### Documento Preliminar

# PROTOCOLO DE DESARROLLO EN BASES DE DATOS INDAP

## Introducción

Éste documento pretende establecer convenciones de creación, programación y consumo de objetos de bases de datos pertenecientes a la institución, en especial aquellos elementos pertenecientes al naciente Sistema Único.

Contemplará normas y directrices desde la construcción de objetos en ambiente de desarrollo hasta su implantación final en ambiente de producción.

## Roles y Responsabilidades

A continuación se detallará una estructura colaborativa, no jerárquica, para la construcción de bases de datos:

### Arquitecto de BD

Principal encargado de velar por la consistencia y robustez del modelo de datos del Sistema Único. Es el encargado de modelar e implementar objetos de BD en el esquema BDI, revisar y visar los cambios propuestos por los Encargados de BD de Proyecto y Desarrolladores en el esquema BDI, analizando su impacto en el resto del proyecto y decidirá si procede su implementación en conjunto con el Jefe de Proyecto. También deberá servir de referente en cuanto a solución de dudas en torno a la creación de objetos de BD y prestar asistencia en torno al desarrollo de dichas funcionalidades. Será responsable de revisar y supervigilar la implantación de soluciones de BD en ambiente de QA-Testing y de apoyar en la implantación final en ambiente de Producción. Será responsable de mantener la documentación del esquema BDI y procurar que la documentación de otros esquemas sea proveída adecuadamente.

### Encargado de BD de Proyecto

Su tarea principal es de Modelar y generar el script de creación de esquemas y objetos de base de datos para el proyecto que se le ha solicitado. También deberá prestar asistencia a los desarrolladores respecto de los objetos creados y ayudarlos en la optimización de sus consultas. Será responsable de generar la documentación adecuada para el proyecto en que se encuentra involucrado.

### Desarrollador

Encargado de programar funciones y/o procedimientos almacenados para sustentar la funcionalidad que le ha sido solicitada, basándose en el modelo entregado por el Encargado de BD de Proyecto. Apoyará al Encargado de BD de Proyecto en la generación de la documentación sobre las funcionalidades desarrolladas.

## Convenciones de Desarrollo en Base de Datos

Las siguientes convenciones pretenden servir de guía al momento de crear y mantener objetos de bases de datos, de manera que quienquiera que observe o necesite realizar una modificación, tenga clara su estructura.

### Nomenclatura

Se utilizará la siguiente convención de nomenclatura para nombrar los objetos de BDD y sus atributos:

#### Palabras Reservadas

Se sugiere su escritura en mayúscula.

#### Bases de Datos

Su nombre debe ser en singular. De ser de dos o más palabras, se debe separar por underscore, en minúsculas, sin tildes ni caracteres especiales. No debe comenzar con un número, tratando de evitar el uso de éstos. Considerar un nombre descriptivo acorde al proyecto general en el que se encuentra incluida.

#### Esquemas

Su nombre debe ser en singular. De ser de dos o más palabras, se debe separar por underscore, en minúsculas, sin tildes ni caracteres especiales. No debe comenzar con un número, tratando de evitar el uso de éstos. Considerar un nombre descriptivo acorde al proyecto o módulo para el cual está siendo construido.

#### Tablas

Su nombre debe ser en singular. De ser de dos o más palabras, se debe separar por underscore, en minúsculas, sin tildes ni caracteres especiales. No debe comenzar con un número, tratando de evitar el uso de éstos. Debe utilizar el esquema que corresponda al proyecto o módulo para el cual está siendo construido, con excepción de aquellas tablas que involucren datos transversales a todos los módulos del sistema, ejemplo: bdi. Los atributos (campos) de una tabla, deben cumplir con las mismas consideraciones.

#### Restricciones

Su nombre debe ser en singular. De ser de dos o más palabras, se debe separar por underscore, en minúsculas, sin tildes ni caracteres especiales. No debe comenzar con un número, tratando de evitar el uso de éstos. Debe considerar el prefijo “pk\_” para las Claves Primarias y “fk\_” para la Clave Externa, seguido del nombre del esquema, seguido del nombre de la tabla, seguido del (los) campo(s) que componen la restricción. Se sugiere el indicar tanto las PK como las FK al final de las definiciones de campos en el script de creación. No use columnas de tipo FLOAT, REAL o DATETIME como candidata a FK.

