Política de Estándares de Codificación de Bases de Datos

	Responsable	Cargo	Fecha
AUTOR	Raúl García Barrientos	Administrador de Base de Datos	03/10/2025
REVISADO POR	Christian Villaverde / Daniel Carhuamaca	Team Desarrollo	
APROBADO POR	Patricio Sandoval	Gerente TI	

Control del Documento

Fecha	Autor	Versión	Referencia de Cambio
03/10/2025	Raúl García	1.0	Creación del documento

Objetivo

El presente documento tiene como objetivo establecer y formalizar un lenguaje (código y estructura) común para la implementación de objetos de base de datos en Migiva y sus empresas, permitiendo un fácil entendimiento y mantenimiento en el desarrollo de los proyectos.

Alcance

El alcance del documento involucra a todos aquellos proyectos y áreas que tengan que ver con el desarrollo y mantenimiento de aplicaciones de Base de Datos ya sea que estén en producción o en fase de proyectos.

Documentos Asociados

Definiciones

BD: Base de Datos.

DBA: Administrador de base de datos.

TI: Área de Tecnologías de Información.

ANSI: American National Standards Institute.

DBMS: Data Base Management System, Sistema gestor de bases de datos.

DCL: Data Control Language, Lenguaje de Control de Datos.

DDL: Data Definition Language, Lenguaje de Definición de Datos.

DML: Data Modification Language. Lenguaje de Modificación de Datos.

ER: Entidad Relación.

FK: Foreign Key. Llave foránea.

PK: Primary Key. Llave primaria.

SGBD: Véase DBMS.

SO: Sistema Operativo.

SQL: Structured Query Language. Lenguaje Estructurado de Consultas.

Nomenclatura

Consideraciones Generales

1. Los nombres de los objetos deben ser lo más corto posible, fáciles de leer, y lo más descriptivo posible, evitando términos ambiguos o que se prestan a distintas interpretaciones.
2. Los nombres deben incluir solo caracteres del alfabeto español excepto vocales con acento, eñes y diéresis, y no deben utilizarse caracteres especiales (‘#’, ‘/’, ‘;’, ‘%’, ‘+’, etc) ni espacios, el único carácter especial que se permitirá y exclusivamente en los casos que se especificarán posteriormente será el underscore ‘_’ ; el uso de números debe evitarse de ser posible.
3. Se usará el **CamelCase** como estilo de escritura. El **LowerCamelCase** para diferentes nombres de los objetos de las bases de datos.
4. Se usará el lenguaje español como el lenguaje para los diferentes nombres de los objetos.

Nota: Existen nombres en inglés, se debe respetar los nombres origen de los campos.

Abreviaturas Recomendadas

A continuación se muestra una lista con las abreviaturas que pueden ser usadas como prefijos para dar nombre a un objeto. Aquí algunos ejemplos:

Descripción	Nemónico
AGRICOLA	AGRI
ARANDANO	ARA
CALIDAD	CAL
CLIENTE	CLI
COMERCIAL	COM
CONCEPTOS	CON
CONTROL	CTR
COSECHA	COS
DEFECTO	DEF
DESHIDRATACION	DHID
DETALLE	DET
DOCUMENTO	DOCU
EMBARQUE	EMB
ENFERMEDAD	ENF
EVALUACION	EVAL
FACTURA	FT
FECHA	FEC
FORMULARIO	FORM
GASIFICADO	GAS
GRUPO	GRP
INDIVIDUAL	INDV
INDUSTRIAL	INDU
INSPECCION	INSP
INSTRUCTIVO	INST
MADUREZ	MDZ
MATERIAL	MAT
MIGRACION	MIG
MOVIMIENTO	MOV
NOTA CREDITO	NC
NOTA DEBITO	ND

Descripción	Nemónico
PACKING	PKG
PAGO	PAG
PARAMETRO	PARAM
PLANTILLA	PLA
PRECOSECHA	PCOS
PROCESO	PROC
PRODUCCION	PROD
PROGRAMACION	PROG
PROYECCION	PROY
REPORTE	REP
SEGUIMIENTO	SEG
SOLICITUD	SOL
STOCK	STK
SUPERVISION	SPV
TUNELIZADO	TUN

Usuarios

Usuario Owner (Propietario)

Se definirán con la nomenclatura “UCOWN_” seguido del nombre del servicio, un nombre representativo o abreviatura.

UCOWN_ [Nombre del Servicio] => UCOWN_SISPACKING

Usuario de Servicio (Sistema, Aplicación o Conexión)

Se definirán con la primera letra “UCSER” seguido del nombre del servicio, un nombre representativo o abreviatura.

UCSER_ [Nombre del Servicio] => UCSER_SISPACKING

Usuario de Soporte

Se definirán con el prefijo "UCSOP" seguido del nombre del personal de TI

UCSOP_ [Inicial nombre del personal] [Apellido del personal] => UCSOP_RGARCIA

Usuario de Responsabilidad

Se definirán con el prefijo "UCRES" seguido del nombre del personal de TI

UCRES_[Inicial nombre del personal] [Apellido del personal] => UCRES_RGARCIA

Usuario de Link Server

El prefijo a usar para este usuario es UCLNK_

UCLNK_[Nombre de BD o SCHEMA] _ [Nombre de Instancia] =>UCLNK_PACKING_CAL

Usuarios de Uso Temporal

Contendrán el prefijo "TMP" en el nombre del usuario según los siguientes casos:

Usuarios temporales para responsabilidad.
UCRES_[Inicial nombre del personal] [Apellido del personal] _TMP => UCRES_RGARCIA_TMP

Usuarios temporales de servicio asignado a una persona,

UCSER_[Inicial nombre del personal] [Apellido del personal] _TMP => UCSER_RGARCIA_TMP

Para otros casos:

UCTMP_[Inicial nombre del personal] [Apellido del personal] _ [Nombre del servicio] => UCTMP_RGARCIA_VENTURA

Esquema

Los nombres de los esquemas irán en base al área, proceso, módulo, sistema que especifique el alcance de la necesidad. Los siguientes son esquemas de base de datos:

Esquema
agroamigo
agrocom
archivo
asist
audit
calidadAgricola
Comercial
commerce
cultivador
EMBARQUE
erp
evalAgri
grower
history
HumanResource
Integraciones
interp
lora
Maestra
mast
package
packing
phytosanitary
planta
ppp
proper
pwbi
quality
qualitycrop
recursoHumano

Esquema
report
sales
shipping
util
wow

Tablas

Las tablas según sea el caso deben se definirán de la siguiente manera:

- Comienzan con el ***nombre*** de la entidad
- Luego la ***Descripción*** corresponde a una descripción de la entidad y/o uso de la tabla.

Comunes

[nombreDescripción] => inspeccionCosechaUvaSupervisor

Externas

[nombreDescripción]_ext => inspeccionCosechaUvaSupervisor_ext

Temporales (de paso)

Que se usan de paso para un determinado proceso y comunmente se truncan.

[nombreDescripción]_tmp => inspeccionCosechaUvaSupervisor_tmp

Temporales (de sesión)

#tempXXX_[nombreDescripcion] => #temp001_inspeccionCosechaUvaSupervisor

Primary Key

PK_[nombreTabla] => PK_inspeccionCosechaUvaSupervisor

(Se debe especificar el nombre al crearlo, no esta permitido los nombres autogenerados)

Constraints

Los constraint se definirán de la siguiente manera:

Foreign Key

FK _ [nombreTabla]_ [nombreTablaReferencia] _XX => FK_inspeccionCosechaUvaSupervisor_packing_01

Donde:

- *nombreTabla* es el nombre de la **tabla actual o tabla hija**
- *nombreTablaReferenciada* es el nombre de la **tabla referenciada o padre**
- *XX* es el correlativo.

Unique Constraint

UQ_ [nombreTabla]_ [nombreColumna] _XX => UQ_inspeccionCosechaUvaSupervisor_dni_01

Donde:

- *nombreTabla* es el nombre de la **tabla actual**
- *nombreColumnas* es el nombre de la o las columnas separadas por "_"
- *XX* es el correlativo.

