DIVISIÓN DE EDUCACIÓN CONTINUA DE LA FACULTAD DE INGENIERÍA, UNAM
EVALUACIÓN DEL PERSONAL DOCENTE

Curso: CC41 EL DISEÑO DE BASES DE DATOS

Instructor: ING. RODOLFO GONZALEZ MALDONADO

Duración: 20 Hrs.

El propósito de este cuestionario es conocer su opinión acerca del desarrollo. Marque con una "X" considerando el siguiente puntaje. Es muy importante contar con su objetividad.

10=Excelente 9=Muy bueno 8=Bueno 7=Suficiente 6=Malo 5=Deficiente

EVALUACIÓN DEL INSTRUCTOR

Factores a evaluar	10	9	8	7	6	5
1. Mostró dominio del tema						
2. Utilizo un lenguaje claro y sencillo						
3. Propicio la integración del grupo con el propósito de alcanzar el objetivo del curso						
4. Despertó y mantuvo el interés de los participantes						
5. El instructor supervisó adecuadamente los trabajos						
6. Resolvió oportunamente las dudas y los problemas de los participantes						
7. Manejo correctamente los apoyos y recursos didácticos durante su intervención						
8. Ante situaciones conflictivas presentadas por el grupo el instructor fue profesional en su actuación						
9. Ilustro los temas con casos prácticos						
10. Inició y concluyó puntualmente y empleó adecuadamente el tiempo destinado para su exposición						

Comentarios y sugerencias: _____

Factores a evaluar	10	9	8	7	6	5
1. El temario del curso cumplió sus expectativas						
2. El conocimiento adquirido es aplicable en las funciones que desempeña						
3. Los temas tuvieron una secuencia lógica						
4. Las instalaciones fueron adecuadas y cómodas						
5. La coordinación del curso fue adecuada						
6. La duración del curso fue suficiente						
7. Los ejercicios y la dinámica fueron acordes con el contenido del curso						
8. Los temas acordes tuvieron un equilibrio teórico práctico						
9. Los materiales de apoyo y manuales empleados fueron suficientes y de calidad						

Comentarios y sugerencias: _____

HERRAMIENTAS TECNOLOGICAS

Diseño de Base de Datos

Objetivo

El participante conocera y aprendera a desarrollar un sistema de consultas y de base de datos, asi como la forma de compartir esta informacion a traves de sistemas de conectividad a fin de poder realizar reportes y consolidados que le permitan el analisis cuantitativo de la informacion.

Justificación

Los sistemas actuales de información, son actualmente las herramientas que dan respuesta a las necesidades de la administración y ayudan en los procesos de toma de decisiones del Intitulo. Asi, este curso, cubre el aspecto tecnologico para la implementacion de un sistema de informacion, tanto a nivel conceptual, como en sus requerimientos de Software y Hardware.

Programa

I Bases de datos-Modelo de datos relacional
- 1 1Introducción
- 1.2Conceptos
- 1 3 Modelo de Entidades y Asociaciones
- 1 4 Proceso de normalizacion
- 1 5 El SQL
- I 6 Implementación del Sistema

Índice

I.- Bases de datos-Modelo de datos relacional-

1.1 Introducción

Las bases de datos relacionales son el tipo de bases de datos actualmente más difundido. Los motivos de este éxito son fundamentalmente dos:

1. ofrecen sistemas simples y eficaces para representar y manipular los datos
2. se basan en un modelo, el relacional, con sólidas bases teoricas

El modelo relacional fue propuesto originariamente por E.F. Codd en un ya famoso artículo de 1970. Gracias a su coherencia y facilidad de uso, el modelo se ha convertido en los años 80 en el más usado para la producción de DBMS.

1.2 Conceptos

La estructura fundamental del modelo relacional es precisamente esa, "relación", es decir una tabla bidimensional constituida por lineas (tuple) y columnas (atributos). Las relaciones representan las entidades que se consideran interesantes en la base de datos. Cada instancia de la entidad encontrará sitio en una tupla de la relación, mientras que los atributos de la relación representarán las propiedades de la entidad. Por ejemplo, si en la base de datos se tienen que representar personas, se podrá definir una relación llamada "Personas", cuyos atributos describen las características de las personas (tabla siguiente). Cada tupla de la relación "Personas" representará una persona concreta.

Persona

Nombre	Apellido	Nacimiento	Sexo	Estado Civil
Juan	Loza	15/06/1971	H	Soltero
Isabel	Galvez	23/12/1969	M	Casada
Micaela	Ruiz	02/10/1985	M ,	Soltera

En realidad, siendo rigurosos, una relación es sólo la definición de la estructura de la tabla, es decir su nombre y la lista de los atributos que la componen. Cuando se puebla con las tuplas, se habla de "instancia de relación". Por eso, la tabla anterior representa una instancia de la relación persona. Una representación de la definiticón de esa relacion podría ser la siguiente:

Personas (nombre, apellido, fecha_nacimiento, sexo, estado _civil)

A continuación, se indicarán ambas (relación e instancia de relacion) con el término "relación", a no ser que no quede claro por el contexto a qué acepción se refiere.

Las tuplas en una relación son un conjunto en el sentido matemático del término, es decir una colección no ordenada de elementos diferentes. Para distinguir una tupla de otra, se recurre al concepto de "llave primaria", o sea a un conjunto de atributos que permiten identificar unívocamente una tupla en una relación. Naturalmente, en una relacion puede haber más combinaciones de atributos que permitan identificar unívocamente una tupla ("llaves candidatas"), pero entre estas se elegirá una sola para utilizar como llave primaria. Los atributos de la llave primaria no pueden asumir el valor nulo (que significa un valor no determinado), en tanto que ya no permitirían identificar una tupla concreta en una relación. Esta propiedad de las relaciones y de sus llaves primarias está bajo el nombre de integridad de las entidades (entity integrity).

A menudo, para obtener una llave primaria "económica", es decir compuesta de pocos atributos fácilmente manipulables, se introducen uno o más atributos ficticios, con códigos identificativos unívocos para cada tupla de la relación.

Cada atributo de una relación se caracteriza por un nombre y por un dominio. El dominio indica qué valores pueden ser asumidos por una columna de la relación. A menudo un dominio se define a traves de la declaración de un tipo para el atributo (por ejemplo diciendo que es una cadena de diez caracteres), pero también es posible definir dominios más complejos y precisos. Por ejemplo, para el atributo "sexo" de nuestra relacion "Personas" podemos definir un dominio por el cual los únicos valores válidos son 'M' y 'F'; o bien por el atributo "fecha_nacimiento" podremos definir un dominio por el que se consideren válidas sólo las fechas de nacimiento despues del uno de enero de 1960, si en nuestra base de datos no está previsto que haya personas con fecha de nacimiento anterior a esa. El motor de datos se ocupará de controlar que en los atributos de las relaciones se incluyan sólo los valores permitidos por sus dominios. Característica fundamental de los dominios de una base de datos relacional es que sean "atómicos", es decir que los valores contenidos en las columnas no se puedan separar en valores de dominios más simples. Más formalmente se dice que no es posible tener atributos multivalor (multivalued). Por ejemplo, si una característica de las personas en nuestra base de datos fuese la de tener uno o mas hijos, no sería posible escribir la relacion Personas de la siguiente manera:

> Personas (nombre, apellido, fecha_nacimiento, sexo, estado_civil, hijos)

En efecto, el atributo hijos es un atributo no-atómico, bien porque una persona puede tener más de un hijo o porque cada hijo tendrá

diferentes características que lo describen. Para representar estas entidades en una base de datos relacional hay que definir dos relaciones:

Personas (*número_persona, nombre, apellido, fecha_nacimiento, sexo, estado_civil)
Hijos(*número_persona, *nombre_apellido, edad, sexo)

En las relaciones precedentes, los asteriscos (*) indican los atributos que componen sus llaves primarias. Nótese la introducción en la relación Personas del atributo número_persona, a través del cual se asigna a cada persona un identificativo numérico unívoco que se usa como llave primaria. Estas relaciones contienen sólo atributos atómicos. Si una persona tiene más de un hijo, éstos se representaran en tuplas diferentes de la relación Hijos. Las diferentes características de los hijos las representan los atributos de la relación Hijos. La unión entre las dos relaciones está constituida por los atributos número_persona que aparecen en ambas relaciones y que permiten que se asigne cada tupla de la relación hijos a una tupla concreta de la relación Personas. Más formalmente se dice que el atributo número_persona de la relacion Hijos es una llave externa (foreign key) hacia la relación Personas. Una llave externa es una combinación de atributos de una relación que son, a su vez, una llave primaria para otra relación. Una característica fundamental de los valores presentes en una llave externa es que, a no ser que no sean null, tienen que corresponder a valores existentes en la llave primaria de la relación a la que se refieren. En nuestro ejemplo, esto significa que no puede existir en la relación Hijos una tupla con un valor del atributo número_persona sin que también en la relación Personas exista una tupla con el mismo valor para su llave primaria. Esta propiedad va bajo el nombre de integridad referencial (referential integrity).

