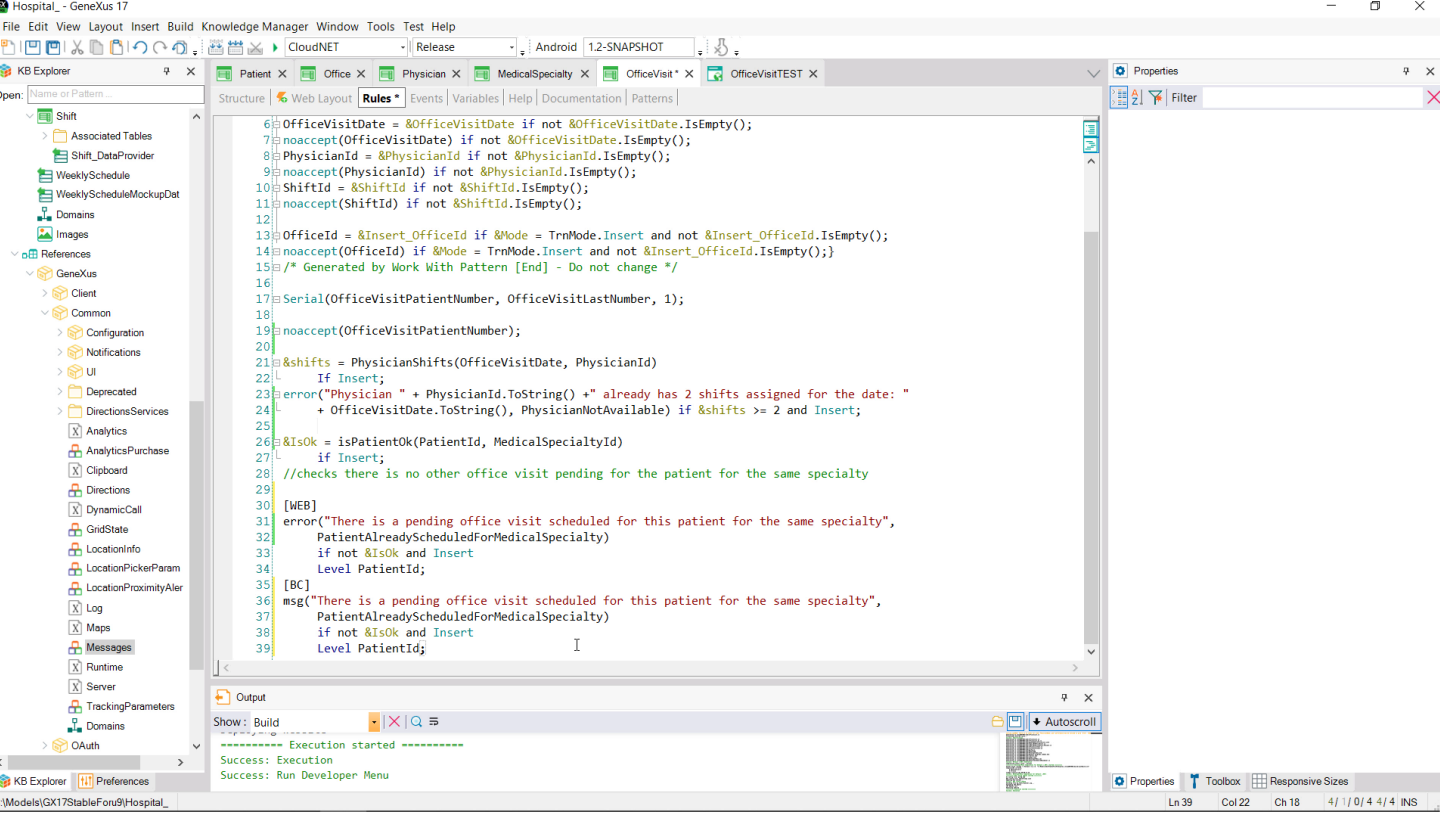


Si vamos a ver cómo quedó grabada esa consulta es indistinguible de cuando la insertamos a través de la transacción. Podemos observar que también se disparó la regla serial, dándole el número 1 al paciente y actualizando el atributo de último número dado.