(Se debe especificar el nombre al crearlo, no esta permitido los nombres autogenerados)

Check Constraint

CK_ [nombreTabla]_ [regla] _XX => CK_inspeccionCosechaUvaSupervisor_precioMayor100_01

Donde:

- *nombreTabla* es el nombre de la **tabla actual**
- *regla* es la descripción de la **regla de negocio breve y concisa**
- *XX* es el correlativo.

(Se debe especificar el nombre al crearlo, no esta permitido los nombres autogenerados)

Default Constraint

DF_ [nombreTabla]_ [nombreColumna] _XX => DF_inspeccionCosechaUvaSupervisor_estadoID_01

Donde:

- *nombreTabla* es el nombre de la **tabla actual**
- *nombreColumna* es el nombre de la o las columnas separadas por "_"
- *XX* es el correlativo.

(Se debe especificar el nombre al crearlo, no esta permitido los nombres autogenerados)

Índices

Los índices se definirán de la siguiente manera:

IDX_ [nombreTabla]_[nombreColumnas] _XXX =>
IDX_inspeccionCosechaUvaSupervisor_usuarioID_fecha_001

- El *IDX* es el prefijo
- *nombreTabla* es el nombre de la tabla o descripción del índice.
- *nombreColumnas* es el nombre de la o las columnas separadas por "_"
- *XXX* es el correlativo según el número de índices.

Nota: No esta permitido la creación de Índices con nombres autogenerados estos siempre tienen que tener un nombre.

Triggers

Los triggers se definirán de la siguiente manera:

trg_ [NNNN] _ [Tipo de Trigger] _[DML] => trg_inspeccionCosechaUvaSupervisor_AF_IU

Donde:

- *NNNN* es el nombre de la Tabla

- *Tipo de Trigger* es la abreviatura del tipo de disparador que se lanza y puede ser uno de los siguientes:

Tipo	Abreviatura
AFTER	AF
INSTEAD OF	IO

- DML es la abreviatura que se define por el tipo de orden DML que provoca la activación del disparador

Orden DML	Abreviatura
INSERT	I
UPDATE	U
DELETE	D

En el ejemplo trg_inspeccionCosechaUvaSupervisor_AF_IU se indica que corresponde a un trigger de la tabla inspeccionCosechaUvaSupervisor que se ejecutará después de la acción insert o update de la tabla.

Vistas

Las vistas se definirán de la siguiente manera según su naturaleza:

vwc_Calidad_ResumenMensualInspecciones.

vwp_ [MMMM] _[NNNN] => Para vistas primarias, es decir se componen de una sola tabla.

vwc_ [MMMM] _[NNNN] => Para vistas compuestas, es decir se componen de más de una tabla.

Donde:

- *MMMM* se refiere al módulo, área, sistema para el cual fue creada la vista
- *NNNN* en el caso de la vista primaria es el nombre de la tabla que compone la vista y en el caso de la vistas compuestas es el nombre que se le da.

Procedimientos almacenados

Los procedimientos almacenados se definirán de la siguiente manera:
usp _ [PREFIJO] _ [Descripcion] => usp_calidadAgricola_calculoInspeccionCosecha

Donde:

- *PREFIJO* corresponde a la abreviatura definida para el servicio.
- *Descripción* corresponde al nombre del procedimiento según la acción(o servicio) que va a realizar.

Para los procedimientos almacenados de un CRUD de una tabla se realizará de la siguiente manera:

usp_calidadAgricola_inspeccionCosechaUvaSupervisor_ins

usp _ [PREFIJO] _ [nombreTabla] _ins => Para el insert

usp _ [PREFIJO] _ [nombreTabla] _upd => Para el update

usp _ [PREFIJO] _ [nombreTabla] _del => Para el delete (cambio de estado)

usp _ [PREFIJO] _ [nombreTabla] _sel => Para el select de la tabla

Funciones

Las funciones se definirán de la siguiente manera

Funcion con valor Escalar

ufn _ [PREFIJO] _ [Descripcion] => ufn_calidadAgricola_obtenerFecha

Funcion con Valor de Tabla

uft _ [PREFIJO] _ [Descripcion] => uft_calidadAgricola_obtenerResumenRendimiento

Donde:

- *PREFIJO* corresponde a la abreviatura definida para el servicio.
- *Descripción* corresponde al nombre de la función según la acción(o servicio) que va a realizar.

Secuencias

Las secuencias se definirán de la siguiente manera:

- Si la secuencia es usada por alguna tabla.

  seq_ [NNNN] _XX

- Si la secuencia no son usadas por las tablas

  seqG_ [NNNN] _XX

Donde:

- *NNNN* es el nombre de la tabla a la que hace referencia o servicio.
- *XX* es un correlativo.

Types

Type Tabla

uTyp _ [PREFIJO] _ [Descripcion]

Los objetos tipo tabla se definirán de la siguiente manera

- *PREFIJO* corresponde a la abreviatura definida para el servicio.
- *Descripción* corresponde al nombre de la función según la acción(o servicio) que va a realizar.

Estructura de Consultas

1. Las consultas (SELECT) deben ser legibles a la vista para una mejor revisión y lectura de esta, un ejemplo sería de la siguiente manera:

```
SELECT
    c.CustomerName,
    c.Email,
    c.City,
    COUNT(o.OrderID) AS NumeroPedidos
FROM
    Customers c
```

```
    INNER JOIN Ordenes o ON o.clienteID = c.clienteID
WHERE
    c.Country = 'Mexico'
    AND c.IsActive = 1
GROUP BY
    c.CustomerName,
    c.Email,
    c.City
HAVING
    COUNT(o.OrderID) > 5
ORDER BY
    NumeroPedidos DESC;
```

2. Evitar el uso del SELECT * en lo posible, nombrar las columnas que necesitas y/o se utilizan en la consulta, esto mejora la legibilidad, la eficiencia y la robustez de la consulta.

```
-- NO USAR
SELECT c.* FROM Customers c

-- USAR
SELECT
    c.CustomerName,
    c.Email,
    c.City
FROM
    Customers c
```

3. Uso del **WITH (NOLOCK)** cuando el rendimiento es más importante que la precisión de los datos, como por ejemplo en reportes o análisis, evitando que las lecturas bloqueen operaciones de escritura. Recordar que si se requiere precisión de los datos no es recomendable su uso ya que realiza "Lectura sucia".
4. No usar funciones en los campos de filtro de una consulta (WHERE), ya que esto impiden un uso eficiente de los índices.

Diseño de Objetos

Tabla

Tipos de datos

Los tipos de datos deben ser los mas acordes al dato que se va a guardar en la base de datos, usar lo siguiente como base:

- Fechas cortas: date
- Fecha y hora: datetime
- Horas: time
- Texto corto: varchar(n)
- Texto largo: varchar(max)
- Númericos: decimal(18,4)
- Flag: bit

Primary Key

El Primary Key a excepción de otra casuística, debe ser INT con IDENTITY(1,1)

Foreing Key

Todas las tablas deben tener y estar referenciadas con su tabla padre para la integridad referencial respectiva.

Columnas NOT NULL

Las columnas deben tener NOT NULL definido, en caso sea NULL debe estar justificado.

Creación del CHECK Contraint

En las tablas debe estar definido los CHECK CONSTRAINT según la regla de negocio y deben ser nombrados según la nomenclatura.

Creación de Índices

En las tablas debe estar definido los ÍNIDCES según las búsquedas y/o consultas que realizan las operaciones, y deben ser nombrados según la nomenclatura.