Una de las grandes ventajas del modelo relacional es que define también un álgebra, llamada "álgebra relacional". Todas las manipulaciones posibles sobre las relaciones se obtienen gracias a la combinación de tan sólo cinco operadores: RESTRICT, PROJECT, TIMES, UNION y MINUS. Por comodidad, se han definido también tres operadores adicionales que de todos modos se pueden obtener aplicando los cinco fundamentales: JOIN, INTERSECT y DIVIDE. Los operadores relacionales reciben como argumento una relación o un conjunto de relaciones y restituyen una única relación como resultado.

Veamos brevemente estos ocho operadores:

RESTRICT: restituye una relación que contiene un subconjunto de las tuplas de la relación a la que se aplica. Los atributos se quedan como estaban.

PROJECT: restituye una relación con un subconjunto de los atributos de la relación a la que viene aplicado. Las tuplas de la

relación resultado se componen de las tuplas de la relacion original, de manera que siguen siendo un conjunto en sentido matemático.

TIME: se aplica a dos relaciones y efectúa el producto cartesiano de las tuplas. Cada tupla de la primera relación está concatenada con cada tupla de la segunda.

JOIN: se concatenan las tuplas de dos relaciones de acuerdo con el valor de un conjunto de sus atributos.

UNION: aplicando este operador a dos relaciones compatibles se obtiene una que contiene las tuplas de ambas relaciones. Dos relaciones son compatibles si tienen el mismo número de atributos y los atributos correspondientes en las dos relaciones tienen el mismo dominio.

MINUS: aplicado a dos relaciones compatibles restituye una tercera que contiene las tuplas que se encuentran sólo en la primera relación.

INTERSECT: aplicado a dos relaciones compatibles restituye una relación que contiene las tuplas que existen en ambas.

DIVIDE: aplicado a dos relaciones que tengan atributos comunes, restituye una tercera que contiene todas las tuplas de la primera relación que se puede hacer que correspondan con todos los valores de la segunda relación.

En las siguientes tablas, a título de ejemplo, se representan los resultados de la aplicación de algunos operadores relacionales a las relaciones Personas e Hijos. Como nombres para las relaciones resultado se han utilizado las expresiones que las producen.

Personas

número_persona	nombre	apellido	fecha_nacimiento	sexo	estado civil
2	Mario	Rossi	29/03/1965	M	Casado
1	Giuseppe	Russo	15/11/1972	M	Soltero
3	Alessandra	Mondella	13/06/1970	F	Soltera

Hijos

número_persona	**nombre_apellido**	**edad**	**sexo**
2	Maria Rossi	3	F
2	Gianni Rossi	5	M

RESTRICT *(Personas)*
sesso='M'

número_persona	**nombre**	**apellido**	**fecha_nacimiento**	**sexo**	**estado _civil**
2	Mario	Rossi	29/03/1965	M	Casado
1	Giuseppe	Russo	15/11/1972	M	Soltero

PROJECT *sexo* *(Personas)*
sexo
M
F
RESTRICT *(Personas)*
sexo='M'

n.	**nombre**	**apellido**	**nacimiento**	**sexo**	**stado_civil**	**nombre**	**edad**	**sexo**
Mario Rossi	**apellido**	**29/03/1965**	**M**	**Csado**	**Maria Rossi**	3	F
Mario Rossi	**Apellido**	**29/03/1965**	**M**	**Casado**	**Gianni Rossi**	5	M

Las bases de datos relacionales efectúan todas las operaciones en las tablas usando el álgebra relacional, aunque normalmente no le permiten al usuario usarla. El usuario interacciona con la base de datos a través de una interfaz diferente el lenguaje SQL, un lenguaje declarativo que permite escribir conjuntos de datos. Las instrucciones SQL vienen

descompuestas por el motor de datos en una serie de operaciones relacionales.

1.2.1Tipos de Datos

Los tipos de datos SQL se clasifican en 13 tipos de datos primarios y de varios sinónimos válidos reconocidos por dichos tipos de datos. Los tipos de datos primarios son:

Tipo de Datos	Longitud	Descripción
BINARY	1 byte	Para consultas sobre tabla adjunta de productos de bases de datos que definen un tipo de datos Binario.
BIT	1 byte	Valores Sí/No ó True/False
BYTE	1 byte	Un valor entero entre 0 y 255.
COUNTER	4 bytes	Un número incrementado automaticamente (de tipo Long)
CURRENCY	8 bytes	Un entero escalable entre 922.337.203.685.477,5808 y 922.337.203.685.477,5807.
DATETIME	8 bytes	Un valor de fecha u hora entre los años 100 y 9999.
SINGLE	4 bytes	Un valor en punto flotante de precisión simple con un rango de - $3.402823*10^{38}$ a $1.401298*10^{-45}$ para valores negativos, $1.401298*10^{-45}$ a $3.402823*10^{38}$ para valores positivos, y 0.
DOUBLE	8 bytes	Un valor en punto flotante de doble precisión con un rango de - $1.79769313486232*10^{308}$ a $4.94065645841247*10^{-324}$ para valores negativos, $4.94065645841247*10^{-324}$ a $1.79769313486232*10^{308}$ para valores positivos, y 0.
SHORT	2 bytes	Un entero corto entre -32,768 y 32,767.
LONG	4 bytes	Un entero largo entre -2,147,483,648 y 2,147,483,647.
LONGTEXT	1 byte por carácter	De cero a un máximo de 1.2 gigabytes.
LONGBINARY	Según se necesite	De cero 1 gigabyte. Utilizado para objetos OLE.
TEXT	1 byte por carácter	De cero a 255 caracteres.

La siguiente tabla recoge los sinónimos de los tipos de datos definidos:

Tipo de Dato	Sinónimos
BINARY	VARBINARY
BIT	BOOLEAN LOGICAL LOGICAL1 YESNO
BYTE	INTEGER1
COUNTER	AUTOINCREMENT
CURRENCY	MONEY
DATETIME	DATE TIME TIMESTAMP
SINGLE	FLOAT4 IEEESINGLE REAL
DOUBLE	FLOAT FLOAT8 IEEEDOUBLE NUMBER NUMERIC
SHORT	INTEGER2 SMALLINT.
LONG	INT INTEGER INTEGER4
LONGBINARY	GENERAL OLEOBJECT
LONGTEXT	LONGCHAR MEMO NOTE
TEXT	ALPHANUMERIC CHAR - CHARACTER STRING - VARCHAR
VARIANT (No Admitido)	VALUE

1.3 Modelo de Entidades y Asociaciones

1.3.1 Entidades

Se puede definir cono entidad a cualquier objeto, real o abstracto, que existe en un contexto determinado o puede llegar a existir y del cual

deseamos guardar información, por ejemplo: "PROFESOR", "CURSO", "ALUMNO". Las entidades las podemos clasificar en:

A. Regulares: aquellas que existen por sí mismas y que la existencia de un ejemplar en la entidad no depende de la existencia de otros ejemplares en otra entidad. Por ejemplo "EMPLEADO", "PROFESOR". La representación gráfica dentro del diagrama es la siguiente:

B. Débiles: son aquellas entidades en las que se hace necesaria la existencia de ejemplares de otras entidades distintas para que puedan existir ejemplares en esta entidad. Un ejemplo sería la entidad "ALBARÁN" que sólo existe si previamente existe el correspondiente pedido. La representación gráfica dentro del diagrama es la siguiente:

Como complemento al diagrama de entidades del modelo de datos, podemos utilizar la siguiente plantilla para definir las diferentes entidades:

Nombre	PROFESOR
Objeto	Almacenar la información relativa de los profesores de la organización.
Alcance	Se entiende como profesor a aquella persona que, contratada por la organización, imparte, al menos, un curso dentro de la misma.
Número de ejemplares	10 profesores
Crecimiento previsto	2 profesores / año

Confidencialidad	1. Nombre y apellidos: Acceso público. 2. Datos personales: Acceso restringido a secretaría y dirección. 3. Salario: Acceso restringido a direccion.
Derechos de Acceso	Para garantizar la total confidencialidad de esta entidad, el sistema de bases de datos deberá solicitar un usuario y una contraseña para visualizar los elementos de la misma.
Observaciones	Los ejemplares dados de baja no serán eliminados de la base de datos; pasarán a tener una marca de eliminado y no serán visualizados desde la aplicación.