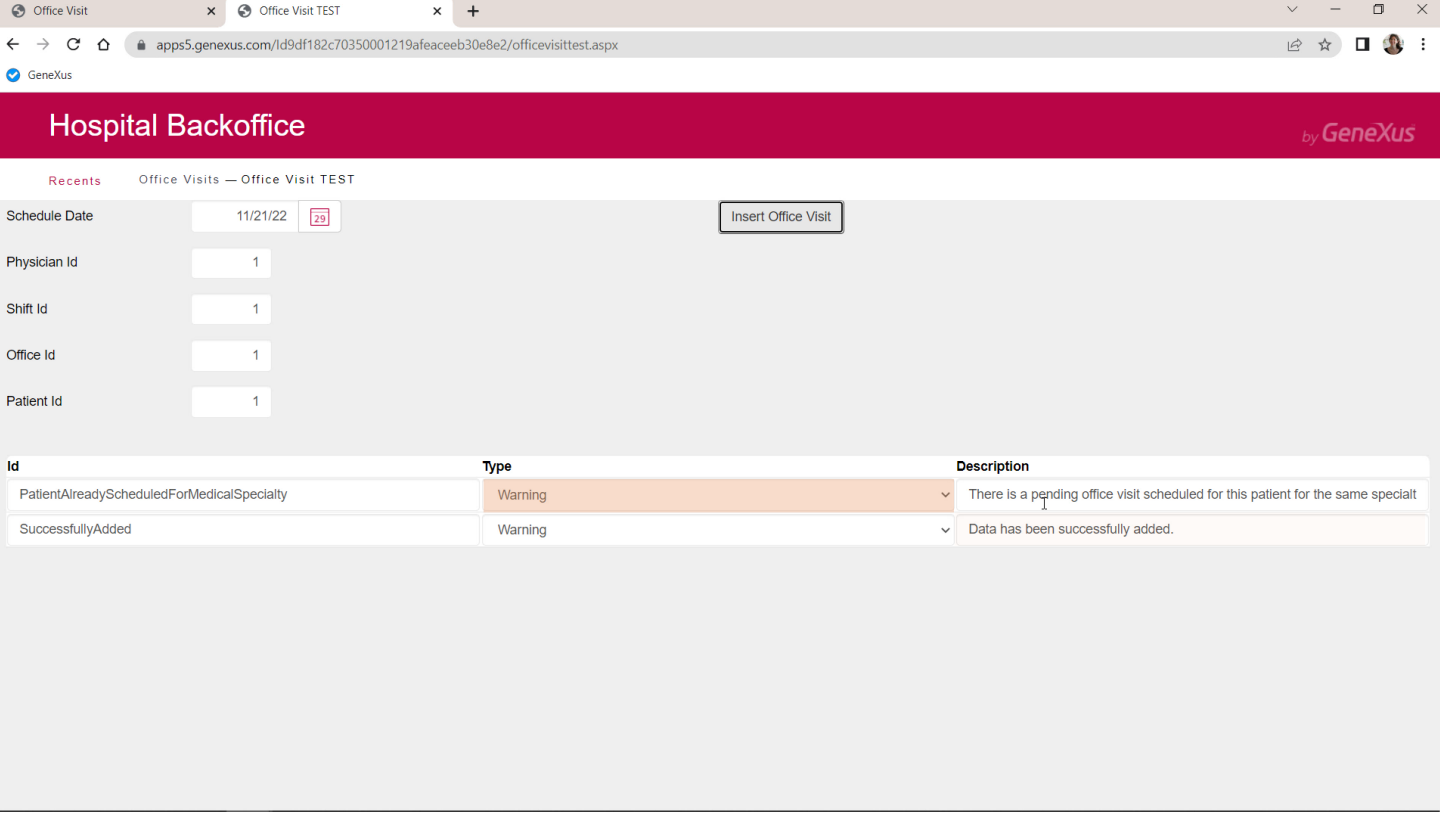


De todas las reglas de la transacción, ¿Cuáles se dispararon al insertar a través del Business Component?
Como ya sabemos se disparan todas las que no dependen de la interfaz ni tienen sentido en este contexto, como la regla parm, lógicamente, porque el Business Component no es un objeto invocable como la transacción, sino un tipo de datos especial, con métodos especiales.

En nuestro caso la regla parm no solo no se disparó porque no tiene sentido, sino porque además observemos que todo el bloque de reglas agregadas por el pattern fueron calificadas con esto, que se conoce como atributo de environment. Así se indica que solamente se incluyan estas reglas cuando se está ejecutando la transacción Web, y no cuando se ejecuta como Business component. Si, por el contrario, quisiéramos que alguna regla solo se ejecute en el contexto del Business Component, entonces podemos calificarla así. Las llaves se necesitan si son varias reglas las calificadas y no una sola.