Procedimientos Almacenados y Funciones

Parámetros

Los parámetros deben poder identificarse si son de entrada (IN), de salida (OUT) o ambos a la vez (INOUT) se crearán de la siguiente manera:

- **pIn_** para parámetros de entrada
- **pOu_** para parámetros de salida
- **pIO_** para parámetros de entrada y salida

Variables

Las variables deben poder identificarse del mismo modo que los parámetros. Para ello solo usaremos **v** como inicio de la variable. Ejemplo: **v**PackingNuevo

Try Catch

Uso del TRY - CATCH para el manejo de errores y gestionar y mitigar fallos durante la ejecución de código Transact-SQL

Throw

Para la generación de errores personalizados debe usarse THROW

Return

Uso con cuidado del RETURN, usar de preferencia y con mayor control los antes mencionados (Try-Catch y Throw)

Uso de instrucciones SET

Se usarán como parte del procedimiento almacenado las siguientes instrucciones SET:

```
SET NOCOUNT ON; -- Evitar que el servidor envie mensajes de filas afectadas
SET ARITHABORT ON; -- Interrumpe la ejecución por errores aritméticos
SET ANSI_NULLS ON; -- Evaluar las comparaciones de valor NULL según estandar de
SQL
SET XACT_ABORT ON; -- Interrumpe la ejecución si una sentencia T-SQL genera un
error
```

Documentación

Tablas

1. Para la creación de tablas, estas deben tener sus respectivos comentarios tanto a nivel de tabla como de a nivel de columna, esto orientado a tener el diccionario de datos dentro de la misma Base de Datos

```
-- Tablas
EXEC sys.sp_addextendedproperty @name = N'MS_TablaDescription',   @value =
N'[Descripcion de la tabla]',  @level0type = N'SCHEMA', @level0name =
'[schema_name]',  @level1type = N'TABLE',  @level1name = '[table_name]';

-- Columnas de las tablas
EXEC sp_addextendedproperty  @name = N'MS_Col1Desc',  @value = '[Descripcion de
la columna 1]',  @level0type = N'Schema', @level0name = '[schema_name]',
@level1type = N'Table',  @level1name = '[table_name]',  @level2type = N'Column',
@level2name = '[column1_name]';
EXEC sp_addextendedproperty  @name = N'MS_Col2Desc',  @value = '[Descripcion de
la columna 2]',  @level0type = N'Schema', @level0name = '[schema_name]',
@level1type = N'Table',  @level1name = '[table_name]',  @level2type = N'Column',
@level2name = '[column2_name]';
EXEC sp_addextendedproperty  @name = N'MS_Col3Desc',  @value = '[Descripcion de
la columna 3]',  @level0type = N'Schema', @level0name = '[schema_name]',
@level1type = N'Table',  @level1name = '[table_name]',  @level2type = N'Column',
@level2name = '[column3_name]';
```

2. Las tablas deben tener los siguientes columnas y/o campos de auditoría

```
usuarioCreaID INT
fechaCreacion date
usuarioModificaID INT
fechaModificacion date
```

Procedimientos Almacenados, Triggers y Funciones

1. Todos los procedimientos almacenados, triggers y funciones deben contener el siguiente encabezado y en la parte revisiones se debe iterar según los cambios realizados:

```
--
**********************************************************************************
***************
```

```
-- Cliente          : < Nombre del Cliente >
-- Sistema          : < Nombre del sistema >
-- Módulo           : < Nombre del Módulo del Sistema >
-- Autor            : < Nombre del autor / desarrollador >
-- Nombre Objeto    : < Nombre del trigger/funcion/store procedure >
-- Fecha Creacion   : < Fecha Creacion dd/mm/yyyy >
-- ------------------------------------------------------------------------
-----------------
-- Descripcion      : < Describe el propósito del objeto a nivel funcional. Si
es necesario,
-- describe el diseño del objeto a muy alto nivel. >
--
--
-- Input Parameters: < Describe cada uno de los parámetros de entrada del objeto.
>
--
-- Output Parameters: < Describe  cada uno de los parámetros de salida del
objeto. >
--
-- Input/Output Parameters: < Describe cada uno de los parámetros de
entrada/salida del objeto. >
--
-- ------------------------------------------------------------------------
-----------------
-- -- Revisiones
-- ------------------------------------------------------------------------
-----------------
-- -- Modificacion      : < MOD_XXXX >
-- -- Modificado por    : < Nombre de quien modifica >
-- -- Fecha Modificacion: < Fecha de modificacion >
-- -- Motivo Cambio     : < Descripcion del motivo de cambio >
-- ------------------------------------------------------------------------
-----------------
-- -- Modificacion      : < MOD_0002 >
-- -- Modificado por    :
-- -- Fecha Modificacion:
-- -- Motivo Cambio     :
-- ------------------------------------------------------------------------
-----------------
--
************************************************************************
***************
```

2. Las modificaciones realizadas sobre los triggers, los procedimientos almacenados y funciones deben ser identificables, por tanto deben tener la siguiente nomenclatura.

```
-- <INI - MOD_XXXX >
-- Líneas de código
-- ....
-- ..
-- .
-- <FIN - MOD_XXXX >
```

Vistas

1. Todas las vistas deben contener el siguiente encabezado y en la parte revisiones se debe iterar según los cambios realizados:

```
--
********************************************************************************
***************
-- Cliente          : < Nombre del Cliente >
-- Sistema          : < Nombre del sistema >
-- Módulo           : < Nombre del Módulo del Sistema >
-- Autor            : < Nombre del autor / desarrollador >
-- Nombre Objeto    : < Nombre del trigger/funcion/store procedure >
-- Fecha Creacion   : < Fecha Creacion dd/mm/yyyy >
-- ------------------------------------------------------------------------------
-----------------
-- Descripcion      : < Describe el propósito del objeto a nivel funcional. Si
es necesario,
-- describe el diseño del objeto a muy alto nivel. >
--
-- ------------------------------------------------------------------------------
-----------------
-- -- Revisiones
-- ------------------------------------------------------------------------------
-----------------
-- -- Modificacion      : < MOD_XXXX >
-- -- Modificado por    : < Nombre de quien modifica >
-- -- Fecha Modificacion: < Fecha de modificacion >
-- -- Motivo Cambio     : < Descripcion del motivo de cambio >
-- ------------------------------------------------------------------------------
-----------------
--
********************************************************************************
***************
```

2. Las modificaciones realizadas sobre las vistas deben ser identificables, por tanto deben tener la siguiente nomenclatura.

```
-- <INI - MOD_XXXX >
-- Líneas de código
-- ....
-- ..
-- .
-- <FIN - MOD_XXXX >
```

Definición de scripts

1. La extensión de los archivos y scripts para bases de datos debe ser **.sql**
2. Los objetos nuevos solo deben tener la sentencia **CREATE**, en caso que sea modificado recién colocar **CREATE OR REPLACE**.
3. Los nombres de los scripts deben tener la siguiente nomenclatura: enumeración por orden de ejecución, numero de proyecto, nombre del script (sin espacios en blanco) y la extensión.