1.3.2 Atributos

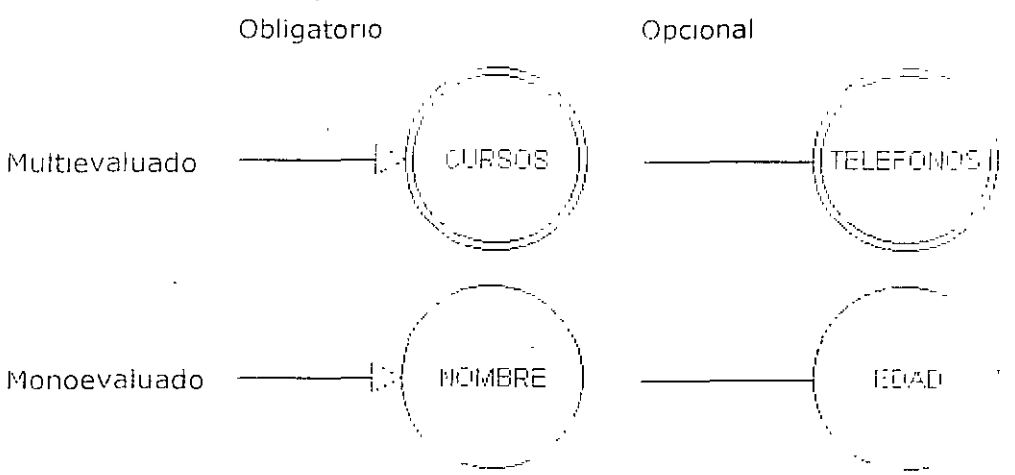
Las entidades se componen de atributos que son cada una de las propiedades o características que tienen las entidades. Cada ejemplar de una misma entidad posee los mismos atributos, tanto en nombre como en número, diferenciándose cada uno de los ejemplares por los valores que toman dichos atributos. Si consideramos la entidad "PROFESOR" y definimos los atributos Nombre, Teléfono y Salario, podríamos obtener los siguientes ejemplares:

{Luis García, 91.555.55.55, 80.500}
{Juan Antonio Alvarez, 91.666.66.66, 92.479}
{Marta López, 91.777.77.77, 85.396}

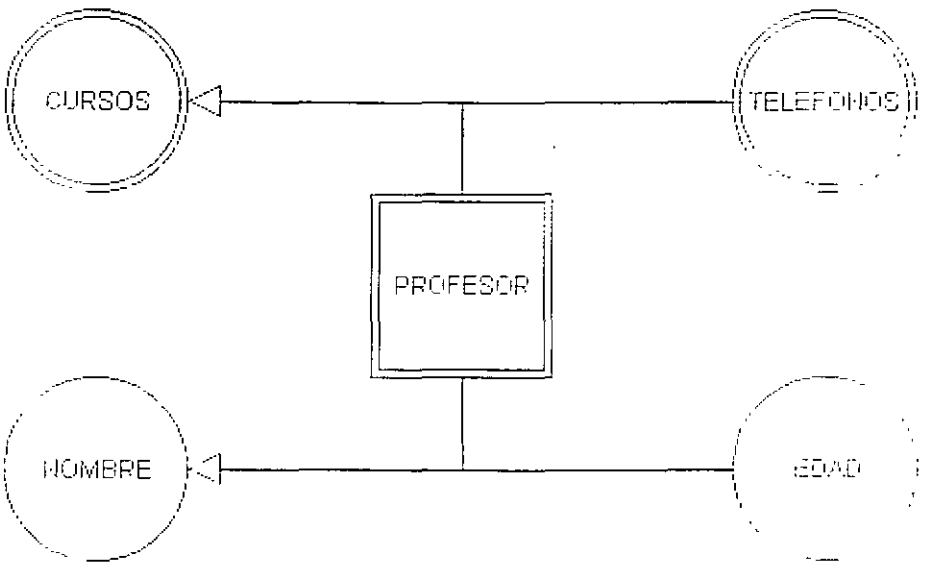
Existen cuatro tipos de atributos:

A. Obligatorios: aquellos que deben tomar un valor y no se permite ningún ejemplar no tenga un valor determinado en el atributo.
B. Opcional: aquellos atributos que pueden tener valores o no tenerlo.
C. Monoevaluado: aquel atributo que sólo puede tener un único valor.
D. Multievaluado: aquellos atributos que pueden tener varios valores.

La representación gráfica de los atributos, en función del tipo es la siguiente:

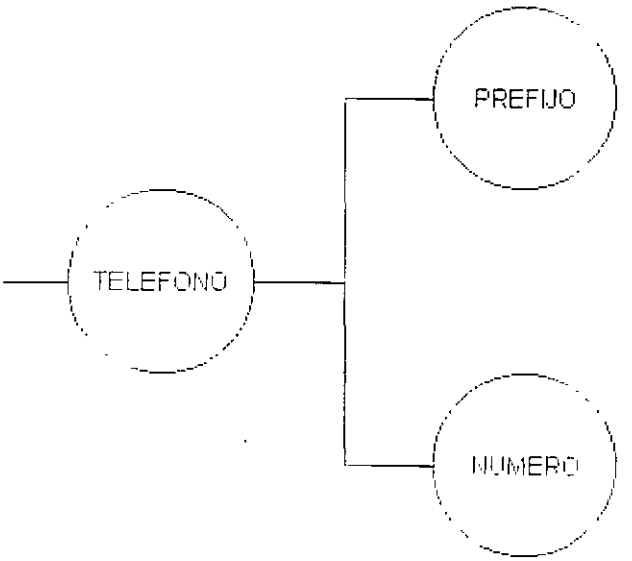


Dentro del diagrama la entidad "PROFESOR" y sus atributos quedaría de la siguiente forma:



Existen atributos, llamados derivados, cuyo valor se obtiene a partir de los valores de otros atributos. Pongamos como ejemplo la entidad "PROFESOR" que tiene los atributos "NOMBRE", "FECHA DE NACIMIENTO", "EDAD"; el atributo "EDAD" es un atributo derivado por que se calcula a partir del valor del atributo "FECHA DE NACIMIENTO". Su representación gráfica es la siguiente:

En determinadas ocasiones es necesaria la descomposición de un atributo para definirlos en más de un dominio, podría ser el caso del atributo "TELEFONO" que toma valores del dominio "PREFIJOS" y del dominio "NUMEROS DE TELEFONO". Estos atributos se representan de la siguiente forma:



Como complemento al diagrama de entidades del modelo de datos, podemos utilizar la siguiente plantilla para definir los diferentes atributos:

Nombre del atributo	FECHA DE NACIMIENTO
Tipo de dato	Número largo
Formato interno	aaaammdd
Longitud	8
Formato externo	dd/MM/aaaa
Descripción	Fecha de nacimiento del profesor
Dato requerido	SI
Permitir valor vacio	NO
Valor único	NO
Indexado	SI
Dominio	Calendario Gregoriano
Validaciones	La fecha debe ser superior a 01/01/1900
Confidencial	NO
Derechos de acceso	NO
Observaciones	

1.4 Proceso de normalización

El proceso de normalización es un estándar que consiste, básicamente, en un proceso de conversión de las relaciones entre las entidades, evitando:

- La redundancia de los datos: repetición de datos en un sistema.
- Anomalías de actualización: inconsistencias de los datos como resultado de datos redundantes y actualizaciones parciales.
- Anomalías de borrado: pérdidas no intencionadas de datos debido a que se han borrado otros datos.
- Anomalías de inserción: imposibilidad de adicionar datos en la base de datos debido a la ausencia de otros datos.

Tomando como referencia la tabla siguiente:

AUTORES Y LIBROS

NOMBRE	NACION	CODLIBRO	TITULO	EDITOR
Date	USA	999	IBD	AW
Ad.Mig.	ESP	888	CyD	RM
Ma.Piat.	ITA	777	CyD	RM
Date	USA	666	BdD	AW

Se plantean una serie de problemas:

- Redundancia: cuando un autor tiene varios libros, se repite la nacionalidad.
- Anomalías de modificación: Si Ad.Mig. y Ma.Piat. desean cambiar de editor, se modifica en los 2 lugares. A priori no podemos saber cuántos autores tiene un libro. Los errores son frecuentes al olvidar la modificación de un autor. Se pretende modificar en un sólo sitio.
- Anomalías de inserción: Se desea dar de alta un autor sin libros, en un principio. NOMBRE y CODLIBRO son campos clave, una clave no puede tomar valores nulos.

Asegurando:

- Integridad entre los datos: consistencia de la información.

El proceso de normalización nos conduce hasta el modelo físico de datos y consta de varias fases denominadas formas normales, estas formas se detallan a continuación.

1.4.1 Definición de la clave

Antes de proceder a la normalización de la tabla lo primero que debemos de definir es una clave, esta clave deberá contener un valor único para cada registro (no podrán existir dos valores iguales en toda la tabla) y podrá estar formado por un único campo o por un grupo de campos.

En la tabla de alumnos de un centro de estudios no podemos definir como campo clave el nombre del alumno ya que pueden existir varios alumnos con el mismo nombre. Podríamos considerar la posibilidad de definir como clave los campos nombre y apellidos, pero estamos en la misma situación: podría darse el caso de alumnos que tuvieran los mismo apellidos y el mismo nombre (Juan Fernández Martín)

La solución en este caso es asignar un código de alumno a cada uno, un número que identifique al alumno y que estemos seguros que es unico.

Una vez definida la clave podremos pasar a estudiar la primera forma normal.