CREATE TABLE bdi.persona(

 id\_persona INT NOT NULL IDENTITY(1,1),

 persona VARCHAR(2000) NOT NULL,

 vigente BIT NOT NULL DEFAULT 1,

 fecha\_nacimiento DATETIME NOT NULL,

 sueldo\_base BIGINT NULL,

 id\_tipo\_persona INT NOT NULL,

 CONSTRAINT pk\_bdi\_persona\_id\_persona PRIMARY KEY (id\_persona),

 CONSTRAINT fk\_bdi\_persona\_id\_tipo\_persona FOREIGN KEY (id\_tipo\_persona) REFERENCES bdi.tipo\_persona(id\_tipo\_persona));

#### Índices

Todo índice que deba crearse para la optimización de consultas a la tabla, debe utilizar el prefijo “ix\_” al comienzo de éste, seguido del nombre del esquema, seguido por el nombre de la tabla y de los campos involucrados en el índice, separados por underscore. Esto busca mantener la coherencia con las otras convenciones de nomenclatura propuestas. Ésta exigencia no es necesaria de aplicar a la PK de una tabla, ya que el motor crea un índice cluster con ella y, en caso de presentar excepciones por violación de PK, inmediatamente identifica que se trata de una Primary Key.

CREATE INDEX ix\_persona\_apellido\_paterno\_apellido\_materno

ON bdi.persona(apellido\_paterno, apellido\_materno);

#### Procedimientos Almacenados

Su nombre debe ser descriptivo respecto de la actividad principal que realiza (obtención de datos, modificación, eliminación, etc.). De ser de dos o más palabras, se debe separar por underscore, en minúsculas, sin tildes ni caracteres especiales. No debe comenzar con un número, tratando de evitar el uso de éstos. Debe considerar no utilizar prefijos, además de utilizar el esquema que corresponda al proyecto o módulo que lo consumirá, a excepción de aquellos transversales al sistema que pertenecerán al esquema bdi. Debe incluirse un resumen del procedimiento como cabecera de éste, indicando autor, fecha de creación, modificado por, fecha última modificación, descripción general y cualquier otra descripción especial relativa a los parámetros de entrada que deban funcionar de manera “especial”, para servir de guía a quien desee consumir dicha funcionalidad. Se sugiere utilizar los vocablos “get”, “set”, “list”, “del”, para poder identificar su función principal.

Ejemplos:

* bdi.p\_persona\_get 🡪 Obtendría un registro de la tabla persona.
* bdi.p\_persona\_set 🡪 Realizaría Insert o Update en la tabla persona.
* bdi.p\_persona\_upd 🡪 Realizaría Update en la tabla persona (en caso de necesitar separar del Insert).
* bdi.p\_persona\_del 🡪 Eliminaría registros en la tabla persona.
* bdi.p\_persona\_list 🡪 Obtendría un listado de los registros de la tabla persona.

Adicionalmente, se debe evitar el uso del prefijo “sp”, ya que el motor los reconoce como un procedimiento almacenado del motor, esto se traduce en que éste busca el procedimiento almacenado en la base MASTER, y al no encontrarlo recurre a la base activa. Este paso innecesario puede provocar bajas de rendimiento del motor frente a una alta demanda de estos procedimientos.

#### Funciones

Su nombre debe ser descriptivo de los datos a obtener o el proceso que realizará. De ser de dos o más palabras, se debe separar por underscore, en minúsculas, sin tildes ni caracteres especiales. No debe comenzar con un número, tratando de evitar el uso de éstos. Debe considerar usar prefijos:

* ft\_ : para funciones tipo tabla
* fe\_: para funciones escalares ( que solo devuelven un valor)

Además debe utilizar el esquema que corresponda al proyecto o módulo que lo consumirá, a excepción de aquellos transversales al sistema que pertenecerán al esquema bdi. Debe incluirse un resumen del procedimiento como cabecera de éste, indicando autor, fecha de creación, modificado por, fecha última modificación, descripción general y cualquier otra descripción especial relativa a los parámetros de entrada que deban funcionar de manera “especial”, para servir de guía a quien desee consumir dicha funcionalidad (ver plantilla sugerida más adelante).