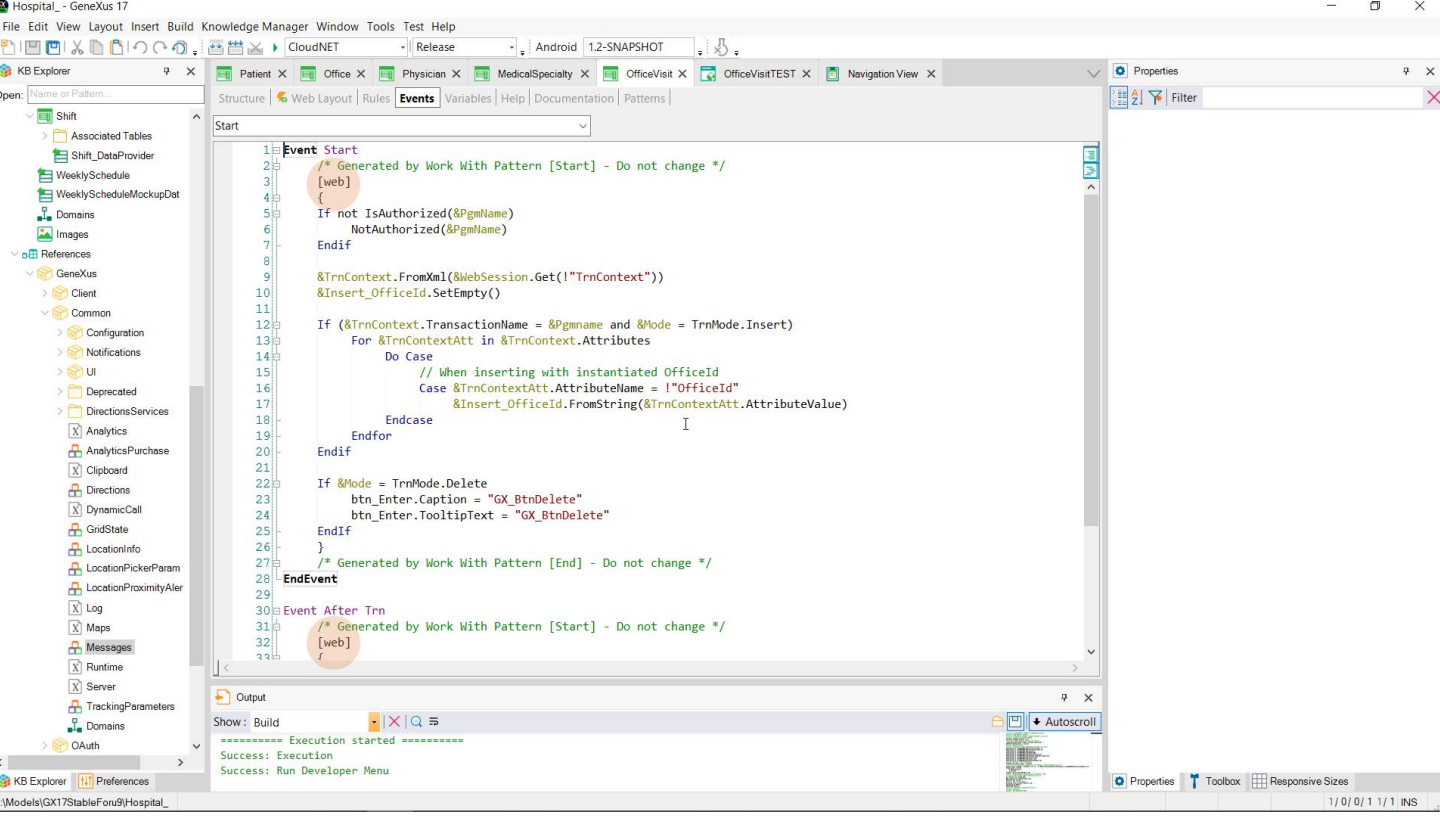


Por ejemplo, supongamos que si se inserta por Business Component no queremos impedir que se ingrese un paciente que tenga una consulta pendiente. En todo caso solo queremos arrojar un mensaje en ese caso. Entonces calificamos a la regla de error para que se ejecute solo en environment Web, y agregamos una regla message para el caso de Business Component.



Probemos. Si intentamos por la transacción, nos arroja un error, tal como esperábamos. Si, en cambio, intentamos por el Business Component, vemos que el tipo de mensaje es Warning e insertó sin inconveniente.

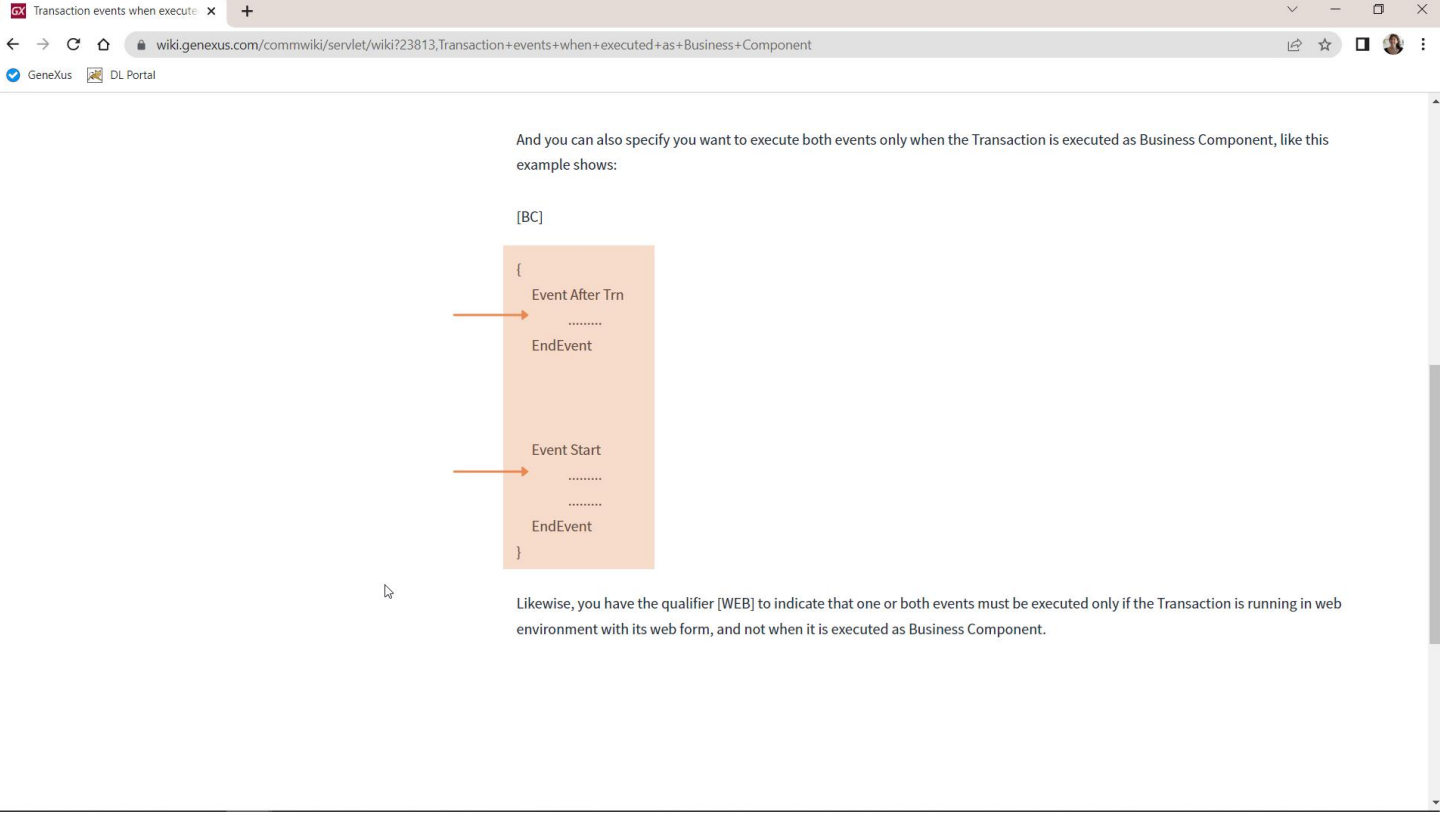
Volvamos a dejarlo como estaba.



También los eventos pueden ser calificados.