1.4.2 Primera forma normal (1NF)

Se dice que una tabla se encuentra en primera forma normal (1NF) si y solo si cada uno de los campos contiene un unico valor para un registro determinado. Supongamos que deseamos realizar una tabla para guardar los cursos que están realizando los alumnos de un determinado centro de estudios, podríamos considerar el siguiente diseño:

Código	Nombre	Cursos
1	Marcos	Inglés
2	Lucas	Contabilidad, Informática
3	Marta	Inglés, Contabilidad

Podemos observar que el registro de código 1 si cumple la primera forma normal, cada campo del registro contiene un único dato, pero no ocurre así con los registros 2 y 3 ya que en el campo cursos contiene más de un dato cada uno. La solución en este caso es crear dos tablas del siguiente modo:

TABLA A		TABLA B	
Código	Nombre	Código	Curso
1	Marcos	1	Inglés
2	Lucas	2	Contabilidad
3	Marta	2	Informática

	3	Inglés
	3	Informática

Como se puede comprobar ahora todos los registros de ambas tablas contienen valores únicos en sus campos, por lo tanto ambas tablas cumplen la primera forma normal.

Una vez normalizada la tabla en 1NF, podemos pasar a la segunda forma normal.

1.4.3 Segunda forma normal (2NF)

La segunda forma normal compara todos y cada uno de los campos de la tabla con la clave definida. Si todos los campos dependen directamente de la clave se dice que la tabla está es segunda forma normal (2NF)

Supongamos que construimos una tabla con los años que cada empleado ha estado trabajando en cada departamento de una empresa:

Código Empleado	Código Dpto.	Nombre	Departamento	Años
1	6	Juan	Contabilidad	6
2	3	Pedro	Sistemas	
3	2	Sonia	I+D	
4	3	Verónica	Sistemas	
2	6	Pedro	Contabilidad	

Tomando como punto de partida que la clave de esta tabla está formada por los campos código de empleado y código de departamento, podemos decir que la tabla se encuentra en primera forma normal, por tanto vamos a estudiar la segunda:

1. El campo nombre no depende funcionalmente de toda la clave, sólo depende del código del empleado.
2. El campo departamento no depende funcionalmente de toda la clave, sólo del código del departamento.
3. El campo años si que depende funcionalmente de la clave ya que depende del código del empleado y del código del departamento (representa el número de años que cada empleado ha trabajado en cada departamento)

Por tanto, al no depender todos los campos de la totalidad de la clave la tabla no está en segunda forma normal, la solución es la siguiente:

Tabla A		Tabla B		Tabla C		
Código Empleado	Nombre	Código Departamento	Dpto.	Código Empleado	Código Departamento	ños
1	Juan	2	I+D	1	6	5
2	Pedro	3	Sistemas	2	3	
3	Sonia	6	Contabilidad	3	2	
4	Verónica			4	3	
				2	6	

Podemos observar que ahora si se encuentras las tres tabla en segunda forma normal, considerando que la tabla A tiene como indice el campo Código Empleado, la tabla B Código Departamento y la tabla C una clave compuesta por los campos Código Empleado y Código Departamento.

1.4.4 Tercera forma normal (3NF)

Se dice que una tabla está en tercera forma normal si y solo si todos los campos de la tabla dependen únicamente de la clave, dicho en otras palabras los campos de las tablas no dependen unos de otros. Tomando como referencia el ejemplo anterior, supongamos que cada alumno solo puede realizar un único curso a la vez y que deseamos guardar en que aula se imparte el curso. A voz de pronto podemos plantear la siguiente estructura:

Código	Nombre	Curso	Aula
1	Marcos	Informática	Aula A
2	Lucas	Inglés	Aula B
3	Marta	Contabilidad	Aula C

Estudiemos la dependencia de cada campo con respecto a la clave codigo:

- Nombre depende directamente del código del alumno.
- Curso depende de igual modo del código del alumno.
- El aula, aunque en parte tambien depende del alumno, esta mas ligado al curso que el alumno está realizando.

Por esta última razón se dice que la tabla no está en 3NF. La solucion sería la siguiente:

Tabla A			Tabla B	
Código	Nombre	Curso	Curso	Aula
1	Marcos	Informática	Informática	Aula A

2	Lucas	Inglés	Inglés	Aula B
3	Marta	Contabilidad	Contabilidad	Aula C

Una vez conseguida la segunda forma normal, se puede estudiar la cuarta forma normal.

1.4.5 Cuarta forma normal (4NF)

Una tabla está en cuarta forma normal si y sólo si para cualquier combinación clave - campo no existen valores duplicados. Veámoslo con un ejemplo:

Geometría

Figura	**Color**	**Tamaño**
Cuadrado	Rojo	Grande
Cuadrado	Azul	Grande
Cuadrado	Azul	Mediano
Círculo	Blanco	Mediano
Círculo	Azul	Pequeño
Círculo	Azul	Mediano

Comparemos ahora la clave (Figura) con el atributo Tamaño, podemos observar que Cuadrado Grande está repetido; igual pasa con Círculo Azul, entre otras. Estas repeticiones son las que se deben evitar para tener una tabla en 4NF.

La solución en este caso sería la siguiente:

Tamaño

Figura	**Tamaño**
Cuadrado	Grande
Cuadrado	Pequeño
Círculo	Mediano
Círculo	Pequeño

Color

Figura	**Color**
Cuadrado	Rojo
Cuadrado	Azul
Círculo	Blanco
Círculo	Azul

Ahora si tenemos nuestra base de datos en 4NF.

1.4.6 Otras formas normales

Existen otras dos formas normales, la llamada quinta forma normal (5FN) que no detallo por su dudoso valor práctico ya que conduce a una

gran división de tablas y la forma normal dominio / clave (FNDLL) de la que no existe método alguno para su implantación.

1.5 El SQL

1.5.1 Breve Historia del SQL

La historia de SQL (que se pronuncia deletreando en inglés las letras que lo componen, es decir "ese-cu-ele" y no "siquel" como se oye a menudo) empieza en 1974 con la definición, por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de IBM, de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba SEQUEL (Structured English Query Language) y se implementó en un prototipo llamado SEQUEL-XRM entre 1974 y 1975. Las experimentaciones con ese prototipo condujeron, entre 1976 y 1977, a una revisión del lenguaje (SEQUEL/2), que a partir de ese momento cambió de nombre por motivos legales, convirtiéndose en SQL. El prototipo (System R), basado en este lenguaje, se adoptó y utilizó internamente en IBM y lo adoptaron algunos de sus clientes elegidos. Gracias al éxito de este sistema, que no estaba todavía comercializado, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL. A partir de 1981, IBM comenzó a entregar sus productos relacionales y en 1983 empezó a vender DB2. En el curso de los años ochenta, numerosas compañías (por ejemplo Oracle y Sybase, sólo por citar algunos) comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho por lo que respecta a las bases de datos relacionales.

En 1986, el ANSI adoptó SQL (sustancialmente adoptó el dialecto SQL de IBM) como estándar para los lenguajes relacionales y en 1987 se transfomó en estándar ISO. Esta versión del estándar va con el nombre de SQL/86. En los años siguientes, éste ha sufrido diversas revisiones que han conducido primero a la versión SQL/89 y, posteriormente, a la actual SQL/92.

El hecho de tener un estándar definido por un lenguaje para bases de datos relacionales abre potencialmente el camino a la intercomunicabilidad entre todos los productos que se basan en él. Desde el punto de vista práctico, por desgracia las cosas fueron de otro modo. Efectivamente, en general cada productor adopta e implementa en la propia base de datos sólo el corazón del lenguaje SQL (el así llamado Entry level o al máximo el Intermediate level), extendiéndolo de manera individual según la propia visión que cada cual tenga del mundo de las bases de datos.

Actualmente, está en marcha un proceso de revisión del lenguaje por parte de los comités ANSI e ISO, que debería terminar en la definición de lo que en este momento se conoce como SQL3. Las características principales de esta nueva encarnación de SQL deberían ser su transformación en un lenguaje stand-alone (mientras ahora se usa como

lenguaje hospedado en otros lenguajes) y la introducción de nuevos tipos de datos más complejos que permitan, por ejemplo, el tratamiento de datos multimediales.

1.5.2 Introducción SQL

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado, utilizado por los diferentes motores de bases de datos para realizar determinadas operaciones sobre los datos o sobre la estructura de los mismos. Pero como sucede con cualquier sistema de normalización hay excepciones para casi todo; de hecho, cada motor de bases de datos tiene sus peculiaridades y lo hace diferente de otro motor, por lo tanto, el lenguaje SQL normalizado (ANSI) no nos servirá para resolver todos los problemas, aunque si se puede asegurar que cualquier sentencia escrita en ANSI será interpretable por cualquier motor de datos.