#### Vistas

Su nombre debe ser en singular. De ser de dos o más palabras, se debe separar por underscore, en minúsculas, sin tildes ni caracteres especiales. No debe comenzar con un número, tratando de evitar el uso de éstos. Debe considerar el prefijo “v\_”. Debe utilizar el esquema que corresponda al proyecto o módulo para el cual está siendo construido, con excepción de aquellas vistas que involucren datos transversales a todos los módulos del sistema, ejemplo: bdi. Los atributos (campos) de una vista, deben cumplir con las mismas consideraciones. **Evitar su uso en la medida de lo posible.**

### Plantilla para Funciones y/o Procedimientos Almacenados

Se sugiere la utilización de la siguiente plantilla para la construcción de Funciones y Procedimientos almacenados en SQL Server. Esta incluye el encabezado propuesto para información de la funcionalidad y el flujo principal de control de excepciones. **Éste último no será exigible en aquellas funciones o procedimientos almacenados que solo rescaten datos**, pero será de uso obligatorio si existe alguna instrucción INSERT, UPDATE o DELETE. El desarrollador debe sentirse libre de agregar cualquier comentario al interior del procedimiento o función para explicar el código y facilitar su mantención.

-- ==================================================================

-- Author: <nombre, apellidos>

-- Create date: <fecha>

-- Modified by: <nombre, apellido> de quien realizó la última modificiación.

-- Last Modified: <fecha> última modificación.

-- Description: <brebe explicación de la funcionalidad y/o caso(s) que cubre>

-- Parameters: <breve descripción de parámetros que requieran alguna explicación especial>

-- ==================================================================

CREATE PROCEDURE esquema.nombre

 --parametros

AS

BEGIN

 BEGIN TRANSACTION

 BEGIN TRY

 --acciones

 COMMIT TRANSACTION

 END TRY

 BEGIN CATCH

 ROLLBACK TRANSACTION

 -- Parametros de Errores

 DECLARE @ErrorMessage NVARCHAR(4000),

 @ErrorSeverity INT,

 @ErrorState INT;

 SELECT @ErrorMessage = ERROR\_MESSAGE(),

 @ErrorSeverity = ERROR\_SEVERITY(),

 @ErrorState = ERROR\_STATE();

 RAISERROR (@ErrorMessage, -- Message text.

 @ErrorSeverity, -- Severity.

 @ErrorState -- State.

 );

 END CATCH

END

GO

#### Tablas Paramétricas

Se recomienda que para cualquier tabla paramétrica, o similar, que tenga un nombre descriptivo y evitar el uso de un campo llamado simplemente “id”, y un campo llamado “descripción”. A cambio se sugiere utilizar el mismo nombre de la tabla de la siguiente manera:

CREATE TABLE bdi.tipo\_persona

(

 id\_tipo\_persona INT NOT NULL IDENTITY(1,1),

 tipo\_persona VARCHAR(500) NOT NULL,

 CONSTRAINT pk\_tipo\_persona\_id\_tipo\_persona PRIMARY KEY(id\_tipo\_persona)

);

#### Indentación

Se recomienda, para una mayor comprensión, un uso adecuado de la indentación en las consultas, procedimientos, funciones, etc., de manera de procurar un orden mínimo para facilitar las posteriores labores de mantención y/o análisis de consultas. El siguiente script puede servir de ejemplo de un uso adecuado de la indentación, y no pretende ser un modelo estricto, ya que el objetivo es el orden y entendimiento del código, mientras esto sea alcanzable, el desarrollador tendrá la libertad de usar su propio esquema. Puede consultar al “Arquitecto de BDD”, o a Encargados de BDD de Proyecto, si su esquema es adecuadamente inteligible.