4. La sintaxis de los archivos de las sentencias múltiples de DML deben terminar en punto y coma ‘;’.

Ejemplo

Tabla y PK

```
CREATE TABLE planta.productoEtiquetadoPlanta (
    ProductoEtiquetadoPlantaID INT IDENTITY(1,1),
    finalProductID         INT NOT NULL,         --
packing.finalproducts.finalProductID
    campaignID             INT NOT NULL,         -- grower.campaign.campaignID
    packingID              INT NOT NULL,         -- packing.packings.packingID
    gtin                   VARCHAR(50) NULL,     -- Código GTIN
    ean13                  VARCHAR(13) NULL,     -- Código EAN13
    upc                    VARCHAR(12) NULL,     -- Código UPC
    code11                 VARCHAR(50) NULL,     -- Código GS1 (renombrado como
code11)
    estado                 BIT NOT NULL DEFAULT 1, -- 0: Anulado, 1: Activo
    usuarioCreaID          INT NOT NULL,
    fechaCreacion          DATETIME NOT NULL DEFAULT GETDATE(),
    usuarioModificaID      INT NULL,
    fechaModificacion      DATETIME NULL
);
-- Primary Key
ALTER TABLE planta.productoEtiquetadoPlanta ADD CONSTRAINT
PK_productoEtiquetadoPlanta PRIMARY KEY (idProductoEtiquetadoPlanta);
GO

-- Descripciones extendidas
-- La tabla
EXEC sp_addextendedproperty @name = N'MS_TablaDescription', @value = N'Tabla
local para configuración de productos etiquetados por planta y campaña',
@level0type = N'SCHEMA', @level0name = 'planta', @level1type = N'TABLE',
@level1name = 'productoEtiquetadoPlanta';

-- Las columnas
EXEC sp_addextendedproperty @name = N'MS_Col1Desc', @value = 'Identificador
único de la tabla', @level0type = N'Schema', @level0name = 'planta',@level1type
= N'Table', @level1name = 'productoEtiquetadoPlanta', @level2type = N'Column',
@level2name = 'ProductoEtiquetadoPlantaID';
EXEC sp_addextendedproperty @name = N'MS_Col2Desc', @value = 'Identificador del
producto final', @level0type = N'Schema', @level0name = 'planta',@level1type =
N'Table', @level1name = 'productoEtiquetadoPlanta', @level2type = N'Column',
@level2name = 'finalProductID';
EXEC sp_addextendedproperty @name = N'MS_Col3Desc', @value = 'Identificador de
la campaña', @level0type = N'Schema', @level0name = 'planta',@level1type =
N'Table', @level1name = 'productoEtiquetadoPlanta', @level2type = N'Column',
@level2name = 'campaignID';
EXEC sp_addextendedproperty @name = N'MS_Col4Desc', @value = 'Identificador del
packing', @level0type = N'Schema', @level0name = 'planta',@level1type =
N'Table', @level1name = 'productoEtiquetadoPlanta', @level2type = N'Column',
@level2name = 'packingID';
```

```
EXEC sp_addextendedproperty @name = N'MS_Col5Desc', @value = 'Código de barras
GTIN', @level0type = N'Schema', @level0name = 'planta',@level1type = N'Table',
@level1name = 'productoEtiquetadoPlanta', @level2type = N'Column', @level2name =
'gtin';
EXEC sp_addextendedproperty @name = N'MS_Col6Desc', @value = 'Código de barras
EAN13', @level0type = N'Schema', @level0name = 'planta',@level1type = N'Table',
@level1name = 'productoEtiquetadoPlanta', @level2type = N'Column', @level2name =
'ean13';
EXEC sp_addextendedproperty @name = N'MS_Col7Desc', @value = 'Código de barras
UPC', @level0type = N'Schema', @level0name = 'planta',@level1type = N'Table',
@level1name = 'productoEtiquetadoPlanta', @level2type = N'Column', @level2name =
'upc';
EXEC sp_addextendedproperty @name = N'MS_Col8Desc', @value = 'Código de barras
GS1 (Code11)', @level0type = N'Schema', @level0name = 'planta',@level1type =
N'Table', @level1name = 'productoEtiquetadoPlanta', @level2type = N'Column',
@level2name = 'code11';
EXEC sp_addextendedproperty @name = N'MS_Col9Desc', @value = 'Estado del
registro (0: Anulado, 1: Activo)', @level0type = N'Schema', @level0name =
'planta',@level1type = N'Table', @level1name = 'productoEtiquetadoPlanta',
@level2type = N'Column', @level2name = 'estado';
EXEC sp_addextendedproperty @name = N'MS_Col10Desc', @value = 'Usuario creador
del registro', @level0type = N'Schema', @level0name = 'planta',@level1type =
N'Table', @level1name = 'productoEtiquetadoPlanta', @level2type = N'Column',
@level2name = 'usuarioCreaID';
EXEC sp_addextendedproperty @name = N'MS_Col11Desc', @value = 'Fecha de creacion
del registro', @level0type = N'Schema', @level0name = 'planta',@level1type =
N'Table', @level1name = 'productoEtiquetadoPlanta', @level2type = N'Column',
@level2name = 'fechaCreacion';
EXEC sp_addextendedproperty @name = N'MS_Col12Desc', @value = 'Usuario que
actualizó por ultima vez', @level0type = N'Schema', @level0name =
'planta',@level1type = N'Table', @level1name = 'productoEtiquetadoPlanta',
@level2type = N'Column', @level2name = 'usuarioModificaID';
EXEC sp_addextendedproperty @name = N'MS_Col13Desc', @value = 'Fecha de ultima
actualizacion', @level0type = N'Schema', @level0name = 'planta',@level1type =
N'Table', @level1name = 'productoEtiquetadoPlanta', @level2type = N'Column',
@level2name = 'fechaModificacion';
GO
```

Constraint

```
ALTER TABLE planta.productoEtiquetadoPlanta ADD CONSTRAINT
CK_productoEtiquetadoPlanta_estadoID_01 CHECK (estado IN (0,1));
```

Foreing Key

```
ALTER TABLE planta.productoEtiquetadoPlanta ADD CONSTRAINT
FK_productoEtiquetadoPlanta_FinalProduct_01
    FOREIGN KEY (finalProductID) REFERENCES
packing.finalproducts(finalProductID);

ALTER TABLE planta.productoEtiquetadoPlanta ADD CONSTRAINT
FK_productoEtiquetadoPlanta_Campaign_02
    FOREIGN KEY (campaignID) REFERENCES grower.campaign(campaignID);
```

Indice

```
CREATE INDEX IDX_productoEtiquetadoPlanta_finalProductID_001
ON planta.productoEtiquetadoPlanta (finalProductID);
```

Procedimiento Almacenado

1. Ejemplo cuando se crea un procedimiento almacenado

```
CREATE PROCEDURE [planta].[usp_productoEtiquetadoPlanta_ins]
(
    @pIn_finalProductID INT,
    @pIn_campaignID     INT,
    @pIn_packingID      INT,
    @pOu_mensajeID      INT OUTPUT,
    @pOu_mensaje        VARCHAR(200) OUTPUT
)
AS
--
*****************************************************************************
***************
-- Cliente          : < Migiva >
-- Sistema          : < Sispacking >
-- Módulo           : < Nombre del Módulo del Sistema >
-- Autor            : < Christian Villaverde >
-- Nombre Objeto    : < planta.sp_productoEtiquetadoPlanta >
-- Fecha Creacion   : < 03/10/2025 >
-- -------------------------------------------------------------------------
-----------------
-- Descripcion      : < SP para registrar en la tabla
planta.productoEtiquetadoPlanta >
--
--
-- Input Parameters: < Describe cada uno de los parámetros de entrada del objeto.
>
-- @pIn_campaignID      Parametro de campaña
-- @pIn_packingID       Parametro de packing
--
-- Output Parameters: < Describe  cada uno de los parámetros de salida del
objeto. >
-- @pOu_mensajeID       Codigo/Id de mensaje retornado al usuario
-- @pOu_mensaje         Mensaje retornado al usuario
--
-- Input/Output Parameters: < Describe cada uno de los parámetros de
entrada/salida del objeto. >
--
--
-- -------------------------------------------------------------------------
-----------------
-- -- Revisiones
-- -------------------------------------------------------------------------
-----------------
-- -- Modificacion      : < MOD_XXXX >
-- -- Modificado por    : < Nombre de quien modifica >
-- -- Fecha Modificacion: < Fecha de modificacion >
-- -- Motivo Cambio     : < Descripcion del motivo de cambio >
```

```
-- ------------------------------------------------------------------------------------
----------------
--
********************************************************************************
***************
BEGIN
    SET NOCOUNT ON;
    SET ARITHABORT ON;
    SET ANSI_NULLS ON;
    SET XACT_ABORT ON;

    BEGIN TRY
        BEGIN TRANSACTION Insertar
        -- Validar que la campaña exista
        IF NOT EXISTS (SELECT 1 FROM grower.campaign WHERE campaignID =
@pIn_campaignID)
        BEGIN
            SET @pOu_mensajeID  = 2;
            SET @pOu_mensaje    = N'❌ La campaña especificada no existe.';
            RETURN;
        END

        -- Validar que existe el packing
        IF NOT EXISTS (SELECT 1 FROM packing.packings WHERE packingID =
@pIn_packingID AND statusID = 1)
        BEGIN
            SET @pOu_mensajeID  = 2;
            SET @pOu_mensaje    = N'❌ El packing especificado no existe.';
            RETURN;
        END

        INSERT INTO planta.productoEtiquetadoPlanta (
            finalProductID,
            campaignID,
            packingID,
            estado,
            usuarioCreaID,
            fechaCreacion,
            usuarioModifcaID,
            fechaModificacion
        ) VALUES (
            @pIn_finalProductID,
            @pIn_campaignID,
            @pIn_packingID,
            1,
            12,
            GETDATE(),
            12,
            GETDATE()
        );
        COMMIT TRAN Insertar
        SET @pOu_mensajeID  = 1;
        SET @pOu_mensaje    = N'✅ Configuración de códigos de barras creada
correctamente.';

    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
```

```
            ROLLBACK TRANSACTION Insertar

        SET @pOu_mensajeID  = 0;
        SET @pOu_mensaje    = N'⛔ Error al insertar: ' + ERROR_MESSAGE();
    END CATCH
END
```