1.5.3 Componentes del SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

Comandos

Existen dos tipos de comandos SQL:

- DLL que permiten crear y definir nuevas bases de datos, campos e índices.
- DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DLL

Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e índices
DROP	Empleado para eliminar tablas e índices
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Comandos DML

Comando	Descripción
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados
DELETE	Utilizado para eliminar registros de una tabla de una base de datos

Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Cláusula	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

Operadores Lógicos

Operador	Uso
AND	Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

Operadores de Comparación

Operador	Uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor o igual que
>=	Mayor o igual que
=	Igual que
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo
In	Utilizado para especificar registros de una base de datos

Funciones de Agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Funciónn	Descripción
AVG	Utilizada para calcular el promedio de los valores de un campo determinado
COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un campo especificado
MIN	Utilizada para devolver el valor más bajo de un campo especificado

Orden de ejecución de los comandos

Dada una sentencia SQL de selección que incluye todas las posibles cláusulas, el orden de ejecución de las mismas es el siguiente:

1. Cláusula FROM
2. Cláusula WHERE
3. Cláusula GROUP BY
4. Cláusula HAVING
5. Cláusula SELECT
6. Cláusula ORDER BY

1.5.4 Consultas de Selección

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros puede ser modificable.

Consultas básicas

La sintaxis básica de una consulta de selección es la siguiente:

```
SELECT
        Campos
FROM
        Tabla
```

En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:

```
SELECT
        Nombre, Teléfono
FROM
        Clientes
```

Esta sentencia devuelve un conjunto de resultados con el campo nombre y teléfono de la tabla clientes.

Devolver Literales

En determinadas ocasiones nos puede interesar incluir una columna con un texto fijo en una consulta de selección, por ejemplo, supongamos que tenemos una tabla de empleados y deseamos recuperar las tarifas semanales de los electricistas, podríamos realizar la siguiente consulta:

```
SELECT
        Empleados.Nombre, 'Tarifa semanal: ', Empleados.TarifaHora * 40
FROM
```

```
    Empleados
WHERE
    Empleados.Cargo = 'Electricista'
```

Ordenar los registros

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula ORDER BY Lista de Campos. En donde Lista de campos representa los campos a ordenar. Ejemplo:

```
SELECT
    CodigoPostal, Nombre, Telefono
FROM
    Clientes
ORDER BY
    Nombre
```

Esta consulta devuelve los campos CodigoPostal, Nombre, Telefono de la tabla Clientes ordenados por el campo Nombre.

Se pueden ordenar los registros por mas de un campo, como por ejemplo:

```
SELECT
    CodigoPostal, Nombre, Telefono
FROM
    Clientes
ORDER BY
    CodigoPostal, Nombre
```

Incluso se puede especificar el orden de los registros: ascendente mediante la cláusula (ASC - se toma este valor por defecto) o descendente (DESC)

```
SELECT
    CodigoPostal, Nombre, Telefono
FROM
    Clientes
ORDER BY
    CodigoPostal DESC , Nombre ASC
```

Uso de Indices de las tablas

Si deseamos que la sentecia SQL utilice un índice para mostrar los resultados se puede utilizar la palabra reservada INDEX de la siguiente forma:

```
SELECT ... FROM Tabla (INDEX=Indice) ...
```

Normalmente los motores de las bases de datos deciden que índice se debe utilizar para la consulta, para ello utilizan criterios de rendimiento y sobre todo los campos de búsqueda especificados en la cláusula WHERE. Si se desea forzar a no utilizar ningún índice utilizaremos la siguiente sintaxis:

SELECT ... FROM Tabla (INDEX=0) ...

Consultas con Predicado

El predicado se incluye entre la cláusula y el primer nombre del campo a recuperar, los posibles predicados son:

Predicado	Descripción
ALL	Devuelve todos los campos de la tabla
TOP	Devuelve un determinado numero de registros de la tabla
DISTINCT	Omite los registros cuyos campos seleccionados coincidan totalmente
DISTINCTOW	Omite los registros duplicados basandose en la totalidad del registro y no sólo en los campos seleccionados.

ALL

Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL y devuelve todos y cada uno de sus campos. No es conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

```
SELECT ALL
FROM
      Empleados
SELECT *
FROM
      Empleados
```

TOP

Devuelve un cierto número de registros que entran entre al principio o al final de un rango especificado por una cláusula ORDER BY. Supongamos que queremos recuperar los nombres de los 25 primeros estudiantes del curso 1994:

```
SELECT TOP 25
      Nombre, Apellido
FROM
```

```
    Estudiantes
ORDER BY
    Nota DESC
```

Si no se incluye la cláusula ORDER BY, la consulta devolverá un conjunto arbitrario de 25 registros de la tabla de Estudiantes. El predicado TOP no elige entre valores iguales. En el ejemplo anterior, si la nota media número 25 y la 26 son iguales, la consulta devolverá 26 registros. Se puede utilizar la palabra reservada PERCENT para devolver un cierto porcentaje de registros que caen al principio o al final de un rango especificado por la cláusula ORDER BY. Supongamos que en lugar de los 25 primeros estudiantes deseamos el 10 por ciento del curso:

```
SELECT TOP 10 PERCENT
    Nombre, Apellido
FROM
    Estudiantes
ORDER BY
    Nota DESC
```

El valor que va a continuación de TOP debe ser un entero sin signo. TOP no afecta a la posible actualización de la consulta.

DISTINCT

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos. Por ejemplo, varios empleados listados en la tabla Empleados pueden tener el mismo apellido. Si dos registros contienen López en el campo Apellido, la siguiente instrucción SQL devuelve un único registro:

```
SELECT DISTINCT
    Apellido
FROM
    Empleados
```

Con otras palabras el predicado DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente. El resultado de una consulta que utiliza DISTINCT no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

DISTINCTROW

Este predicado no es compatible con ANSI. Que yo sepa a día de hoy sólo funciona con ACCESS.

Devuelve los registros diferentes de una tabla; a diferencia del predicado anterior que sólo se fijaba en el contenido de los campos seleccionados.

éste lo hace en el contenido del registro completo independientemente de los campos indicados en la cláusula SELECT.

```
SELECT DISTINCTROW
        Apellido
FROM Empleados
```

Si la tabla empleados contiene dos registros: Antonio López y Marta López el ejemplo del predicado DISTINCT devuelve un único registro con el valor López en el campo Apellido ya que busca no duplicados en dicho campo. Este último ejemplo devuelve dos registros con el valor López en el apellido ya que se buscan no duplicados en el registro completo.

ALIAS

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho o porque estamos recuperando datos de diferentes tablas y resultan tener un campo con igual nombre. Para resolver todas ellas tenemos la palabra reservada AS que se encarga de asignar el nombre que deseamos a la columna deseada. Tomado como referencia el ejemplo anterior podemos hacer que la columna devuelta por la consulta, en lugar de llamarse apellido (igual que el campo devuelto) se llame Empleado. En este caso procederíamos de la siguiente forma:

```
SELECT DISTINCTROW
        Apellido AS Empleado
FROM Empleados
```

AS no es una palabra reservada de ANSI, existen diferentes sistemas de asignar los alias en función del motor de bases de datos. En ORACLE para asignar un alias a un campo hay que hacerlo de la siguiente forma:

```
SELECT
        Apellido AS "Empleado"
FROM Empleados
```

También podemos asignar alias a las tablas dentro de la consulta de selección, en esta caso hay que tener en cuenta que en todas las referencias que deseemos hacer a dicha tabla se ha de utilizar el alias en lugar del nombre. Esta técnica será de gran utilidad más adelante cuando se estudien las vinculaciones entre tablas. Por ejemplo:

```
SELECT
        Apellido AS Empleado
FROM
        Empleados AS Trabajadores
```

Para asignar alias a las tablas en ORACLE y SQL-SERVER los alias se asignan escribiendo el nombre de la tabla, dejando un espacio en blanco y escribiendo el Alias (se asignan dentro de la cláusula FROM).

```
SELECT
        Trabajadores.Apellido (1) AS Empleado
FROM
        Empleados Trabajadores
```

(1)Esta nomenclatura [Tabla].[Campo] se debe utilizar cuando se esta recuperando un campo cuyo nombre se repite en varias de las tablas que se utilizan en la sentencia. No obstante cuando en la sentencia se emplean varias tablas es aconsejable utilizar esta nomenclatura para evitar el trabajo que supone al motor de datos averiguar en que tabla está cada uno de los campos indicados en la cláusua SELECT.

1.5.5 Recuperar Información de una base de Datos Externa

Para concluir este capítulo se debe hacer referencia a la recuperacion de registros de bases de datos externas. Es ocasiones es necesario la recuperación de información que se encuentra contenida en una tabla que no se encuentra en la base de datos que ejecutará la consulta o que en ese momento no se encuentra abierta, esta situación la podemos salvar con la palabra reservada IN de la siguiente forma:

```
SELECT
        Apellido AS Empleado
FROM
        Empleados IN c: \databases\gestion.mdb'
```

En donde c: \databases\gestion.mdb es la base de datos que contiene la tabla Empleados. Esta técnica es muy sencilla y común en bases de datos de tipo ACCESS en otros sistemas como SQL-SERVER u ORACLE. la cosa es más complicada la tener que existir relaciones de confianza entre los servidores o al ser necesaria la vinculación entre las bases de datos. Este ejemplo recupera la información de una base de datos de SQL-SERVER ubicada en otro servidor (se da por supuesto que los servidores están lincados):

```
SELECT
        Apellido
FROM
        Servidor1.BaseDatos1.dbo.Empleados
```

1.5.6 Consultas de Acción

Las consultas de acción son aquellas que no devuelven ningún registro, son las encargadas de acciones como añadir y borrar y modificar registros. Tanto las sentencias de actualización como las de borrado desencaderán (según el motor de datos) las actualizaciones en cascada, borrados en cascada, restricciones y valores por defecto definidos para los diferentes campos o tablas afectadas por la consulta.