SELECT

 usuario\_indap.rut

 ,usuario\_indap.digito\_verificador

 ,usuario\_indap.fecha\_ultima\_actualizacion

 ,tipo\_usuario.id\_tipo\_usuario

 ,tipo\_usuario.descripcion AS tipo\_usuario

 ,usuario\_indap.id\_estado\_usuario

 ,estado\_usuario.descripcion AS estado\_usuario

 ,CASE UI.id\_estado\_usuario

 WHEN 1 THEN 'Estado 1'

 WHEN 2 THEN 'Estado 2'

 ELSE 'No definido'

 END AS otro\_estado

FROM

 bdi.usuario\_indap AS usuario\_indap

 INNER JOIN bdi.tipo\_usuario AS tipo\_usuario

 ON usuario\_indap.id\_tipo\_usuario = tipo\_usuario.id\_tipo\_usuario

 INNER JOIN bdi.estado\_usuario AS estado\_usuario

 ON usuario\_indap.id\_estado\_usuario = estado\_usuario.id\_estado\_usuario

WHERE

 usuario\_indap.id\_tipo\_usuario = 2

ORDER BY

 usuario\_indap.rut

#### Alias

Se recomienda el uso de aliases en la construcción de consultas, ya que permite el ordenamiento y mejor identificación de los campos a obtener y filtros a aplicar. Es importante tener en cuenta que los aliases deben ser adecuadamente descriptivos, y si ha seguido las convenciones para un correcto modelado de la base, el nombre de la tabla es suficientemente descriptivo, use éste como alias. Note el ejemplo anterior.

#### Parámetros de Salida (Output)

Se recomienda utilizar parámetros de salida (output) para recuperar valores que requieran diferenciarse de la salida general de un procedimiento almacenado con valores de tabla. Si su procedimiento solo retornará un valor por medio de un parámetro output, reemplace su procedimiento por una función escalar que retorne el valor. Bajo ninguna circunstancia utilice un parámetro output para devolver un mensaje de error o un indicador de la ejecución de su procedimiento, utilice para ello la estructura propuesta para el control de errores (Plantilla para Procedimientos almacenados).

#### Deshabilitar el conteo de filas

Se recomienda el uso de la instrucción **SET NOCOUNT ON/OFF** dentro del bloque de un procedimiento almacenado cada vez que no sea estrictamente necesario realizar el conteo de filas afectadas por una instrucción DML. Esto puede ayudar a mejorar el rendimiento del servidor en relación a los procedimientos en ejecución. Recuerde siempre usar **SET NOCOUNT OFF** al finalizar el procedimiento almacenado.

#### No utilizar SELECT \*

Se hace hincapié en el hecho de declarar las columnas de salida de una instrucción select. Esto impide al optimizador de consultas del motor hacer un uso eficiente de los índices, además de ser una fuente de riesgo de futuras fallas si la tabla consultada cambia su estructura.

#### No utilizar cursores

Ya que los cursores pueden ser usados para modificar datos fila a fila, éstos utilizan de manera muy desmedida los recursos del servidor, además de generar bloqueos. Recordemos que los cursores en SQL SERVER son por defecto FOR UPDATE. Por ello, busque alternativas al uso de cursores, como por ejemplo: ciclos WHILE, tablas derivadas, instrucciones CASE, etc. Si aún no tiene alguna otra alternativa, y solo desea recorrer un recordset, recuerde que puede usar las instrucciones READ ONLY en la declaración del cursor en sintaxis ISO, o usar la sintaxis extendida de SQL SERVER: FAST\_FORWARD, FORWARD\_ONLY o READ\_ONLY, éstas permiten evitar el bloqueo de tablas y mejorar el rendimiento del cursor.

#### Uso de tablas temporales y variables tipo tabla

Ante la necesidad de trabajar con sets parciales de datos, tenga en cuenta cuál es el propósito de su procedimiento y evalúe cuál es la mejor opción. De todas maneras, se recomienda el uso de variables tipo tabla. A continuación, se mencionará algunas características de cada tipo.

|  |  |
| --- | --- |
| Tabla Temporal | Variable Tipo Tabla |
| * Se almacena en la base TEMPDB.
* Puede generar bloqueos en sysobjects y sysindexes, pudiendo afectar a toda la instancia.
* Permite su manipulación “post-creación”, por ejemplo, para la creación de índices.
* Dentro de un procedimiento almacenado puede provocar continuas recompilaciones.
* Debe descartarse manualmente (drop).
 | * Dentro de procedimientos almacenados, provoca menos recompilaciones.
* Su estructura se almacena en memoria.
* Si el contenido es muy grande, podría almacenarse en la base TEMPDB y no en memoria.
* No le afecta la acción de la instrucción **ROLLBACK.**
* No puede ser generada con una instrucción SELECT INTO.
* No usan multiple threads (paralelismo).
* Su estructura no puede ser modificada una vez creada.
* No necesita ser descartada manualmente.
 |