2. Ejemplo cuando se modifica un procedimiento almacenado

```
CREATE OR ALTER PROCEDURE [planta].[usp_productoEtiquetadoPlanta_ins]
(
    @pIn_finalProductID INT,
    @pIn_campaignID     INT,
    @pIn_packingID      INT,
    @pOu_mensajeID      INT OUTPUT,
    @pOu_mensaje        VARCHAR(200) OUTPUT
)
AS
--
*****************************************************************************************
***************
-- Cliente          : < Migiva >
-- Sistema          : < Sispacking >
-- Módulo           : < Nombre del Módulo del Sistema >
-- Autor            : < Christian Villaverde >
-- Nombre Objeto    : < planta.sp_productoEtiquetadoPlanta >
-- Fecha Creacion   : < 03/10/2025 >
-- ---------------------------------------------------------------------------------
-----------------
-- Descripcion      : < SP para registrar en la tabla
planta.productoEtiquetadoPlanta >
--
--
-- Input Parameters: < Describe cada uno de los parámetros de entrada del objeto.
>
-- @pIn_finalProductID  Parametro de finalProduct
-- @pIn_campaignID      Parametro de campaña
-- @pIn_packingID       Parametro de packing
--
-- Output Parameters: < Describe  cada uno de los parámetros de salida del
objeto. >
-- @pOu_mensajeID       Codigo/Id de mensaje retornado al usuario
-- @pOu_mensaje         Mensaje retornado al usuario
--
-- Input/Output Parameters: < Describe cada uno de los parámetros de
entrada/salida del objeto. >
--
--
-- ---------------------------------------------------------------------------------
-----------------
-- -- Revisiones
-- ---------------------------------------------------------------------------------
-----------------
-- -- Modificacion      : < MOD_0001 >
-- -- Modificado por    : < Anders Romero >
-- -- Fecha Modificacion: < 05/10/2025 >
-- -- Motivo Cambio     : < Validar si existen duplicados >
```

```
-- ------------------------------------------------------------------------------
----------------
-- -- Modificacion      : < MOD_0002 >
-- -- Modificado por    : < Christian Villaverde >
-- -- Fecha Modificacion: < 08/10/2025 >
-- -- Motivo Cambio     : < Validar si campaña esta activa >
-- ------------------------------------------------------------------------------
----------------
-- -- Modificacion      : < MOD_0003 >
-- -- Modificado por    : < Christian Villaverde >
-- -- Fecha Modificacion: < 09/10/2025 >
-- -- Motivo Cambio     : < Cambiar RETURN por THROW wn la validacion packing >
-- ------------------------------------------------------------------------------
----------------
--
********************************************************************************
***************
BEGIN
    SET NOCOUNT ON;
    SET ARITHABORT ON;
    SET ANSI_NULLS ON;
    SET XACT_ABORT ON;

    BEGIN TRY
        BEGIN TRANSACTION Insertar
        -- Validar que la campaña exista
        -- <INI - MOD_0002 >
        -- IF NOT EXISTS (SELECT 1 FROM grower.campaign WHERE campaignID =
@pIn_campaignID)
        IF NOT EXISTS (SELECT 1 FROM grower.campaign WHERE campaignID =
@pIn_campaignID AND statusID = 1)
        -- <FIN - MOD_0002 >
        BEGIN
            SET @pOu_mensajeID  = 2;
            SET @pOu_mensaje    = N'❌ La campaña especificada no existe.';
            RETURN;
        END

        -- Validar que existe el packing
        IF NOT EXISTS (SELECT 1 FROM packing.packings WHERE packingID =
@pIn_packingID AND statusID = 1)
        BEGIN
            SET @pOu_mensajeID  = 2;
            SET @pOu_mensaje    = N'❌ El packing especificado no existe.';
            -- <INI - MOD_0003 >
            -- RETURN;
            THROW @pOu_mensajeID, @pOu_mensaje, 1;
            -- <FIN - MOD_0003 >
        END

        -- <INI - MOD_0001 >
        -- Validar duplicados activos
        IF EXISTS (
            SELECT 1 FROM planta.productoEtiquetadoPlanta
            WHERE finalProductID = @pIn_finalProductID
              AND campaignID = @pIn_campaignID
              AND packingID = @pIn_packingID
              AND estado = 1
```

```
        )
        BEGIN
            SET @pOu_mensajeID  = 2;
            SET @pOu_mensaje    = N'❌ Ya existe una configuración activa para
este producto en esta campaña y packing.';
            RETURN;
        END
        -- <FIN - MOD_0001 >

        INSERT INTO planta.productoEtiquetadoPlanta (
            finalProductID,
            campaignID,
            packingID,
            estado,
            usuarioCreaID,
            fechaCreacion,
            usuarioModifcaID,
            fechaModificacion
        ) VALUES (
            @pIn_finalProductID,
            @pIn_campaignID,
            @pIn_packingID,
            1,
            12,
            GETDATE(),
            12,
            GETDATE()
        );
        COMMIT TRAN Insertar

        SET @pOu_mensajeID  = 1;
        SET @pOu_mensaje    = N'✅ Configuración de códigos de barras creada
correctamente.';

    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION Insertar

        SET @pOu_mensajeID  = 0;
        SET @pOu_mensaje    = N'🚫 Error al insertar: ' + ERROR_MESSAGE();
        -- <INI - MOD_0003 >
        THROW @pOu_mensajeID, @pOu_mensaje, ERROR_STATE();
        -- <FIN - MOD_0003 >
    END CATCH
END
```

Funcion

Funcion Escalar

```
CREATE FUNCTION [agroamigo].[ufn_sistemas_obtenerFecha]()
RETURNS DATETIME
AS
--
**********************************************************************************
***************
```

```
-- Cliente          : < Migiva >
-- Sistema          : < Todos los sitemas >
-- Módulo           : < Todos los módulos >
-- Autor            : < Christian Villaverde >
-- Nombre Objeto    : < agroamigo.ufn_sistemas_obtenerFecha >
-- Fecha Creacion   : < 03/10/2025 >
-- ----------------------------------------------------------------------------
-----------------
-- Descripcion      : < Función que devuelve la fecha en el formato UTC - 5 >
--
--
-- Input Parameters: < No presenta. >
--
-- Output Parameters: < No presenta. >
--
-- Input/Output Parameters: < No presenta. >
--
--
-- ----------------------------------------------------------------------------
-----------------
-- -- Revisiones
-- ----------------------------------------------------------------------------
-----------------
-- -- Modificacion      : < MOD_XXXX >
-- -- Modificado por    : < Nombre de quien modifica >
-- -- Fecha Modificacion: < Fecha de modificacion >
-- -- Motivo Cambio     : < Descripcion del motivo de cambio >
-- ----------------------------------------------------------------------------
-----------------
--
****************************************************************************
***************
BEGIN
    DECLARE @vFecha AS DATETIMEOFFSET
    SET @vFecha = CONVERT(DATETIMEOFFSET, GETDATE()) AT TIME ZONE 'SA Pacific
Standard Time'
    RETURN CONVERT(DATETIME, @vFecha);
END
GO
```