DELETE

Crea una consulta de eliminación que elimina los registros de una o más de las tablas listadas en la cláusula FROM que satisfagan la cláusula WHERE. Esta consulta elimina los registros completos, no es posible eliminar el contenido de algún campo en concreto. Su sintaxis es:

```
DELETE FROM Tabla WHERE criterio
```

Una vez que se han eliminado los registros utilizando una consulta de borrado, no puede deshacer la operación. Si desea saber qué registros se eliminarán, primero examine los resultados de una consulta de selección que utilice el mismo criterio y después ejecute la consulta de borrado. Mantenga copias de seguridad de sus datos en todo momento. Si elimina los registros equivocados podrá recuperarlos desde las copias de seguridad.

```
DELETE
FROM
    Empleados
WHERE
    Cargo = 'Vendedor'
```

INSERT INTO

Agrega un registro en una tabla. Se la conoce como una consulta de datos añadidos. Esta consulta puede ser de dos tipo: Insertar un único registro ó Insertar en una tabla los registros contenidos en otra tabla.

Para insertar un único Registro:

En este caso la sintaxis es la siguiente:

```
INSERT INTO Tabla (campo1, campo2, ..., campoN)
VALUES (valor1, valor2, ..., valorN)
```

Esta consulta graba en el campo1 el valor1, en el campo2 y valor2 y así sucesivamente.

Para seleccionar registros e insertarlos en una tabla nueva

En este caso la sintaxis es la siguiente:

```
SELECT campo1, campo2, ..., campoN INTO nuevatabla
FROM tablaorigen [WHERE criterios]
```

Se pueden utilizar las consultas de creación de tabla para archivar registros, hacer copias de seguridad de las tablas o hacer copias para exportar a otra base de datos o utilizar en informes que muestren los datos de un periodo de tiempo concreto. Por ejemplo, se podría crear un informe de Ventas mensuales por región ejecutando la misma consulta de creación de tabla cada mes.

Para insertar Registros de otra Tabla:

En este caso la sintaxis es:

```
INSERT INTO Tabla [IN base_externa] (campo1, campo2, ..., campoN)
SELECT TablaOrigen.campo1, TablaOrigen.campo2,,TablaOrigen.campoN FROM Tabla Origen
```

En este caso se seleccionarán los campos 1,2,..., n de la tabla origen y se grabarán en los campos 1,2,.., n de la Tabla. La condición SELECT puede incluir la cláusula WHERE para filtrar los registros a copiar. Si Tabla y Tabla Origen poseen la misma estructura podemos simplificar la sintaxis a:

```
INSERT INTO Tabla SELECT Tabla Origen.* FROM Tabla Origen
```

De esta forma los campos de Tabla Origen se grabarán en Tabla, para realizar esta operación es necesario que todos los campos de Tabla Origen estén contenidos con igual nombre en Tabla. Con otras palabras que Tabla posea todos los campos de Tabla Origen (igual nombre e igual tipo).

En este tipo de consulta hay que tener especial atención con los campos contadores o autonuméricos puesto que al insertar un valor en un campo de este tipo se escribe el valor que contenga su campo homólogo en la tabla origen, no incrementándose como le corresponde.

Se puede utilizar la instrucción INSERT INTO para agregar un registro único a una tabla, utilizando la sintaxis de la consulta de adición de registro único tal y como se mostró anteriormente. En este caso, su código especifica el nombre y el valor de cada campo del registro. Debe especificar cada uno de los campos del registro al que se le va a asignar un valor así como el valor para dicho campo. Cuando no se especifica dicho campo, se inserta el valor predeterminado o Null. Los registros se agregan al final de la tabla.

También se puede utilizar INSERT INTO para agregar un conjunto de registros pertenecientes a otra tabla o consulta utilizando la cláusula SELECT... FROM como se mostró anteriormente en la sintaxis de la consulta de adición de múltiples registros. En este caso la cláusula SELECT especifica los campos que se van a agregar en la tabla destino especificada.

La tabla destino u origen puede especificar una tabla o una consulta. Si la tabla destino contiene una clave principal, hay que asegurarse que es única, y con valores no nulos; si no es así, no se agregarán los registros. Si se agregan registros a una tabla con un campo Contador, no se debe incluir el campo Contador en la consulta. Se puede emplear la cláusula IN para agregar registros a una tabla en otra base de datos.

Se pueden averiguar los registros que se agregarán en la consulta ejecutando primero una consulta de selección que utilice el mismo criterio de selección y ver el resultado. Una consulta de adición copia los registros de una o más tablas en otra. Las tablas que contienen los registros que se van a agregar no se verán afectadas por la consulta de adición. En lugar de agregar registros existentes en otra tabla, se puede especificar los valores de cada campo en un nuevo registro utilizando la cláusula VALUES. Si se omite la lista de campos, la cláusula VALUES debe incluir un valor para cada campo de la tabla, de otra forma fallara INSERT.

Ejemplos

```
INSERT INTO
        Clientes
SELECT
        ClientesViejos.*
FROM
        ClientesNuevos
SELECT
        Empleados.*
INTO Programadores
FROM
        Empleados
WHERE
        Categoria = 'Programador'
```

Esta consulta crea una tabla nueva llamada programadores con igual estructura que la tabla empleado y copia aquellos registros cuyo campo categoria se programador

```
INSERT INTO
        Empleados (Nombre, Apellido, Cargo)
VALUES
        (
        'Luis', 'Sánchez', 'Becario'
        )
INSERT INTO
        Empleados
```

```
SELECT
        Vendedores.*
FROM
        Vendedores
WHERE
        Provincia = 'Madrid'
```

UPDATE

Crea una consulta de actualización que cambia los valores de los campos de una tabla especificada basándose en un criterio específico. Su sintaxis es:

```
UPDATE Tabla SET Campo1=Valor1, Campo2=Valor2,
CampoN=ValorN
WHERE Criterio
```

UPDATE es especialmente útil cuando se desea cambiar un gran número de registros o cuando éstos se encuentran en múltiples tablas. Puede cambiar varios campos a la vez. El ejemplo siguiente incrementa los valores Cantidad pedidos en un 10 por ciento y los valores Transporte en un 3 por ciento para aquellos que se hayan enviado al Reino Unido.:

```
UPDATE
        Pedidos
SET
        Pedido = Pedidos * 1.1,
        Transporte = Transporte * 1.03
WHERE
        PaisEnvío = 'ES'
```

UPDATE no genera ningún resultado. Para saber qué registros se van a cambiar, hay que examinar primero el resultado de una consulta de selección que utilice el mismo criterio y después ejecutar la consulta de actualización.

```
UPDATE
        Empleados
SET
        Grado = 5
WHERE
        Grado = 2
UPDATE
        Productos
SET
        Precio = Precio * 1.1
WHERE
        Proveedor = 8
        AND
        Familia = 3
```

Si en una consulta de actualización suprimimos la cláusula WHERE todos los registros de la tabla señalada serán actualizados.

```
UPDATE
        Empleados
SET
        Salario = Salario * 1.1
```

Consultas de Unión Internas

Consultas de Combinación entre tablas

Las vinculaciones entre tablas se realizan mediante la cláusula INNER que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Su sintaxis es:

```
SELECT campos FROM tb1 INNER JOIN tb2 ON
tb1.campo1 comp tb2.campo2
```

En donde:

tb1, tb2	Son los nombres de las tablas desde las que se combinan los registros.
campo1, campo2	Son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.
comp	Es cualquier operador de comparación relacional: =, <, >, <=, =>, ó >.

Se puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones equivalentes son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas. Se puede utilizar INNER JOIN con las tablas Departamentos y Empleados para seleccionar todos los empleados de cada departamento. Por el contrario, para seleccionar todos los departamentos (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea LEFT JOIN o todos los empleados (incluso si alguno no está asignado a ningún departamento), en este caso RIGHT JOIN.

Si se intenta combinar campos que contengan datos Memo u Objeto OLE, se produce un error. Se pueden combinar dos campos numéricos cualesquiera, incluso si son de diferente tipo de datos. Por ejemplo, puede combinar un campo Numérico para el que la propiedad Size de su objeto Field está establecida como Entero, y un campo Contador.