#### Bloques IF-ELSE

Si requiere ejecutar bloques distintos según el valor de un parámetro, es común pensar en usar bloques IF-ELSE, sin embargo, esto genera cambios en el plan de ejecución e impide al motor generar la precompilación adecuada, ya que no contemplará todos los casos de uso. Para situaciones como ésta, es mejor generar procedimientos almacenados dedicados para cada caso. Ésta recomendación cae especialmente cuando se debe trabajar con instrucciones SELECT de gran tamaño, no afectando tanto a la performance de instrucciones INSERT o UPDATE.

## Procedimiento de Paso a Ambiente de Producción – Testing

### Consideraciones Generales

Antes de realizar un paso al ambiente de Producción, o al ambiente de Testing (o similar), debe primero asegurarse de tener todos los scripts con DML o DDL, adecuadamente identificados y se recomienda nombrarlos con el número de secuencia en el que deben ser ejecutados, especialmente si son más de un archivo, como se muestra en el siguiente ejemplo:



Asegúrese de incluir todos sus procedimientos almacenados, funciones, tablas, vistas, etc. que necesite para el correcto funcionamiento del software que consumirá tales recursos de Bases de Datos.

Se recomienda que, en caso de conocer el nombre de la base de datos de destino de éstos objetos, incluya la instrucción “USE” correspondiente al inicio del script.

### Paso a Ambiente de Testing

Teniendo en cuenta las consideraciones antes mencionadas, para el paso a un ambiente de Testing o similar, siga los siguientes pasos.

1.- Los desarrolladores, previa recopilación de sus scripts con DDL y/o DML, deben entregar sus archivos al Encargado de BD de Proyecto.

2.- El Encargado de BD de Proyecto, revisará la consistencia de los archivos entregados validando la secuencia indicada por el desarrollador para su correcta ejecución, procurando si hay necesidad de intercalar con la secuencia de otro desarrollador. Una vez hecha tal comprobación, entregará los archivos al Arquitecto de BD. Deberá indicar al Arquitecto cuales son los esquemas y/o objetos críticos que se verán afectados durante la ejecución de los archivos.

3.- El Arquitecto de BD ejecutará los scripts entregados siguiendo la secuencia indicada y entregará reporte con los resultados de la ejecución al encargado de BD de Proyecto. Si existe algún error que ponga en riesgo la integridad de la BDD, dará término a la ejecución del set entregado y devolverá los archivos al Encargado de BD de Proyecto para que los analice junto con el Desarrollador. El Arquitecto deberá revisar los scripts previo a su ejecución si en el proceso de entrega de los archivos, el Encargado de BD de Proyecto le indica que se verá afectado algún esquema u objeto que se considere crítico para la integridad de la base de datos, y podrá negar la ejecución de dicho script si esta se ve muy comprometida.

### Paso a Ambiente de Producción

Una vez que la ejecución de los scripts haya sido validad durante el proceso de Paso a Ambiente de Testing, y bajo la supervisión del Jefe de Proyecto, El Arquitecto de BDD hará entrega de los archivos, secuenciados adecuadamente, para su ejecución en Ambiente de Producción al Administrador de Base de Datos Institucional. Esta entrega se hará por medio de correo electrónico, con copia al Jefe de Proyecto, Jefe de Desarrollo y Soporte.

El DBA, una vez finalizada la ejecución de los scripts deberá entregar respuesta, por la misma vía, sobre los resultados de ejecución de los archivos.

El DBA podrá tener veto sobre la ejecución de los scripts si considera que la integridad de la BDD se ve muy comprometida. Si esto ocurre, deberá informar sobre los riesgos que ha identificado al Arquitecto de BD, quien hará la revisión en conjunto con el Jefe de Proyecto y el Encargado de BD de Proyecto.