Funcion Tabla

```
CREATE FUNCTION [planta].[uft_obtenerOnpremID](
    @pIn_onpremID INT
)
RETURNS TABLE
AS
--
****************************************************************************
***************
-- Cliente          : < Migiva >
-- Sistema          : < Todos los sitemas >
-- Módulo           : < Todos los módulos >
-- Autor            : < Christian Villaverde >
-- Nombre Objeto    : < planta.uft_obtenerOnpremID >
-- Fecha Creacion   : < 03/10/2025 >
```

```
-- ------------------------------------------------------------------------------
-----------------
-- Descripcion      : < Función que devuelve el packing a quien pertenece el
on_premID >
--
--
-- Input Parameters: <  >
-- @pIn_onpremID    Parámetro de Id del  on_premID a visualizar
--
-- Output Parameters: < No presenta. >
--
-- Input/Output Parameters: < No presenta. >
--
--
-- ------------------------------------------------------------------------------
-----------------
-- -- Revisiones
-- ------------------------------------------------------------------------------
-----------------
-- -- Modificacion      : < MOD_XXXX >
-- -- Modificado por    : < Nombre de quien modifica >
-- -- Fecha Modificacion: < Fecha de modificacion >
-- -- Motivo Cambio     : < Descripcion del motivo de cambio >
-- ------------------------------------------------------------------------------
-----------------
--
********************************************************************************
***************
RETURN
    SELECT
        on_premID,
        CASE
            WHEN on_premID = 1 THEN 'CAL'
            WHEN on_premID = 2 THEN 'NAT'
            WHEN on_premID = 3 THEN 'CAR'
            WHEN on_premID = 4 THEN 'BMP'
        ELSE
            ''
        END AS farmID
    FROM
        packing.on_prem
    WHERE
        on_premID = @pIn_onpremID
GO
```

Vista

```
CREATE VIEW [Maestra].[vwc_Maestra_obtenerDatosPersona]
as
--
********************************************************************************
***************
-- Cliente          : < Migiva >
-- Sistema          : < Todos los sitemas >
-- Módulo           : < Todos los módulos >
-- Autor            : < Christian Villaverde >
```

```
-- Nombre Objeto    : < Maestra.vwc_Maestra_obtenerDatosPersona >
-- Fecha Creacion   : < 03/10/2025 >
-- ------------------------------------------------------------------------------
-----------------
-- Descripcion      : < Vista para devolver el listado de personas >
--
--
-- ------------------------------------------------------------------------------
-----------------
-- -- Revisiones
-- ------------------------------------------------------------------------------
-----------------
-- -- Modificacion      : < MOD_XXXX >
-- -- Modificado por    : < Nombre de quien modifica >
-- -- Fecha Modificacion: < Fecha de modificacion >
-- -- Motivo Cambio     : < Descripcion del motivo de cambio >
-- ------------------------------------------------------------------------------
-----------------
--
**************************************************************************************
***************
select
    p.Id,LTRIM(RTRIM(CONCAT(P.ApellidoPaterno,' ',P.ApellidoMaterno,'
',p.Nombres))) Nombre,
    t.Abreviatura TipoDocumento,
    p.NumeroDocumento,
    p.Direccion,
    p.Correo,
    p.Abreviatura,
    p.growerId
from
    [Maestra].[PersonaEntidad]  p
    inner join [Maestra].[TipoDocumento] t on t.Id = p.IdTipoDocumento
where
    p.Estado=1
GO
```

Base de Datos de Proveedores

Estas Bases de Datos no están sujetas completamente a los estándares de codificación, pero deben cumplir lo mínimo requerido por TI para su entrega en Producción. (Ver Documento Asociado ***"Entrega de Base de Datos a Producción"***)

Así mismo se debe tener en cuenta lo siguiente:

- Los nuevos objetos creados (desarrollo interno) que explotan la data de en una Base de Datos de proveedores deben ir en un esquema nuevo cumpliendo los estándares.
- No se debe contar con el rol DBA en los esquemas.
- Contar con un usuarios de conexión para no acceder directamente al core de la Base de Datos.

Buenas Prácticas de Base de Datos T-SQL

Consultas T-SQL

1. Evitar usar `SELECT *`

   Siempre especifica las columnas que realmente necesitas en lugar de usar `SELECT *`, para reducir E/S, mejorar rendimiento y evitar problemas ante cambios de esquema.

   Mal Ejemplo ❌

   ```
   SELECT * FROM Sales.Orders;
   ```

   Buen Ejemplo ✅

   ```
   SELECT OrderID, CustomerID, OrderDate, TotalAmount
   FROM Sales.Orders;
   ```

2. Evita funciones en columnas dentro de filtros `WHERE`

   No utilices funciones sobre columnas en condiciones de búsqueda, ya que invalidan el uso de índices y degradan el rendimiento.

   Mal Ejemplo ❌

   ```
   SELECT * FROM Sales.Orders
   WHERE YEAR(OrderDate) = 2025;
   ```

   Buen Ejemplo ✅

   ```
   SELECT * FROM Sales.Orders
   WHERE OrderDate >= '2025-01-01' AND OrderDate < '2026-01-01';
   ```

3. Usa `JOIN` explícitos en lugar de `WHERE` para unir tablas

   Los `JOIN` explícitos son más legibles, evitan errores lógicos y son el estándar moderno de SQL.

   Mal Ejemplo ❌

   ```
   SELECT o.OrderID, c.CustomerName
   FROM Sales.Orders o, Sales.Customers c
   WHERE o.CustomerID = c.CustomerID;
   ```

   Buen Ejemplo ✅

   ```
   SELECT o.OrderID, c.CustomerName
   FROM Sales.Orders AS o
   INNER JOIN Sales.Customers AS c ON o.CustomerID = c.CustomerID;
   ```

4. Evita cursores cuando puedas usar operaciones en conjunto

   Los cursores son costosos. Prefiere operaciones basadas en conjuntos (`UPDATE`, `INSERT`, `MERGE`, `CTE`, etc.).

Mal Ejemplo ❌

```
DECLARE order_cursor CURSOR FOR
SELECT OrderID FROM Sales.Orders;

OPEN order_cursor;
FETCH NEXT FROM order_cursor INTO @OrderID;

WHILE @@FETCH_STATUS = 0
BEGIN
    UPDATE Sales.Orders SET Status = 'Closed' WHERE OrderID = @OrderID;
    FETCH NEXT FROM order_cursor INTO @OrderID;
END

CLOSE order_cursor;
DEALLOCATE order_cursor;
```

Buen Ejemplo ✅

```
UPDATE Sales.Orders
SET Status = 'Closed'
WHERE Status = 'Open';
```

5. Utiliza `EXISTS` en lugar de `COUNT(*)` para validaciones

   Cuando solo se necesita verificar si existen registros, `EXISTS` es más eficiente que `COUNT(*)`.

   Mal Ejemplo ❌

```
IF (SELECT COUNT(*) FROM Sales.Orders WHERE CustomerID = @CustomerID) > 0
    PRINT 'Tiene pedidos';
```

   Buen Ejemplo ✅

```
IF EXISTS (SELECT 1 FROM Sales.Orders WHERE CustomerID = @CustomerID)
    PRINT 'Tiene pedidos';
```

6. Declara variables con tipos de datos apropiados

   Usar tipos de datos incorrectos genera conversiones implícitas y afecta el rendimiento.