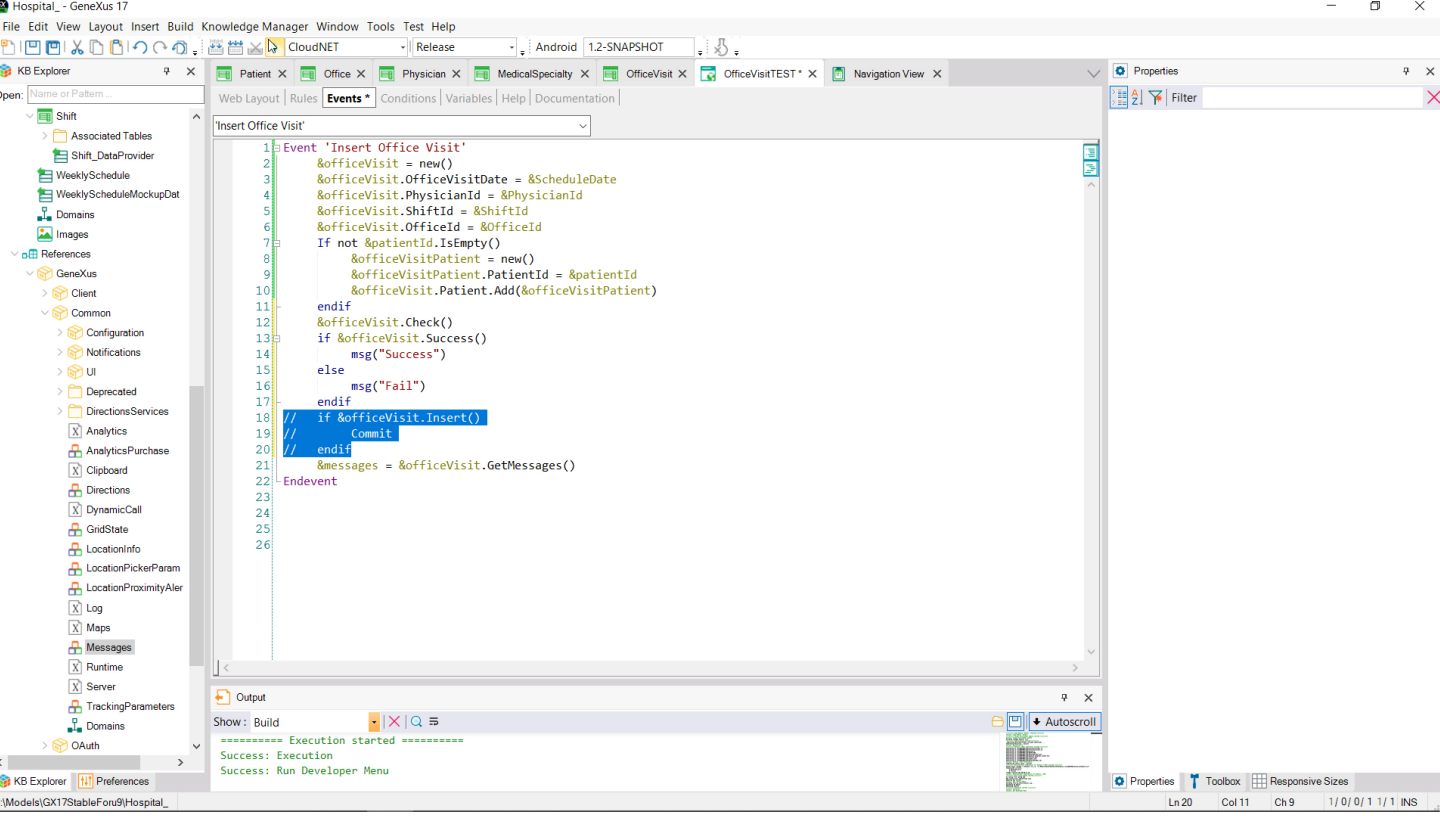
De todos los eventos definidos en una transacción ¿cuáles son ejecutados cuando se utiliza el business component? Únicamente el Start y el After Trn.
Si esos eventos incluyen referencias a objetos con interfaz de usuario, esas referencias son ignoradas.

Casualmente son los dos eventos que aquí agregó el pattern. Pero observemos que están calificados así, es decir que solo se ejecutarán en el contexto de la transacción Web y no cuando se ejecute el Business Component.



Si quisiéramos definir otras opciones para esos eventos que valgan solo para Business Component, entonces los escribiríamos precediéndolos con corchetes rectos BC.

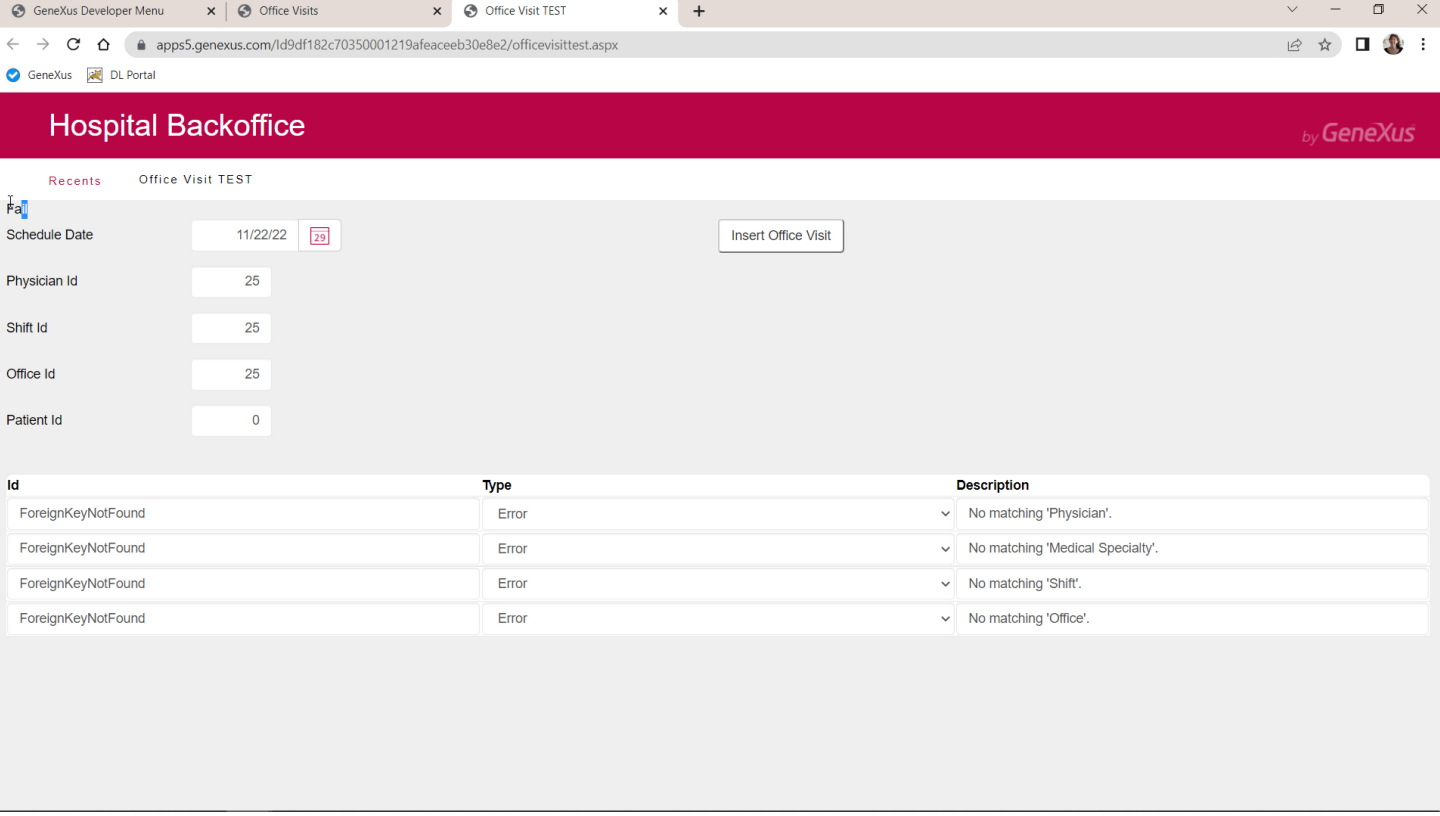
Ya sea así, englobándolos, o dentro de cada uno, diferenciando lo que es web de lo que es bc.