El ejemplo siguiente muestra cómo podría combinar las tablas Categorías y Productos basándose en el campo IDCategoria:

```
SELECT
    NombreCategoria, NombreProducto
FROM
    Categorias
```

```
INNER JOIN
Productos
ON
Categorias.IDCategoria = Productos.IDCategoria
```

En el ejemplo anterior, IDCategoria es el campo combinado, pero no está incluido en la salida de la consulta ya que no está incluido en la instrucción SELECT. Para incluir el campo combinado, incluir el nombre del campo en la instrucción SELECT, en este caso, Categorias.IDCategoria.

También se pueden enlazar varias cláusulas ON en una instruccion JOIN, utilizando la sintaxis siguiente:

```
SELECT campos FROM tabla1 INNER JOIN tabla2
ON (tb1.campo1 comp tb2.campo1 AND ON tb1.campo2 comp
tb2.campo2)
OR ON (tb1.campo3 comp tb2.campo3)
```

También puede anidar instrucciones JOIN utilizando la siguiente sintaxis:

```
SELECT campos FROM tb1 INNER JOIN (tb2 INNER JOIN [( ]tb3
[INNER JOIN [( ]tablax [INNER JOIN ...)]
ON tb3.campo3 comp tbx.campox)]
ON tb2.campo2 comp tb3.campo3)
ON tb1.campo1 comp tb2.campo2
```

Un LEFT JOIN o un RIGHT JOIN puede anidarse dentro de un INNER JOIN, pero un INNER JOIN no puede anidarse dentro de un LEFT JOIN o un RIGHT JOIN.

Ejemplo:

```
SELECT DISTINCT
    Sum(PrecioUnitario * Cantidad) AS Sales,
    (Nombre + ' ' + Apellido) AS Name
FROM
    Empleados
INNER JOIN(
    Pedidos
INNER JOIN
    DetallesPedidos
    ON
    Pedidos.IdPedido = DetallesPedidos.IdPedido)
    ON
    Empleados.IdEmpleado = Pedidos.IdEmpleado
GROUP BY
    Nombre + ' ' + Apellido
```

(Crea dos combinaciones equivalentes: una entre las tablas Detalles de pedidos y Pedidos, y la otra entre las tablas Pedidos y

Empleados. Esto es necesario ya que la tabla Empleados no contiene datos de ventas y la tabla Detalles de pedidos no contiene datos de los empleados. La consulta produce una lista de empleados y sus ventas totales.)

Si empleamos la cláusula INNER en la consulta se seleccionarán solo aquellos registros de la tabla de la que hayamos escrito a la izquierda de INNER JOIN que contengan al menos un registro de la tabla que hayamos escrito a la derecha. Para solucionar esto tenemos dos cláusulas que sustituyen a la palabra clave INNER, estas cláusulas son LEFT y RIGHT. LEFT toma todos los registros de la tabla de la izquierda aunque no tengan ningún registro en la tabla de la izquierda. RIGHT realiza la misma operación pero al contrario, toma todos los registros de la tabla de la derecha aunque no tenga ningún registro en la tabla de la izquierda.

La sintaxis expuesta anteriormente pertenece a ACCESS, en donde todas las sentencias con la sintaxis funcionan correctamente. Los manuales de SQL-SERVER dicen que esta sintaxis es incorrecta y que hay que añadir la palabra reservada OUTER: LEFT OUTER JOIN y RIGHT OUTER JOIN. En la práctica funciona correctamente de una u otra forma.

No obstante, los INNER JOIN ORACLE no es capaz de interpretarlos, pero existe una sintaxis en formato ANSI para los INNER JOIN que funcionan en todos los sistemas. Tomando como referencia la siguiente sentencia:

```
SELECT
        Facturas.*,
        Albaranes.*
FROM
        Facturas
        INNER JOIN
        Albaranes
        ON
        Facturas.IdAlbaran = Albaranes.IdAlbaran
WHERE
        Facturas.IdCliente = 325
```

La transformación de esta sentencia a formato ANSI sería la siguiente:

```
SELECT
        Facturas.*,
        Albaranes.*
FROM
        Facturas, Albaranes
WHERE
        Facturas.IdAlbaran = Albaranes.IdAlbaran
        AND
        Facturas.IdCliente = 325
```

Como se puede observar los cambios realizados han sido los siguientes:

1. Todas las tablas que intervienen en la consulta se especifican en la cláusula FROM.
2. Las condiciones que vinculan a las tablas se especifican en la cláusula WHERE y se vinculan mediante el operador lógico AND.

Referente a los OUTER JOIN, no funcionan en ORACLE y además conozco una sintaxis que funcione en los tres sistemas. La sintaxis en ORACLE es igual a la sentencia anterior pero añadiendo los caracteres (+) detrás del nombre de la tabla en la que deseamos aceptar valores nulos, esto equivale a un LEFT JOIN:

```
SELECT
        Facturas.*,
        Albaranes.*
FROM
        Facturas, Albaranes
WHERE
        Facturas.IdAlbaran = Albaranes.IdAlbaran (+)
        AND
        Facturas.IdCliente = 325
```

Y esto a un RIGHT JOIN:

```
SELECT
        Facturas.*,
        Albaranes.*
FROM
        Facturas, Albaranes
WHERE
        Facturas.IdAlbaran (+) = Albaranes.IdAlbaran
        AND
        Facturas.IdCliente = 325
```

En SQL-SERVER se puede utilizar una sintaxis parecida, en este caso no se utiliza los caracteres (+) sino los caracteres =* para el LEFT JOIN y *= para el RIGHT JOIN.

Consultas de Autocombinación

La autocombinación se utiliza para unir una tabla consigo misma, comparando valores de dos columnas con el mismo tipo de datos. La sintaxis en la siguiente:

```
SELECT
        alias1.columna, alias2.columna, ...
FROM
        tabla1 as alias1, tabla2 as alias2
WHERE
```

```
        alias1.columna = alias2.columna
AND
        otras condiciones
```

Por ejemplo, para visualizar el número, nombre y puesto de cada empleado, junto con el número, nombre y puesto del supervisor de cada uno de ellos se utilizaría la siguiente sentencia:

```
SELECT
        t.num_emp, t.nombre, t.puesto, t.num_sup,s.nombre, s.puesto
FROM
        empleados AS t, empleados AS s
WHERE
        t.num_sup = s.num_emp
```

Consultas de Combinaciones no Comunes

La mayoría de las combinaciones están basadas en la igualdad de valores de las columnas que son el criterio de la combinación. Las no comunes se basan en otros operadores de combinación, tales como NOT, BETWEEN, <>, etc.

Por ejemplo, para listar el grado salarial, nombre, salario y puesto de cada empleado ordenando el resultado por grado y salario habría que ejecutar la siguiente sentencia:

```
SELECT
        grados.grado,empleados.nombre,          empleados.salario,
        empleados.puesto
FROM
        empleados, grados
WHERE
        empleados.salario    BETWEEN    grados.salarioinferior    And
        grados.salariosuperior
ORDER BY
        grados.grado, empleados.salario
```

Para listar el salario medio dentro de cada grado salarial habría que lanzar esta otra sentencia:.

```
SELECT
        grados.grado, AVG(empleados.salario)
FROM
        empleados, grados
WHERE
        empleados.salario    BETWEEN    grados.salarioinferior    And
        grados.salariosuperior
GROUP BY
        grados.grado
```

CROSS JOIN (SQL-SERVER)

Se utiliza en SQL-SERVER para realizar consultas de unión. Supongamos que tenemos una tabla con todos los autores y otra con todos los libros. Si deseáramos obtener un listado combinar ambas tablas de tal forma que cada autor apareciera junto a cada título, utilizaríamos la siguiente sintaxis:

```
SELECT
        Autores.Nombre, Libros.Titulo
FROM
        Autores CROSS JOIN Libros
```

SELF JOIN

SELF JOIN es una técnica empleada para conseguir el producto cartesiano de una tabla consigo misma. Su utilización no es muy frecuente, pero pongamos algún ejemplo de su utilizacion. Supongamos la siguiente tabla (El campo autor es numérico, aunque para ilustrar el ejemplo utilice el nombre):

Autores

Código (Código del libro)	Autor (Nombre del Autor)
B0012	1. Francisco López
B0012	2. Javier Alonso
B0012	3. Marta Rebolledo
C0014	1. Francisco López
C0014	2. Javier Alonso
D0120	2. Javier Alonso
D0120	3. Marta Rebolledo

Queremos obtener, para cada libro, parejas de autores:

```
SELECT
        A.Codigo, A.Autor, B.Autor
FROM
        Autores A, Autores B
WHERE
        A.Codigo = B.Codigo
```

El resultado es el siguiente:

Código	Autor	Autor
B0012	1. Francisco López	1. Francisco López

B0012	1. Francisco López	2. Javier Alonso
B0012	1. Francisco López	3. Marta Rebolledo
B0012	2. Javier Alonso	2. Javier Alonso
B0012	2. Javier Alonso	1. Francisco López
B0012	2. Javier Alonso	3. Marta Rebolledo
B0012	3. Marta Rebolledo	3. Marta Rebolledo
B0012	3. Marta Rebolledo	2. Javier Alonso
B0012	3. Marta Rebolledo	1. Francisco López
C0014	1. Francisco López	1. Francisco López
C0014	1. Francisco López	2. Javier Alonso
C0014	2. Javier Alonso	2. Javier Alonso
C0014	2. Javier Alonso	1. Francisco López
D0120	2. Javier Alonso	2. Javier Alonso
D0120	2. Javier Alonso	3. Marta Rebolledo
D0120	3. Marta Rebolledo	3. Marta Rebolledo
D0120	3. Marta Rebolledo	2. Javier Alonso

Como podemos observar, las parejas de autores se repiten en cada uno de los libros, podemos omitir estas repeticiones de la siguiente forma

```
SELECT
    A.Codigo, A.Autor, B.Autor
FROM
    Autores A, Autores B
WHERE
    A.Codigo = B.Codigo AND A.Autor < B.Autor
```

El resultado ahora es el siguiente:

Código	Autor	Autor
B0012	1. Francisco López	2. Javier Alonso
B0012	1. Francisco López	3. Marta Rebolledo
C0014	1. Francisco López	2. Javier Alonso
D0120	2. Javier Alonso	3. Marta Rebolledo

Ahora tenemos un conjunto de resultados en formato Autor - CoAutor.

Si en la tabla de empleados quisiéramos extraer todas las posibles parejas que podemos realizar, utilizaríamos la siguiente sentencia:

```
SELECT
    Hombres.Nombre, Mujeres.Nombre
FROM
    Empleados Hombre, Empleados Mujeres
WHERE
    Hombre.Sexo = 'Hombre' AND
    Mujeres.Sexo = 'Mujer' AND
    Hombres.Id <>Mujeres.Id
```

Para concluir supongamos la tabla siguiente:

Empleados

Id	Nombre	SuJefe
1	Marcos	6
2	Lucas	1
3	Ana	2
4	Eva	1
5	Juan	6
6	Antonio	

Queremos obtener un conjunto de resultados con el nombre del empleado y el nombre de su jefe:

```
SELECT
    Emple.Nombre, Jefes.Nombre
FROM
    Empleados Emple, Empleados Jefe
WHERE
    Emple.SuJefe = Jefes.Id
```

FULL JOIN

Este tipo de operador se utiliza para devolver todas las filas de una combinación tengan o no correspondencia. Es el equivalente a la utilización de LEFT JOIN y RIGHT JOIN a la misma vez. Mediante este operador se obtendrán por un lado las filas que tengan correspondencia en ambas tablas y también aquellas que no tengan correspondencia sean de la tabla que sean.

Si desearamos obtener un listado que incluyera todos los autores con sus libros correspondientes, pero además todos los autores que no han escrito ningún libro y todos aquellos libros sin autor (devemos suponer que no existe un autor llamado anónimo):

```
SELECT
    Autores.*, Libros.*
```

```
FROM
        Autores FULL Libros
ON
        Autores.IdAutor = Libros.IdAutor
```

1.6 Implementación del Sistema

1.6.1 Criterios de calidad

1.6.1.1 Legibilidad

El diseño de una base de datos ha de estar redactado con la suficiente claridad para que pueda ser entendido rápidamente. El lenguaje utilizado debe ser lo suficientemente claro, conciso y detallado para que explique con total claridad el diseño del modelo, sus objetivos, sus restricciones, en general todo aquello que afecte al sistema de forma directa o indirecta. En este punto conviene aplicar el principio que una imagen vale más que mil palabras, pero en ocasiones son necesarias esas mil palabras y obviar la imagen.

1.6.1.2 Fiabilidad

Se trata de realizar un sistema de bases de datos lo suficientemente robusto para que sea capaz de recuperarse frente a errores o usos inadecuados. Se deben utilizar gestores con las herramientas necesarias para la reparación de los posibles errores que las bases de datos pueden sufrir, por ejemplo tras un corte inesperado de luz.

1.6.1.3 Portabilidad

El diseño deber permitir la implementación del modelo físico en diferentes gestores de bases de datos.

1.6.1.4 Modificabilidad

Ningún sistema informático es estático, las necesidades de los usuarios varían con el tiempo y por lo tanto las bases de datos se deben adaptar a las nuevas necesidades, por lo que se precisa que un buen diseño facilite el mantenimiento, esto es, las modificaciones y actualizaciones necesarias para adaptarlo a una nueva situación.

1.6.1.5 Eficiencia

Se deben aprovechar al máximo los recursos de la computadora, minimizando la memoria utilizada y el tiempo de proceso o ejecución, siempre que no sea a costa de los requisitos anteriores. En este punto se debe tener en cuenta los gestores cliente / servidor de bases de datos. En muchas ocasiones es más rentable cargar de trabajo al servidor y liberar recursos de los clientes, pero no todos los gestores permiten este tipo de trabajo, por lo tanto se ha de tener en cuenta estas dos circunstancias en el diseño de la base de datos.

1.6.1.6 Auto descripción

En la documentación generada debe estar todo el detalle del diseño, evitando referencias a otros documentos que no estén incluidos dentro de la documentación de la base de datos.

1.6.1.7 Trivialidad

Tanto el diseño como la implantación se deben realizar utilizando los estándares fijados a priori, estos estándares deberán quedar reflejados al inicio del documento.

1.6.1.8 Claridad

Todos los documentos deben estar redactados de forma clara y fácil de entender, los nombre utilizados para las tablas, los campos, índices, etc. deben ser autodescriptivos y estar perfectamente documentados.

1.6.1.9 Coherencia

Las anotaciones y terminología utilizada deben ser uniformes, para ello se debe seguir algún tipo de metodología estándar, indicado cual se ha empleado, en los casos en que se utilice alguna metodología no estándar se debe adjuntar a la documentación.

1.6.1.10 Completo

Todos los elementos constitutivos de la base de datos existen, no se han dejado partes incompletas, sin documentar o sin implementar.

1.6.1.11 Concisión

No existen elementos inútiles ni repetitivos. En este apartado hay que hacer un especial hincapié en la repetición de datos en diferentes tablas, hay que evitar a toda costa que el mismo dato se repita en varias tablas para conseguir así una optimización del tamaño de la base de datos.

1.6.1.12 Facilidad de Aprendizaje

La documentación de la base de datos se puede utilizar sin necesidad de otros conocimientos informáticos fuera del alcance del diseño e implementación de la base de datos.

1.6.1.13 Facilidad de Uso

Los datos deben ser fáciles de elaborar y los resultados fáciles de entender.

1.6.1.14 Generalidad

La base de datos debe ser capaz de adaptarse a cualquier tipo de empresa y a cualquier casuística.

1.6.1.15 Independencia de Usuario

La base de datos no debe estar ligada a la utilización en una única instalación, hay que tener en cuenta que, aunque se trate de un desarrollo a medida, en un futuro se podría realizar la instalación en un cliente diferente al inicial.

1.6.1.16 Independencia de Sistema

Las prestaciones y diseño de la base de datos no están vinculadas al entorno.

1.6.1.17 Independencia de Instalación

La base de datos se puede transportar fácilmente de una instalación a otra.

1.6.1.18 Modularidad

La base de datos puede ser descompuesta en elementos independientes. Si se trata de un diseño grande, en donde hay un gran número de tablas, conviene realizar agrupaciones entre ella, creando módulos funcionales que permitan la mejor compresión del diseño y de la implantación.

1.6.1.19 Observable

La base de datos debe permitir observar los accesos a los datos. Siempre que se pueda hay que dejar un rastro de la utilización de los datos por parte de los usuarios, esta información ayuda al redimensionado de la base de datos y a conocer el número de accesos a los datos.

1.6.1.20 Precisión

Los cálculos efectuados se deben realizar con la precisión requerida.

1.6.1.21 Protección

La base de datos debe permitir la protección de los datos frente a usos no debidos, para ello hay que elaborar un sistema de accesos definiendo diferentes usuarios con diferentes claves y especificar que autorizaciones tendrá cada usuario sobre los diferentes datos.

1.6.1.22 Trazabilidad

Tomando como punto de partida la versión actual se puede remontar su diseño hasta las especificaciones iniciales

1.6.2 Indicadores de calidad

Al finalizar el diseño de una base de datos podemos utilizar la siguiente tabla para comprobar el grado de calidad del trabajo.

	1	2	3	4	5	6	7	8	9	10
Legibilidad										
Fiabilidad										
Portabilidad										
Modificabilidad										
Eficiencia										
Auto Descripción										
Trivialidad										
Claridad										
Coherencia										
Completo										
Conciso										
Facilidad de Aprendizaje										
Facilidad de Uso										
Generalidad										
Independencia de Usuario										
Independencia del Sistema										
Independencia de Instalación										
Modularidad										
Observable										
Precision										
Protección										
Trazable										
Legibilidad										
TOTAL										
PUNTUACION FINAL										