   Mal Ejemplo ❌

```
DECLARE @CustomerID NVARCHAR(50) = '100';
SELECT * FROM Sales.Orders WHERE CustomerID = @CustomerID;
```

   Buen Ejemplo ✅

```
DECLARE @CustomerID INT = 100;
SELECT * FROM Sales.Orders WHERE CustomerID = @CustomerID;
```

7. Evita usar `DISTINCT` para corregir duplicados

   Usa `DISTINCT` solo cuando sea realmente necesario. Analiza y corrige la causa de los duplicados desde el origen o la unión.

   Mal Ejemplo ❌

   ```
   SELECT DISTINCT c.CustomerName, o.OrderID
   FROM Sales.Customers c
   JOIN Sales.Orders o ON c.CustomerID = o.CustomerID;
   ```

   Buen Ejemplo ✅

   ```
   SELECT c.CustomerName, o.OrderID
   FROM Sales.Customers c
   JOIN Sales.Orders o ON c.CustomerID = o.CustomerID;
   -- Asegurar integridad y claves únicas correctamente definidas
   ```

8. Evita subconsultas anidadas cuando puedes usar `JOIN` o `APPLY`

   Las subconsultas correlacionadas suelen ser más lentas. Prefiere `JOIN` o `CROSS APPLY` para optimizar.

   Mal Ejemplo ❌

   ```
   SELECT c.CustomerName,
          (SELECT COUNT(*) FROM Sales.Orders o WHERE o.CustomerID =
   c.CustomerID) AS TotalOrders
   FROM Sales.Customers c;
   ```

   Buen Ejemplo ✅

   ```
   SELECT c.CustomerName, COUNT(o.OrderID) AS TotalOrders
   FROM Sales.Customers c
   LEFT JOIN Sales.Orders o ON o.CustomerID = c.CustomerID
   GROUP BY c.CustomerName;
   ```

Uso de Índices

1. Crear índices solo donde aporten valor

   Crea índices basados en las consultas más frecuentes y costosas (lecturas, joins, filtros o agrupaciones).
   Evita crear índices innecesarios, ya que cada índice adicional incrementa el costo de escritura (INSERT, UPDATE, DELETE).

   Mal Ejemplo ❌

   ```
   -- Índices redundantes sin análisis previo
   CREATE INDEX IDX_Orders_OrderDate_001 ON Sales.Orders(OrderDate);
   CREATE INDEX IDX_Orders_CustomerID_002 ON Sales.Orders(CustomerID);
   CREATE INDEX IDX_Orders_CustomerID_OrderDate_003 ON Sales.Orders(CustomerID,
   OrderDate);
   ```

Buen Ejemplo ✅

```
-- Índice creado según patrón de consulta real
CREATE INDEX IDX_Orders_CustomerID_OrderDate_001
ON Sales.Orders (CustomerID, OrderDate)
INCLUDE (TotalAmount);
```

2. Colocar las columnas más selectivas primero

   En un índice compuesto, el orden de las columnas es crítico. Coloca primero la columna **más selectiva** (la que más reduce el número de filas).

   Mal Ejemplo ❌

   ```
   CREATE INDEX IDX_Orders_Status_CustomerID_001 ON Sales.Orders(Status,
   CustomerID);
   ```

   Buen Ejemplo ✅

   ```
   CREATE INDEX IDX_Orders_CustomerID_Status_001 ON Sales.Orders(CustomerID,
   Status);
   ```

3. Usa índices cubrientes (Covering Index)

   Un índice cubriente incluye todas las columnas que una consulta necesita, evitando lecturas adicionales de la tabla (bookmark lookup).

   - Mejora drásticamente el rendimiento de SELECT repetitivos.
   - Reduce el número de lecturas lógicas.

   Buen Ejemplo ✅

   ```
   CREATE INDEX IDX_Orders_CustomerID_OrderDate_001
   ON Sales.Orders (CustomerID)
   INCLUDE (OrderDate, TotalAmount);
   ```

4. Evita índices en columnas con alta volatilidad

   Columnas que cambian con frecuencia degradan el rendimiento de operaciones DML. Evita indexar campos como “LastUpdatedDate”, “Status”, o “ModifiedBy” si cambian constantemente.

   Mal Ejemplo ❌

   ```
   CREATE INDEX IDX_Orders_LastUpdated_001 ON Sales.Orders(LastUpdatedDate);
   ```

   Buen Ejemplo ✅

   ```
   -- Mejor usar índice sobre columnas más estables
   CREATE INDEX IDX_Orders_CustomerID_OrderDate_001 ON Sales.Orders(CustomerID,
   OrderDate);
   ```

5. Usa índices únicos donde corresponda

Un índice `UNIQUE` no solo mejora el rendimiento de búsqueda, sino que garantiza integridad lógica de los datos (Evita duplicidad de datos y reduce necesidad de validaciones adicionales en la aplicación.)

Ejemplo ✅

```
CREATE UNIQUE INDEX UK_Customers_Email_001 ON Sales.Customers(Email);
```

6. Usa índices filtrados (`Filtered Index`) cuando aplique

   Los índices filtrados son ideales para tablas grandes con pocos valores relevantes (ej: registros activos).

Ejemplo ✅

```
CREATE INDEX IDX_Orders_ActiveOrders_001
ON Sales.Orders(OrderDate)
WHERE Status = 'Open';
```

8. Evita duplicar índices (idénticos o parcialmente iguales)

   SQL Server permite crear índices redundantes, lo que consume espacio y degrada el rendimiento en escritura.

   Mal Ejemplo ❌

```
CREATE INDEX IDX_Orders_CustomerID_001 ON Sales.Orders(CustomerID);
CREATE INDEX IDX_Orders_CustomerID_OrderDate_002 ON Sales.Orders(CustomerID,
OrderDate);
```

   Buen Ejemplo ✅

```
-- Deja solo el índice más útil
DROP INDEX IDX_Orders_CustomerID_001 ON Sales.Orders;
```

9. Evita funciones sobre columnas indexadas en `WHERE` o `JOIN`

   El uso de funciones anula el índice, forzando una lectura completa (table scan).

   Mal Ejemplo ❌

```
SELECT * FROM Sales.Orders
WHERE YEAR(OrderDate) = 2025;
```

   Buen Ejemplo ✅

```
SELECT * FROM Sales.Orders
WHERE OrderDate >= '2025-01-01' AND OrderDate < '2026-01-01';
```

10. Mantén consistencia entre índices y claves foráneas

    Siempre indexa las columnas utilizadas en relaciones (`FOREIGN KEY`), especialmente en tablas hijas (Optimiza joins y validaciones referenciales).

    Mal Ejemplo ❌

```
ALTER TABLE Sales.Orders
ADD CONSTRAINT FK_Orders_Customers FOREIGN KEY (CustomerID)
REFERENCES Sales.Customers(CustomerID);
-- Sin índice en CustomerID
```

Buen Ejemplo ✅

```
CREATE INDEX IDX_Orders_CustomerID_001 ON Sales.Orders(CustomerID);
```

Uso de Tablas Temporales

1. Usa tablas temporales solo cuando sea necesario

   Las tablas temporales son útiles para almacenar resultados intermedios, pero si puedes resolver el problema con una **CTE**, **subconsulta** o **table variable simple**, evita crearlas.

   Ten en cuenta: Crea una tabla temporal solo si se reutiliza varias veces o reduce complejidad de joins.