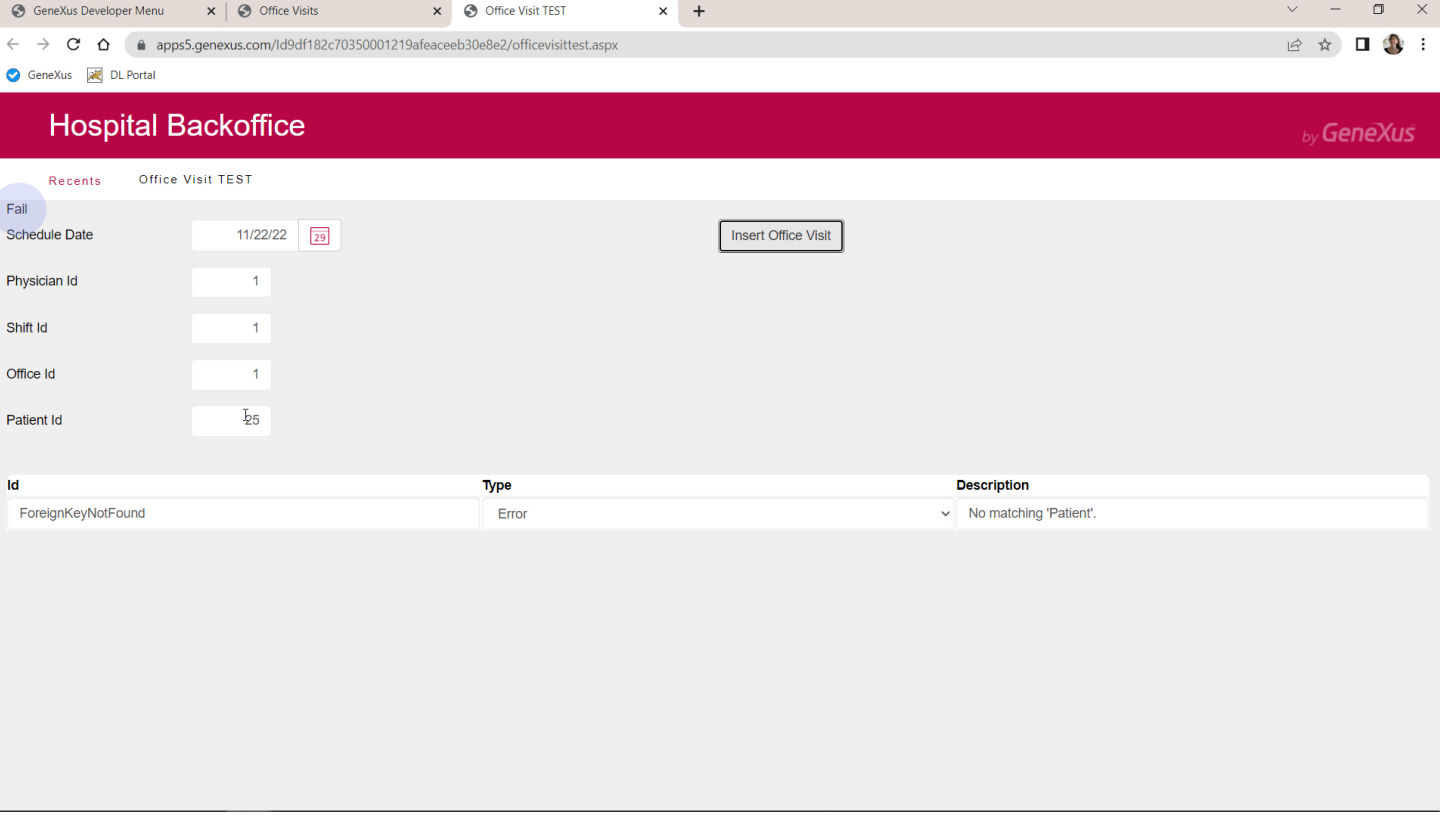
Volvamos a la inserción con el BC. Podríamos querer hacer las mismas validaciones que hará el Insert, pero sin actualizar la base de datos. Para ello contamos con el método Check.

Lo invocamos y luego chequeamos el resultado de la operación, a través del método Success que dará true si fue exitosa y false en caso contrario. Detengámonos aquí un momento: el resultado de aplicar este método a un BC va a referir al resultado de la última operación que se haya efectuado sobre el BC. Aquí es Check, pero podría haber sido Load, o Delete, o Save, o incluso la de Insert o Update o InsertOrUpdate. En particular estos últimos métodos ya devuelven su resultado, por lo que no necesitamos el Success para conocerlo y por ello realizamos la operación y preguntamos por su resultado en la misma sentencia.
Aunque no es imprescindible, también tenemos el método opuesto, Fail.

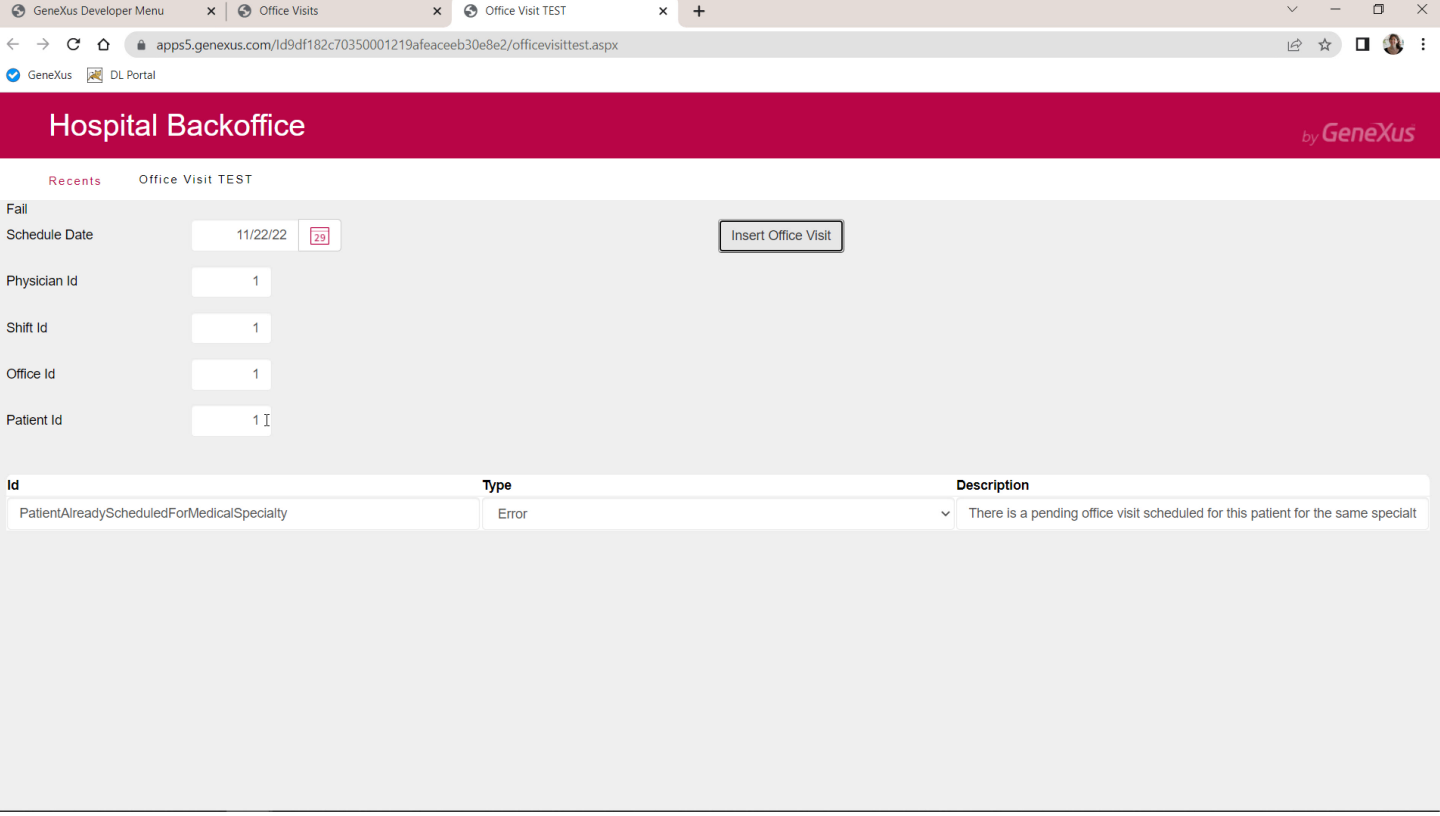
Indicaremos con un mensaje al usuario si el chequeo fue o no exitoso. Para probar dejemos comentada la actualización a la base de datos. Ejecutemos.