   Mal Ejemplo ❌

   ```
   SELECT * INTO #TempOrders FROM Sales.Orders;
   SELECT * FROM #TempOrders WHERE OrderDate >= '2025-01-01';
   ```

   Buen Ejemplo ✅

   ```
   WITH Orders2025 AS (
       SELECT OrderID, CustomerID, OrderDate
       FROM Sales.Orders
       WHERE OrderDate >= '2025-01-01'
   )
   SELECT OrderID, CustomerID, OrderDate FROM Orders2025;
   ```

2. Usa `CREATE TABLE` en lugar de `SELECT INTO` para control y rendimiento

   `SELECT INTO` bloquea la compilación y crea metadatos en `tempdb` sin control.
   Usar `CREATE TABLE` permite definir tipos, índices y control total. Algunas ventajas:

   - Control de tipos de datos
   - Posibilidad de crear índices
   - Evita recompilaciones innecesarias

   Mal Ejemplo ❌

   ```
   SELECT * INTO #TempOrders FROM Sales.Orders WHERE OrderDate >= '2025-01-01';
   ```

   Buen Ejemplo ✅

```sql
CREATE TABLE #TempOrders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    TotalAmount DECIMAL(12,2)
);

INSERT INTO #TempOrders (OrderID, CustomerID, OrderDate, TotalAmount)
SELECT OrderID, CustomerID, OrderDate, TotalAmount
FROM Sales.Orders
WHERE OrderDate >= '2025-01-01';
```

3. Crea índices en tablas temporales grandes o reutilizadas

   Si la tabla temporal tiene muchos registros o será usada en múltiples joins, podrías crear índices para mejorar el rendimiento.

   Mal Ejemplo ❌

```sql
CREATE TABLE #TempOrders (
    OrderID INT NOT NULL,
    CustomerID INT NOT NULL,
    OrderDate DATE
);
```

   Buen Ejemplo ✅

```sql
CREATE TABLE #TempOrders (
    OrderID INT NOT NULL,
    CustomerID INT NOT NULL,
    OrderDate DATE,
    INDEX IDX_TempOrders_Customer_001 ID NONCLUSTERED (CustomerID)
);
```

4. Limpia las tablas temporales al finalizar

   Aunque las tablas temporales se eliminan al cerrar la sesión, es buena práctica liberarlas explícitamente para evitar presión sobre `tempdb`.

   Buen Ejemplo ✅

```sql
CREATE TABLE #TempOrders (...);
-- ... operaciones ...
DROP TABLE #TempOrders;
```

5. Evita crear/dropear temporales en loops o cursores

   Cada ciclo crea/dropea objetos en `tempdb`, generando fragmentación de metadatos.

   Mal Ejemplo ❌

```sql
WHILE (@i < 1000)
BEGIN
    CREATE TABLE #TempLoop (ID INT);
    DROP TABLE #TempLoop;
    SET @i += 1;
END
```

Buen Ejemplo ✅

```
CREATE TABLE #TempLoop (ID INT);
WHILE (@i < 1000)
BEGIN
    INSERT INTO #TempLoop VALUES (@i);
    SET @i += 1;
END
DROP TABLE #TempLoop;
```

6. Usa `##GlobalTemp` solo si es estrictamente necesario

   Las tablas temporales globales (`##`) son visibles por todas las sesiones. Evita su uso salvo en procesos coordinados entre sesiones. Recuerda:

   - Evita `##Temp` en entornos multiusuario o producción.
   - Si debes usarlas, agrega control de nombres (por usuario o proceso).

   Mal Ejemplo ❌

   ```
   CREATE TABLE ##TempReport (...);
   ```

   Buen Ejemplo ✅

   ```
   CREATE TABLE #TempReport (...);
   ```

Uso de CTE

Una **CTE** es una expresión temporal que existe **solo durante la ejecución de una instrucción SQL**. Se define con `WITH NombreCTE AS (...)` y se comporta como una vista temporal.

1. No uses CTE para grandes conjuntos intermedios

   La CTE **no materializa datos físicamente** (no se guarda en memoria o disco), así que si se reusa dentro de la misma consulta, el optimizador puede **recalcularla varias veces**. Recuerda:

   - Si el mismo resultado se usa muchas veces → Usa una tabla temporal.
   - Si solo se usa una vez → CTE.

2. Usa CTE solo para una ejecución

   Una CTE **no se puede reutilizar** dentro de la misma sesión o bloque. Si vas a usar el mismo conjunto de datos varias veces, usa una tabla temporal (`#Temp`) o variable (`@Table`).

3. Usa CTE para mejorar la legibilidad del código

   Las CTE son ideales para **dividir una consulta compleja en pasos lógicos**.

   - Código más claro

- Ideal para documentar pasos intermedios
  - Mantenimiento más fácil

  Mal Ejemplo ❌

```sql
SELECT c.CustomerID, c.Name, SUM(o.Total) AS TotalSales
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN (
 SELECT CustomerID, COUNT(*) AS NumOrders
    FROM Orders
    GROUP BY CustomerID
) x ON c.CustomerID = x.CustomerID
WHERE x.NumOrders > 10
GROUP BY c.CustomerID, c.Name;
```

Buen Ejemplo ✅

```sql
WITH OrderCount AS (
    SELECT CustomerID, COUNT(*) AS NumOrders
    FROM Orders
    GROUP BY CustomerID
)
SELECT c.CustomerID, c.Name, SUM(o.Total) AS TotalSales
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN OrderCount oc ON c.CustomerID = oc.CustomerID
WHERE oc.NumOrders > 10
GROUP BY c.CustomerID, c.Name;
```

4. No abuses de CTE anidados o recursivos

   Cada CTE genera una *recompilación* lógica. Demasiados anidados pueden degradar el rendimiento.

   Mal Ejemplo ❌

```sql
WITH CTE1 AS (...),
     CTE2 AS (SELECT ... FROM CTE1),
     CTE3 AS (SELECT ... FROM CTE2),
     CTE4 AS (SELECT ... FROM CTE3)
SELECT ... FROM CTE4;
```

   Buen Ejemplo ✅

```sql
WITH CTE AS (
    SELECT ... -- combina la lógica necesaria
)
SELECT ... FROM CTE;
```

Uso de Variables Tabla(@table)

Una variable de tabla es una estructura temporal declarada con `DECLARE @Nombre TABLE (...)`. Se almacena principalmente en memoria (aunque puede usar `tempdb` si crece mucho).

1. Úsala solo para conjuntos pequeños

Las variables de tabla **no generan estadísticas completas**. Por eso el optimizador asume que solo tienen **1 fila**, lo que puede causar *malos planes de ejecución*. Tener en cuenta o como consejo si tiene mas de 1000 filas usemos #Temp

Mal Ejemplo ❌

```
DECLARE @Orders TABLE (OrderID INT, Total DECIMAL(10,2));
INSERT INTO @Orders SELECT OrderID, Total FROM Orders;  -- miles de filas
```

Buen Ejemplo ✅

```
DECLARE @Orders TABLE (OrderID INT, Total DECIMAL(10,2));
INSERT INTO @Orders SELECT TOP (500) OrderID, Total FROM Orders; --
Acortamos los resultados
```

2. Usa variables de tabla en funciones o procedimientos pequeños

   Las variables de tabla son **perfectas para devolver conjuntos** en funciones o manejar resultados de pequeña escala.

   Ejemplo ✅

```
CREATE FUNCTION dbo.uft_GetTopCustomers()
RETURNS @Result TABLE (
    CustomerID INT,
    Name NVARCHAR(100),
    TotalSales DECIMAL(12,2)
)
AS
BEGIN
    INSERT INTO @Result
    SELECT TOP (10) CustomerID, Name, SUM(Total)
    FROM Sales
    GROUP BY CustomerID, Name
    ORDER BY SUM(Total) DESC;
    RETURN;
END;
```

3. No uses @Table en operaciones masivas ni cursores

   Si vas a realizar actualizaciones o joins sobre millones de registros, las variables de tabla degradan rendimiento (sin paralelismo ni estadísticas).

Diferencia CTE, @Table y #Temp

Característica	CTE	@Table	#Temp
Persistencia	Solo en la instrucción	En la sesión (memoria)	En la sesión (`tempdb`)
Se almacena físicamente	❌ No	⚠️ Parcialmente	✅ Sí
Reutilizable	❌ No	✅ Sí	✅ Sí
Permite índices	❌ No	✅ Limitado	✅ Completo
Genera estadísticas	❌ No	❌ (hasta SQL 2019)	✅ Sí
Ideal para	Simplificar consultas	Conjuntos pequeños	Conjuntos grandes o reutilizados
Recomendado en	Lecturas únicas	Procedimientos pequeños	Procedimientos complejos