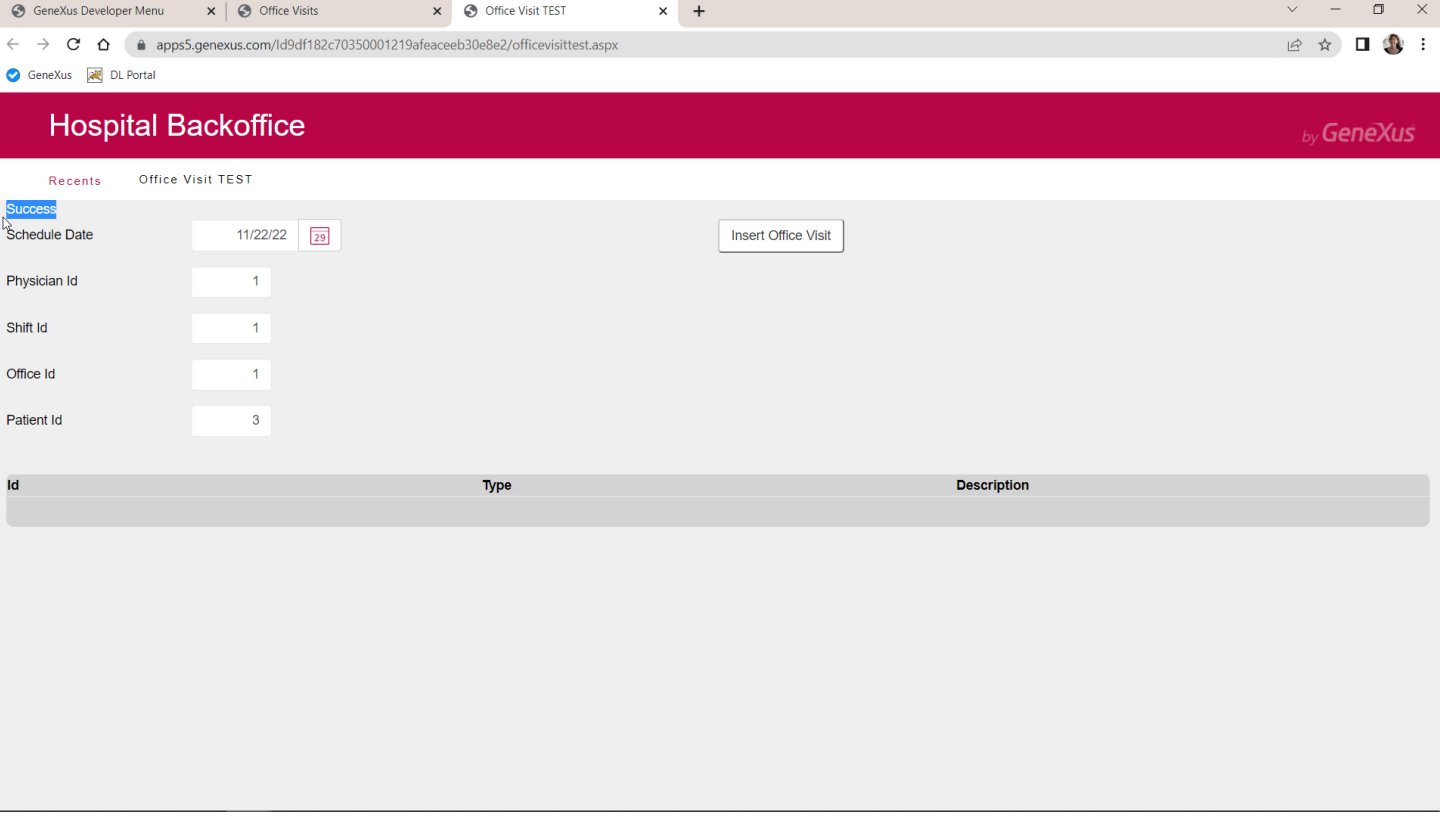
La última fecha de consulta agendada es esta. Intentemos chequear un ingreso para el día siguiente. Coloquemos médico inexistente, turno inexistente, consultorio inexistente. Vemos los mensajes de error y el mensaje Fail.



Ahora coloquemos todo correcto para el primer nivel y un paciente inexistente. Vemos el mensaje de error y el mensaje Fail.

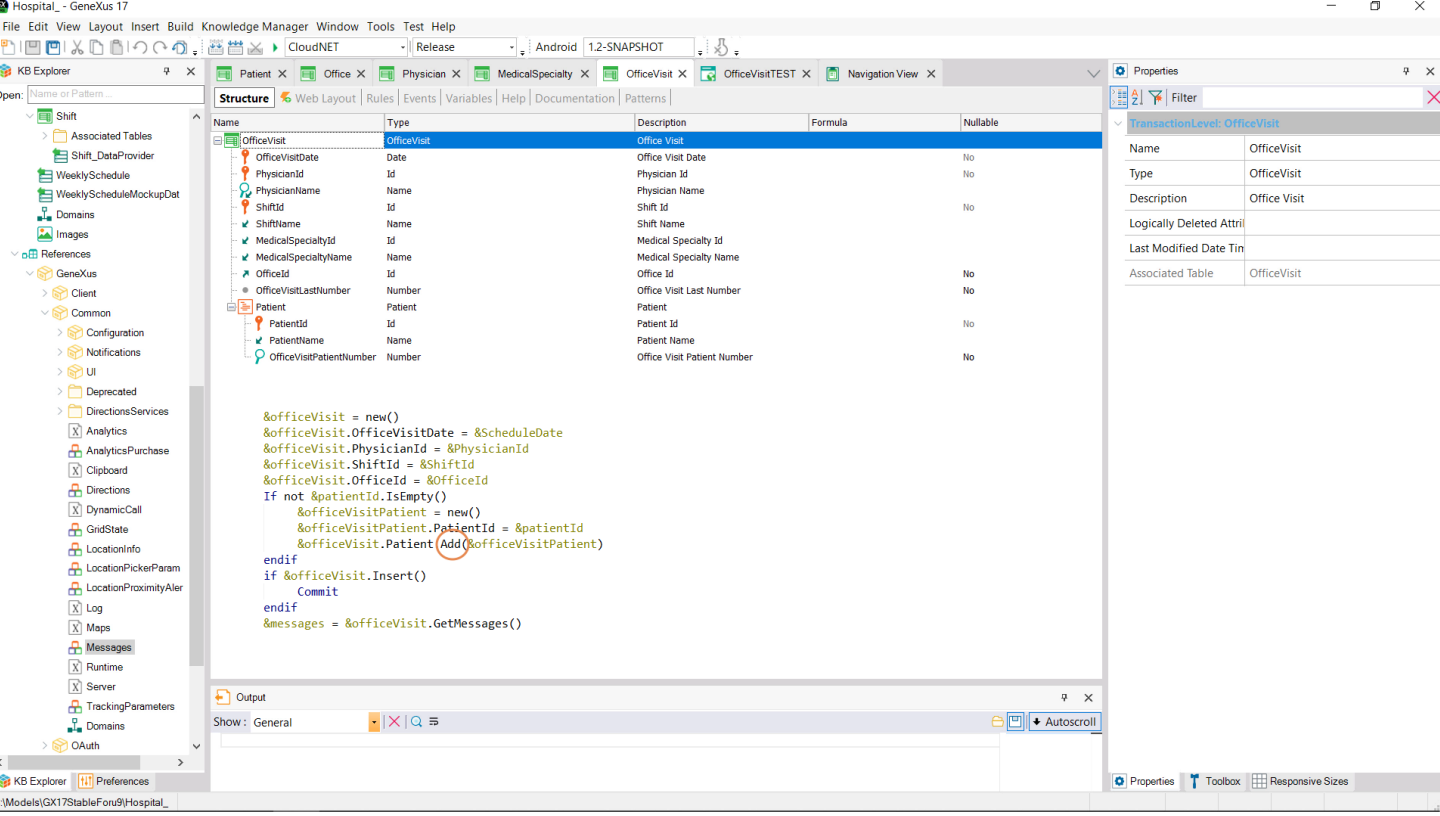


Y con el paciente 1 sabemos que se nos tiene que disparar la regla de error. Bien.



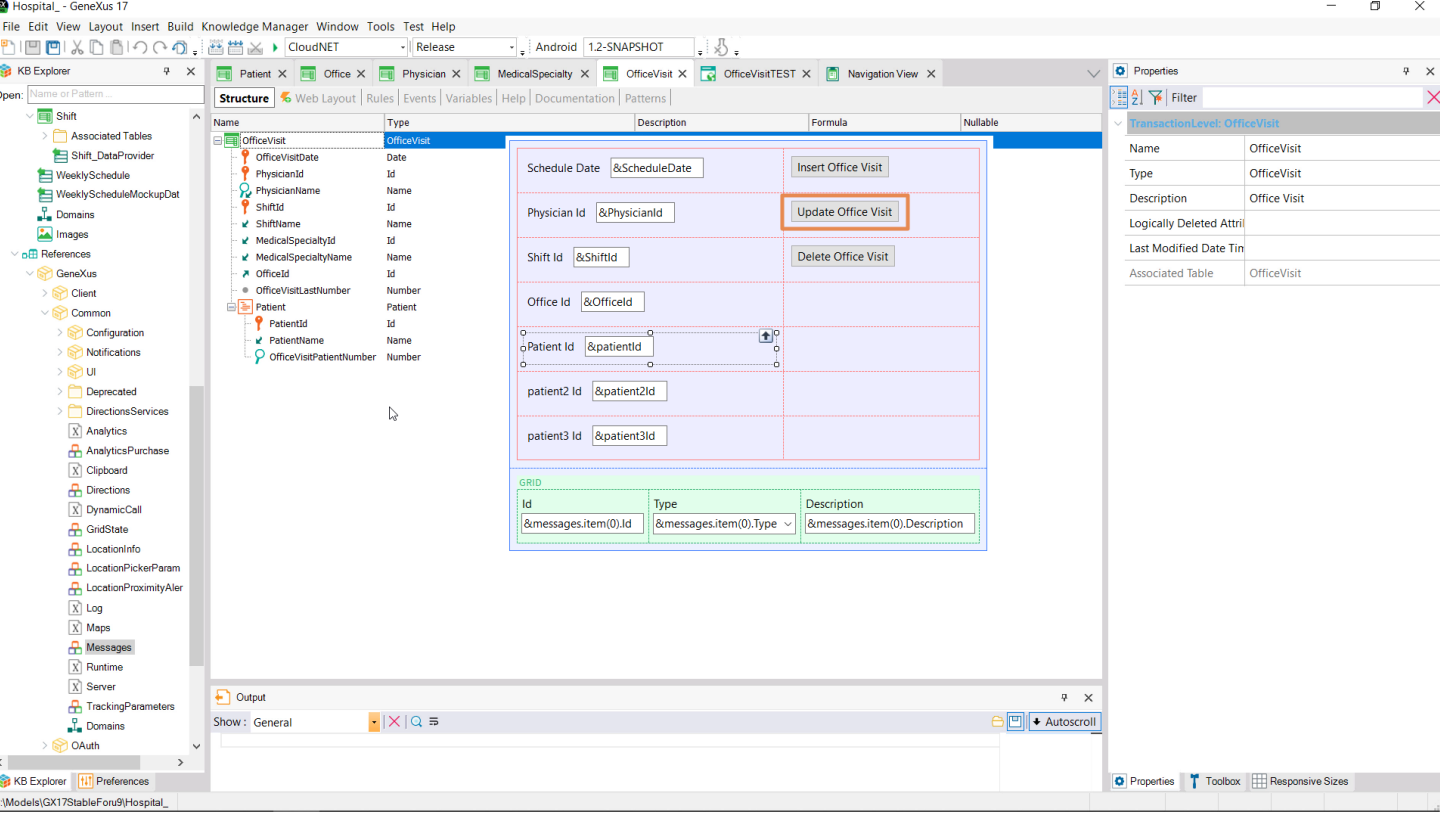
¿Y si ahora elegimos uno correcto? Vemos el mensaje Success.

Lógicamente no se ingresó nada en la base de datos.



Bien, volvamos a dejar todo como estaba.

Hasta aquí vimos que a la hora de los chequeos y ejecución de reglas de negocio y eventos no hay casi diferencia entre ingresar un cabezal y sus líneas a través de la transacción y hacerlo a través del business component. Además repasamos cómo se trabaja con el business component y en particular usamos la forma de agregar líneas con el método Add de la colección. Luego repasaremos otra, con Data Provider.



Una aclaración: aquí testeamos el funcionamiento del Business Component manualmente, a través de un web panel que construimos especialmente e íbamos probando en ejecución. Pero la forma de testear esto de modo que siempre pueda volver a probarse ante cualquier cambio o incluso en tiempos de integración, es con los tests unitarios. No se aconseja hacer pruebas manuales como las que aquí hicimos.

En el próximo video querremos modificar una consulta, no ya insertarla. Modificarla implica modificar datos del cabezal y también de las líneas: insertar nuevas, modificar existentes y eliminar otras. Y veremos, como hicimos aquí, que se disparan las reglas. También analizaremos la eliminación.

Podrías saltearte ese video si crees que dominas estos casos y pasar al siguiente.

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications