



Universidad Nacional Autónoma de México  
Facultad de Ingeniería

# Manual de prácticas del Laboratorio de Bases de Datos

**División de Ingeniería Eléctrica**

Departamento de Ingeniería en Computación



## **PRACTICAS DE LABORATORIO DE LA ASIGNATURA DE BASES DE DATOS**

El siguiente documento es una guía de las prácticas que se realizarán en la asignatura de Bases de Datos durante el semestre.

Cada una de estas prácticas tiene un objetivo específico, que representa el aprendizaje que debe de adquirir el alumno.

Las actividades que se desarrollarán en cada una de estas prácticas fueron elaboradas por los profesores de la asignatura, los ejercicios, casos de estudio a utilizar, diagramas y scripts serán proporcionadas por cada profesor.

Las prácticas están diseñadas para realizarse una por semana a partir de la segunda semana de clase, la fecha de entrega de cada una de estas prácticas será indicada por cada profesor.

Las prácticas fueron elaboradas por los profesores:

- Arellano Mendoza Lucila Patricia
- Hernández Hernández Luciralia
- Rodríguez Campos Jorge Alberto

**INDICE**

PRACTICA 1. Entorno de Trabajo.....	1
Objetivo:.....	1
Introducción .....	1
Actividades a realizar: .....	2
PRACTICA 2. Lenguaje de Control de Datos .....	3
Objetivo:.....	3
Introducción .....	3
Actividades a realizar: .....	4
PRACTICA 3. Diseño de modelos básicos entidad relación con notación CHEN utilizando una herramienta CASE .....	5
Objetivo:.....	5
Introducción .....	5
Actividades a realizar: .....	8
PRACTICA 4. Diseño de modelos avanzados entidad relación con notación CHEN utilizando una herramienta CASE .....	9
Objetivo:.....	9
Introducción .....	9
Actividades a realizar: .....	11
PRACTICA 5. Diseño básico de modelos relacionales .....	12
Objetivo:.....	12
Introducción .....	12
Actividades a realizar: .....	17
PRACTICA 6. Diseño avanzado de modelos relacionales .....	18
Objetivo:.....	18
Introducción .....	18
Actividades a realizar: .....	21
PRACTICA 7. Lenguaje de definición de datos.....	22
Objetivo:.....	22
Introducción .....	22
Actividades a realizar: .....	24
PRACTICA 8. Lenguaje de manipulación de datos y transacciones.....	25
Objetivo:.....	25

Introducción .....	25
Actividades a realizar: .....	27
PRACTICA 9. Álgebra relacional.....	28
Objetivo:.....	28
Introducción .....	28
Actividades a realizar: .....	30
PRACTICA 10. Consultas básicas en SQL y funciones de agregación.....	31
Objetivo:.....	31
Introducción .....	31
Actividades a realizar: .....	35
PRACTICA 11. Utilización de distintos tipos de JOIN, subconsultas y vistas .....	36
Objetivo:.....	36
Introducción .....	36
Actividades a realizar: .....	36
PRACTICA 12. Programación con SQL. Parte 1 .....	37
Objetivo:.....	37
Introducción .....	37
Actividades a realizar: .....	38
PRACTICA 13. Programación con SQL. Parte 2 .....	39
Objetivo:.....	39
Introducción .....	39
Actividades a realizar: .....	39
Bibliografía .....	40

## PRACTICA 1. Entorno de Trabajo

### Objetivo:

El alumno conocerá el entorno de trabajo del manejador de base de datos que utilizará en las prácticas siguientes

### Introducción

El entorno de trabajo está formado básicamente por las siguientes herramientas:

- Herramienta CASE empleada para realizar el diseño conceptual de una base de datos. El principal elemento a generar es un modelo Entidad/Relacion (ER) empleando el estándar Chen, que lleva el nombre de su creador Peter Chen en 1976.
- Herramienta CASE empleada para realizar el diseño lógico de una base de datos relacional. En esta etapa el principal elemento a generar es un modelo relacional empleando 2 notaciones comúnmente empleadas en la industria: Notación Crow's Foot y notación IDEF1X.
- Manejador de Bases de Datos relacionales.
- Herramientas para interactuar con el manejador:
  - Herramientas a línea de comandos. Es importante que el alumno comprenda y sea capaz de interactuar con el cliente a línea de comandos que ofrecen los manejadores de Bases de Datos.
  - Herramientas gráficas para interactuar con una Base de Datos.

Para iniciar con el uso adecuado del entorno de trabajo es necesario realizar una configuración inicial antes de iniciar con el desarrollo de las prácticas, en especial la configuración asociada al manejador de la Base de Datos. Los siguientes puntos generales deben ser desarrollados antes de iniciar:

- Instalación del manejador
- Creación de una Base de datos.
- Configuración de los servicios de red que permitan el acceso remoto a los equipos del laboratorio.
- Creación de los objetos que permitan al alumno establecer conexiones remotas. Por ejemplo, creación de usuarios, creación de esquemas.

La revisión del entorno de trabajo consiste en ejecutar algunas instrucciones para validar que todas estas herramientas estén instaladas y configuradas de forma correcta. Para el caso del manejador, la revisión consiste en aplicar algunas sentencias SQL como las siguientes.

Cabe mencionar que los ejemplos aquí mostrados pueden tener algunas diferencias que dependen del manejador empleado por el profesor.

- Creación de una Base de Datos

```
CREATE DATABASE <databasename>;
```

- Creación de un usuario (Sintaxis Oracle)

```
CREATE USER <username> IDENTIFIED BY <password>
```

**Actividades a realizar:**

1. Una vez que el entorno ha sido instalado, realizar las siguientes acciones:
  - Revisar que el manejador y las herramientas a emplear durante el semestre estén instaladas correctamente.
  - Revisar que la configuración del manejador sea la correcta. Esto incluye creación de usuarios y/o esquemas, ejecución de scripts iniciales.
  - Realizar un ejercicio para que el alumno comprenda la forma en la que se inicia el manejador seleccionado, realizar una conexión para validar que el ambiente funcione correctamente.
2. Uso de algún diagramador, por ejemplo: Dia
  - Explicar su objetivo y beneficios, dar a conocer la barra de herramientas y su objetivo, como guardar y abrir un diagrama ya elaborado.
3. Uso de una herramienta CASE, por ejemplo ErWin, Er-Studio, etc.
  - Dar a conocer los diferentes tipos de CASE y sus ventajas
  - Diferencia con respecto al diagrama entidad relación y modelo relacional
  - Navegar por la herramienta seleccionada, analizar un diagrama sencillo, mostrar un diagrama completo ya elaborado

## PRACTICA 2. Lenguaje de Control de Datos

### Objetivo:

El alumno aprenderá a utilizar comandos de control de datos para crear usuarios, otorgar y revocar permisos.

### Introducción

Las instrucciones para implementar el control de acceso a una Base de Datos relacional varían con respecto al manejador. En general la implementación del control de acceso a una base de datos está basada en la definición de una serie de objetos que tiene por objetivo el control de quién, cuándo y las acciones que puede realizar un usuario una vez que se le ha otorgado algún tipo de acceso a una Base de Datos.

A nivel general el control de acceso se realiza a través del concepto de privilegio. Un privilegio permite a un usuario realizar una acción muy específica. Entre mayor sea el número de privilegios que le sea otorgado a un usuario, mayor será su capacidad para realizar acciones dentro del manejador.

Típicamente, los manejadores definen usuarios administradores que se crean al momento de crear la base de datos. Estos usuarios son los encargados de otorgar privilegios a nuevos usuarios, e inclusive convertirlos en usuarios administradores.

Adicional a los usuarios administradores internos de la base de datos, existen usuarios administradores definidos a nivel del sistema operativo que son considerados como los dueños del software y todos los archivos asociados a la base de datos, incluyendo los datos de un usuario final.

Algunos ejemplos de privilegios son:

- CREATE SESSION
- CREATE TABLE
- CREATE VIEW
- CREATE SYNONYM
- CREATE PUBLIC SYNONYM
- CREATE PROCEDURE
- CREATE TRIGGER
- CREATE SEQUENCE
- CREATE USER

La sintaxis para crear un usuario varía entre manejadores. Ejemplo:

```
CREATE USER <username> IDENTIFIED
[BY <password> | EXTERNALLY | GLOBALLY AS <external_name>]
[DEFAULT TABLESPACE <tablespace>]
[TEMPORARY TABLESPACE <tablespace>]
[{QUOTA <num> [k | m] on <tablespace_name> | UNLIMITED ON
<tablespace_name>}...]
[PROFILE <profile_name>]
[PASSWORD EXPIRE]
[ACCOUNT [LOCK|UNLOCK]
```

Para otorgar un privilegio a un usuario típicamente se emplea una sintaxis similar a la siguiente:

```
GRANT <privilege>[,<privilege>,....]  
TO <username> [,<username>,...]  
[WITH ADMIN OPTION];
```

Otro concepto comúnmente empleado en este contexto es el concepto de Rol. Un rol permite agrupar a un conjunto de privilegios y ser asignado a uno o varios usuarios. Por ejemplo, el rol 'admin' puede estar formado por privilegios como 'create user', 'create table', etc., y puede ser asignado a varios usuarios.

Generalmente existen 2 tipos de privilegios:

- Privilegios a nivel sistema. Permiten realizar acciones que modifican la estructura de la base de datos.
- Privilegios a nivel objeto. Permiten el acceso de objetos que pertenecen a un usuario esquema.

#### **Actividades a realizar:**

1. Administración de usuarios: Creación, modificación y eliminación.
2. Generar ejercicios en el que se otorguen y eliminen privilegios a un usuario;
  - Privilegios de sistema, a nivel de la estructura de objetos
  - Privilegios a nivel de objeto,
3. Roles.
  - Revisión de roles pre existentes en la base de datos.
  - Creación de roles
  - Asignación de roles a usuarios.
4. Revisión de esquemas y funcionalidades adicionales relacionadas con seguridad para el manejador seleccionado, como son: administración de cuentas de usuario: contraseñas, vigencia y almacenamiento en el diccionario de datos.



### PRACTICA 3. Diseño de modelos básicos entidad relación con notación CHEN utilizando una herramienta CASE

#### Objetivo:

El alumno comprenderá y pondrá en práctica la elaboración de diagramas ER empleando notación CHEN y herramientas CASE para construir modelos de datos con sus elementos básicos: Identificación de entidades, atributos y relaciones.

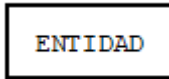
#### Introducción

El primer paso en el diseño de una base de datos es la producción del esquema conceptual para lograr una descripción de alto nivel de la realidad, luego se transforma en un esquema lógico.

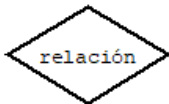
El modelo conceptual más utilizado para el diseño de bases de datos fue introducido por Peter Chen en 1976. Este modelo está formado por un conjunto de conceptos que permiten describir la realidad mediante un conjunto de representaciones gráficas y lingüísticas.

Simbología o nomenclatura básica según notación Chen:

- ▶ Entidad Es un objeto real o abstracto de interés, sobre el que se recoge información y se representa gráficamente mediante un rectángulo y su nombre aparece en el interior en mayúsculas. Un nombre de entidad sólo puede aparecer una vez en el esquema conceptual. Generalmente se expresa con sustantivos.



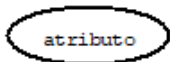
- ▶ Relación Es una asociación, vinculación o correspondencia entre entidades. Se representa gráficamente con un rombo etiquetado. Generalmente representadas por verbos.



Una Relación queda caracterizada por tres propiedades:

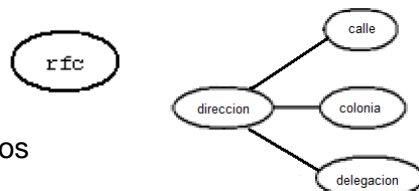
- Nombre. Debe de tener un nombre que la identifique unívocamente.
- Grado. Número de tipos de entidad sobre las que se realiza la asociación. Generalmente binaria.
- Tipo de Correspondencia. Número máximo de ejemplares de cada tipo de entidad que pueden intervenir en un ejemplar del tipo de relación. A esta propiedad también se le denomina *cardinalidad*.

- ▶ Atributo Es una propiedad o característica asociada a una determinada entidad o relación y por lo tanto común a todos los ejemplares. La representación gráfica utilizada es por medio de una elipse etiquetada en letra en minúsculas.

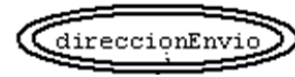


En función de las características respecto de la entidad que definen, se distinguen varios tipos de atributos:

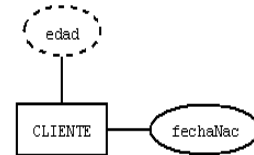
- Normal: Tiene un solo valor.
- Compuestos: Se dividen en otros atributos



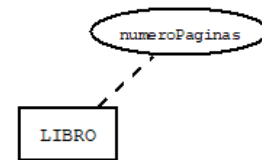
- Multivalorados: Tiene un conjunto de valores para una entidad concreta. Se representa con doble elipse.



- Derivados: Cuando un valor puede calcularse u obtenerse a partir de otro. Se representa con una elipse con línea discontinua.



- Opcionales: Son usados cuando es posible desconocer el valor del atributo para cierta entidad o no se tiene un valor aplicable. Se representa con un línea discontinua entre la entidad y el atributo

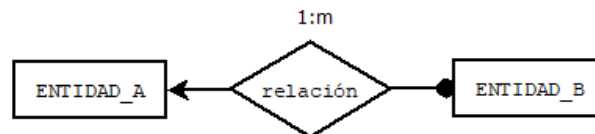


- Cardinalidad. Número de ejemplares de una entidad asociadas a otro ejemplar de una entidad o de la misma. Para una relación binaria (grado = 2), existen tres posibles tipos de correspondencia:

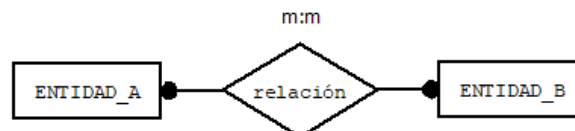
- Cardinalidad uno a uno. Un ejemplar de la Entidad A se asocia con un ejemplar de una Entidad B y viceversa



- Cardinalidad uno a muchos. Un ejemplar de la Entidad A se puede asociar con muchos ejemplares de una Entidad B. y un ejemplar de la Entidad B se asocia con un solo ejemplar de la Entidad A

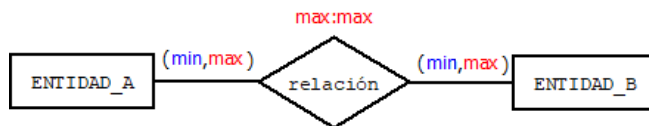


- Cardinalidad muchos a muchos. Un ejemplar de la Entidad A se puede asociar con muchos ejemplares de una Entidad B y viceversa.

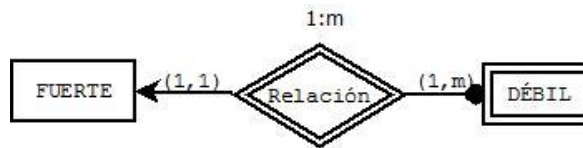


Cardinalidad mínima. Indica el número mínimo de asociaciones en las que aparecerá cada ejemplar de la entidad (el valor puede ser cero o uno)

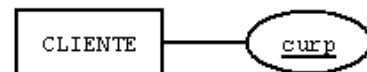
Cardinalidad máxima. Indica el número máximo de relaciones en las que puede aparecer cada ejemplar de la entidad (el valor puede ser uno o muchos)



Hay dos tipos de entidades: fuertes y débiles. Una entidad fuerte es una entidad que tiene existencia propia y tiene una clave primaria. Una entidad débil es una entidad que no tiene suficientes atributos para formar una clave primaria. Una entidad débil no puede existir sin su entidad fuerte. Se representa gráficamente por dos rectángulos concéntricos, la relación de asociación se representa con un doble rombo, su cardinalidad es 1:m.



- ▶ Clave primaria. Es un atributo o conjunto de atributos que identifican en forma única a una entidad. Se representa subrayando el nombre del atributo.



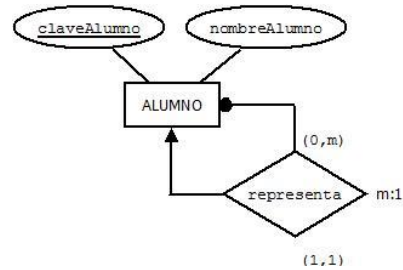
- ▶ Clave débil o discriminante. Es un atributo en una entidad débil que la identifica junto con la clave primaria de la entidad fuerte. Se representa subrayando en forma discontinua el atributo.



- ▶ Clave candidata o alternativa. Es un atributo que puede ser clave primaria, pero no fue elegida como tal. Sin embargo es importante considerarla.



- ▶ Relaciones Recursivas. Son relaciones unarias y, por lo tanto el tipo de relación sólo participa con un único tipo de entidad.

**Actividades a realizar:**

1. Enfatizar los elementos del diagrama Entidad Relación y su nomenclatura, para la estructura lógica de una base de datos en la herramienta CASE seleccionada.
2. Repasar por medio de dos textos diferentes los sustantivos y sus características, sus identificadores, su opcionalidad u obligatoriedad, y sus relaciones.
3. Identificar en cada texto entidades fuertes, débiles, recursivas, atributos simples, compuestos, calculados, multivalorados, etc.
4. Realizar el modelo Entidad Relación en la herramienta CASE seleccionada.

## PRACTICA 4. Diseño de modelos avanzados entidad relación con notación CHEN utilizando una herramienta CASE

### Objetivo:

El alumno reafirmará los conocimientos adquiridos para realizar la elaboración de diagramas Entidad Relación empleando notación CHEN y herramientas CASE para construir modelos de datos de mayor complejidad (modelo extendido).

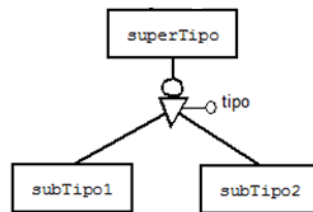
### Introducción

El modelo Entidad–Relación extendido incluye todos los conceptos del modelo Entidad–Relación básico e incorpora los conceptos de jerarquía de tipos con la generalización y especialización. Un conjunto de entidades puede incluir subgrupos de entidades que se diferencian de alguna forma de las otras entidades del conjunto. El modelo ER proporciona una forma de representación de estos grupos de entidades distintos.

**Generalización.** Es el resultado de la unión de 2 o más conjuntos de entidades (de bajo nivel) para producir un conjunto de entidades de más alto nivel.

La generalización se usa para hacer resaltar los parecidos entre tipos de entidades de nivel más bajo y ocultar sus diferencias.

Se deduce la existencia de una entidad de nivel superior llamada superTipo, a partir de otras de nivel inferior denominadas subTipo.



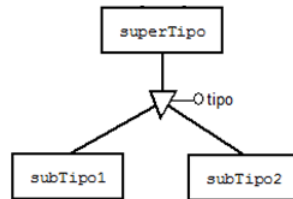
### Características

- Los atributos comunes a los subtipos se guardan en el supertipo
- Todos los ejemplares de los subtipos se encuentran en el supertipo
- Existe una cardinalidad entre el supertipo y cada uno de los subtipos 1:1

**Especialización.** Es el proceso de designación de subgrupos dentro de un conjunto de entidades, es el resultado de tomar un subconjunto de entidades de alto nivel para formar un conjunto de entidades de más bajo nivel.

Se deduce la existencia de entidades de nivel inferior llamadas subtipos, a partir de otra de nivel Superior denominada supertipo.

Un tipo de entidad tiene ciertos atributos que tienen sentido para unos ejemplares pero no para otros, por lo que es conveniente definir varios subtipos que contengan estos atributos, dejando los que son comunes en el supertipo.



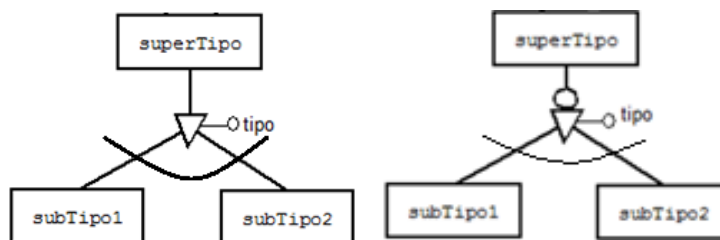
#### Características:

- Los atributos comunes a los subtipos se guardan en el supertipo
- Existen ejemplares en el Supertipo que no se encuentran en los subtipos
- Existen una cardinalidad entre el supertipo y cada uno de los subtipos 1:1

#### Restricción de las jerarquías

##### Exclusividad

Exige que cada entidad no pertenezca a más de un conjunto de entidades del nivel inferior, esto se simboliza con un arco en los subtipos.



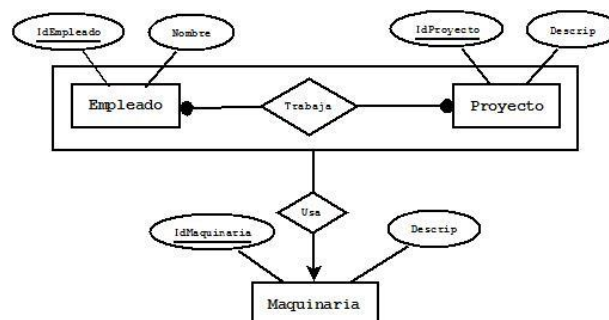
#### Agregación

Consiste en construir un nuevo tipo de entidad como composición de otros y su tipo de relación y así poder manejarlo en un nivel de abstracción mayor. La agregación ayuda a construir entidades de niveles superiores.

La agregación surge de la limitación que existe en el modelo ER, al no permitir expresar las relaciones entre relaciones.

#### Ejemplo

Podemos ver *Empleado-Trabaja-Proyecto* como una sola entidad, con lo cual se tiene lo siguiente:



**Actividades a realizar:**

1. Enfatizar los elementos extendidos del diagrama Entidad Relación y su nomenclatura, para la estructura lógica de una base de datos.
2. Repasar por medio de dos textos diferentes, arco (excluyente), subtipos e incluir texto con una relación excluyente (OR).
3. Realizar el diagrama Entidad – Relación de ambos textos en la herramienta CASE seleccionada.

## PRACTICA 5. Diseño básico de modelos relacionales

### Objetivo:

El alumno comprenderá e implementará la construcción de modelos de datos relacionales básicos empleando herramientas CASE a partir de un diagrama ER.

### Introducción

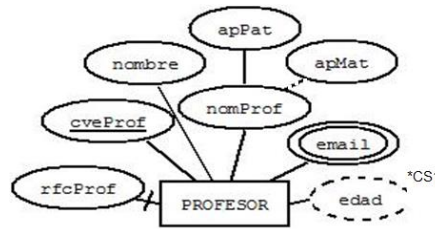
Para obtener un modelo relacional a partir de un modelo Entidad-Relación se tienen varias reglas

#### 1. Toda entidad fuerte se transforma en una relación

- Se conservan los atributos y la llave primaria
- En llaves candidatas se establece restricción de unicidad
- Atributos compuestos se colocan en forma individual
- Atributos multivalorados se crea una nueva relación propagando la llave primaria de la relación como foránea a la nueva relación.
- Atributos derivados o calculados se estable como atributo calculados
- Se establecen restricciones sobre atributos

#### Simbología a utilizar

- PRIMARY KEY (PK)
- UNIQUE (U)
- FOREIGN KEY (FK)
- CHECK (CK)
- Calculados (C)
- Discriminantes (D)
- No obligatorio (N)



#### Ejemplo:

PROFESOR= {cveProf (PK),nomProf,apPat,apMat(N),rfcProf(U),edad(C)\*CS1}

EMAILPROF= {cveProf (PK,FK),email(PK)}

\*CS1 Se obtiene del rfc del profesor

#### 2. Entidades Débiles

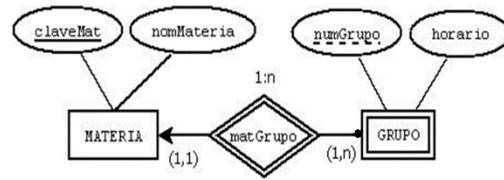
- Se crea la relación conservando todos sus atributos y se propaga la llave principal de la entidad fuerte de la que depende.
- La llave principal se formará con la llave primaria de la entidad fuerte y el discriminante de la entidad débil



Ejemplo:

MATERIA= {claveMat(PK), nomMateria}

GRUPO= {[claveMat(FK), numGrupo(D)](PK),  
horario}



### 3. Relaciones

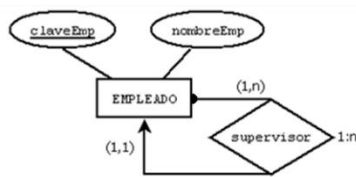
- 1:1 La llave primaria de una entidad se propaga a la otra entidad como llave foránea con la restricción de unicidad, dependiendo de la cardinalidad mínima, si la cardinalidad mínima es cero, se pasa el atributo de la entidad de cardinalidad uno, hacia la de cardinalidad cero, si en ambas es la misma cardinalidad, dependerá del contexto.
- m:1 ó 1:m La llave primaria de la entidad que tiene la cardinalidad uno se propaga a la relación de muchos como llave foránea.
- m: m Se genera una nueva relación que contendrá la llave primaria de cada una de las entidades que une, pero como llaves foráneas y juntas como llave primaria, además de sus atributos propios (descriptivos) de la relación.

Las relaciones (1:1, 1:m, m:1) que contengan atributos descriptivos, se tienen dos opciones

- Generar una nueva relación que contendrá la llave primaria de cada una de las entidades que une como llaves foráneas, además de sus atributos descriptivos, conservando la unicidad en la nueva relación.
- Propagar la llave primaria de la entidad uno junto con todos los atributos de la relación a la entidad m

### 4. Relaciones recursivas

Ejemplo:



La llave primaria de la entidad pasa como llave foránea de la misma entidad.

EMPLEADO= {claveEmp(PK), nombreEmp, claveSup(FK)}

### Modelos relacionales empleando herramientas CASE.



Existe una variedad de formatos y notaciones empleadas para generar modelos relacionales empleando herramientas CASE. La siguiente clasificación ofrece un panorama que comúnmente se emplea en la industria.

- Formato Relacional (Relational Format)
- \*\* Formato IE (International Engineering Format) / Formato de Martín (Martin's Format) / Modelo Pata de gallo (Crow's foot Format).

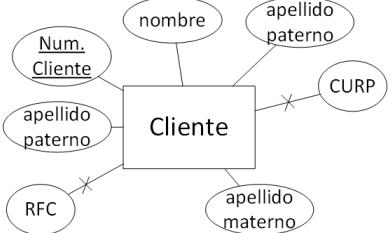
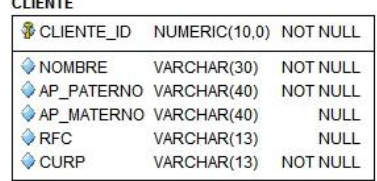
- \*\* Formato IDEF1X.
  - Originalmente desarrollado por “The Computer System Laboratory of the National Institute of Standards and Technology “en diciembre del 1993.

\*\* Estos formatos son los que comúnmente se emplean en la industria los cuales se revisan brevemente.

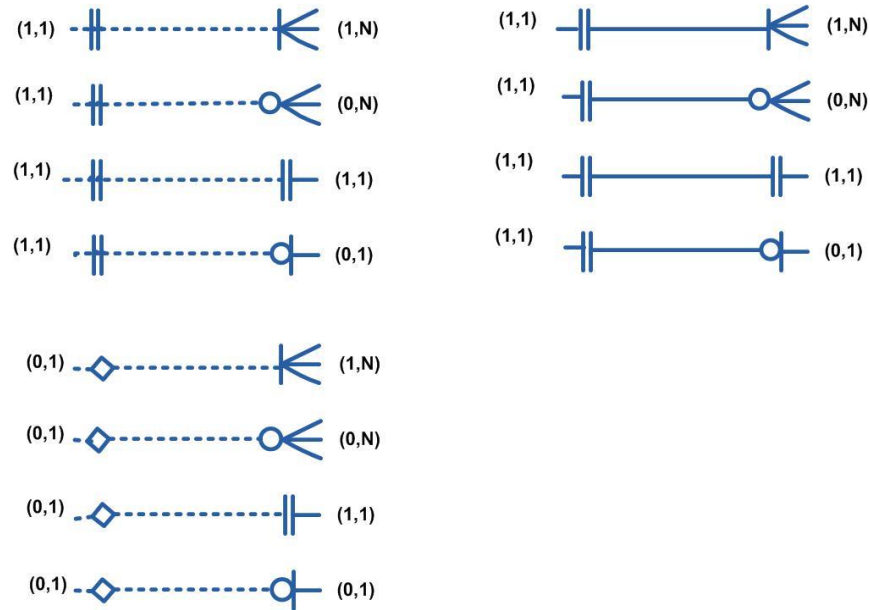
Representación de entidades.

Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X
<p>Entidad</p> 	<p>Tabla</p> 
<ul style="list-style-type: none"> <li>Se emplea un rectángulo con el nombre de la entidad en su interior iniciando en mayúsculas, sustantivo en singular</li> <li>Se emplea notación Camel Case para separar palabras.</li> </ul>	<ul style="list-style-type: none"> <li>El nombre se escribe arriba del rectángulo en mayúsculas.</li> <li>Si el nombre de la tabla es compuesto, se emplea “_” (guion bajo) para separar palabras.</li> <li>En el primer rectángulo se enlistan los atributos que formarán parte de la PK</li> <li>En el segundo rectángulo se enlistan los atributos que no forman parte de la PK.</li> </ul>

Representación general de atributos.

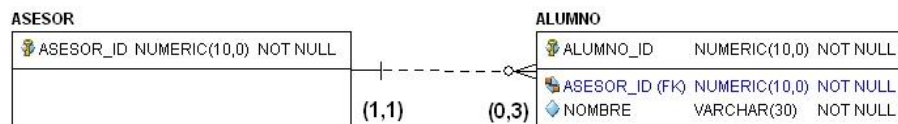
Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X
<p>Atributo</p> 	<p>Atributo</p> 
<ul style="list-style-type: none"> <li>Los atributos se representan por óvalos.</li> <li>El atributo que representa a la PK se subraya.</li> <li>Si existen llaves candidatas se marcan con una “X”</li> </ul>	<ul style="list-style-type: none"> <li>La PK aparece en el primer rectángulo, observar el ícono de una “Llave”.</li> <li>Al igual que el nombre, se emplea “_” para separar palabras, todo en mayúsculas.</li> <li>Por claridad es recomendable agregar el tipo de dato de cada atributo y el constraint NULL/NOT NULL.</li> <li>Por convención el nombre del atributo que representa a la PK es &lt;NOMBRE_TABLA&gt;_ID</li> </ul>

Representación de relaciones con cardinalidades Notación Crow's Foot.



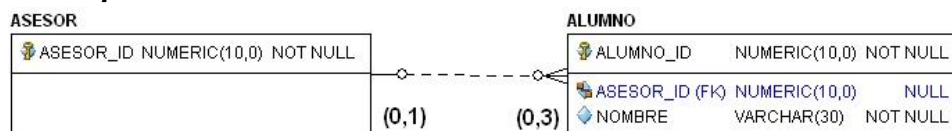
Ejemplo:

- Un profesor si lo desea, puede ser asesor de hasta 3 alumnos.
- Un alumno **debe** contar con su asesor y es uno solo durante su estancia en la escuela.



Suponer que se modifica la regla anterior:

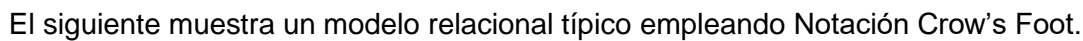
- Un profesor si lo desea puede ser asesor de hasta 3 alumnos.
- Un alumno **puede** solicitar a un único asesor durante su estancia en la escuela.



Observar las diferencias en los 2 diagramas anteriores:

- La cardinalidad de ALUMNO hacia ASESOR es (0,1), ya que el alumno puede o no tener asesor.
- Observar que en este caso, la FK debe ser declarada como NULL. De lo anterior, siempre que en valor mínimo de la cardinalidad sea 0, la FK se declara como opcional (NULL).

Representación de relaciones con cardinalidades, notación IDEF1X



**Actividades a realizar:**

1. Repasar la transformación de los elementos básicos del modelo Entidad Relación a su representación en el modelo relacional
2. A partir de los diagramas Entidad Relación realizados en las prácticas anteriores, realizar la transformación al modelo relacional generando los objetos necesarios en la herramienta CASE elegida.
3. Realizar la revisión de las propuestas de modelos, analizar y discutir posibles errores, ventajas y desventajas. Partir de un modelo ER con errores, corregir y generar el modelo relacional correcto.

## PRACTICA 6. Diseño avanzado de modelos relacionales

### Objetivo:

El alumno comprenderá e implementará la construcción de modelos de datos relacionales avanzados empleando herramientas CASE a partir de un diagrama ER extendido.

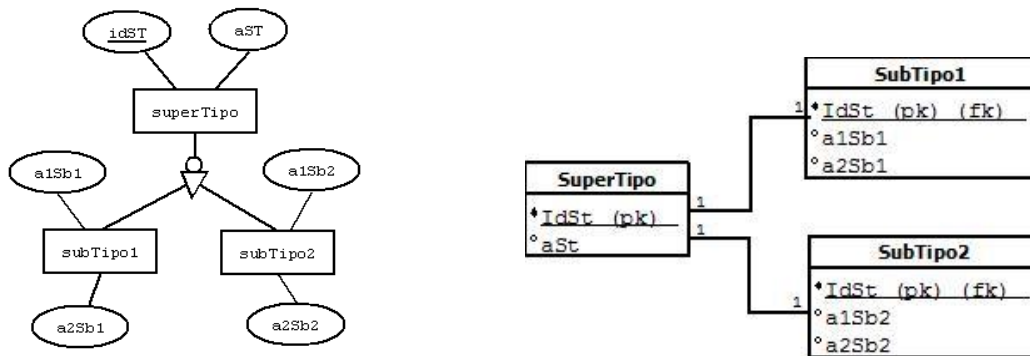
### Introducción

Para obtener un modelo relacional a partir de un modelo Entidad-Relación extendido se tienen varias opciones según el contexto

#### Jerarquía de tipos

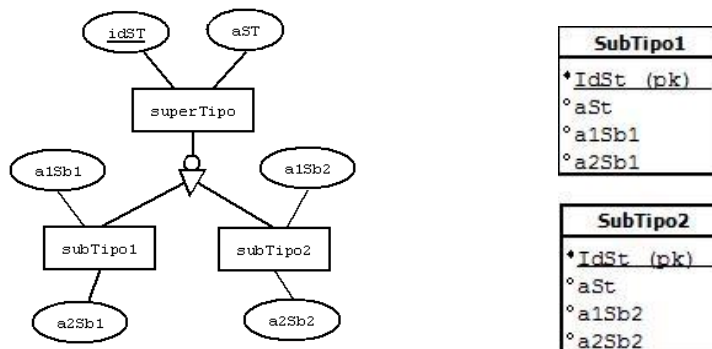
#### Caso 1 **parcial**

- Se crea una relación para el supertipo.
- Se crean una relación para cada uno de los subtipos, propagándose la llave primaria del supertipo a cada uno de los subtipos como llave foránea y este a su vez es llave primaria.

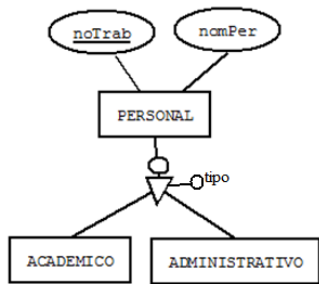


#### Caso 2 **total**

- Se propagan los atributos del supertipo a los subtipos.
- Se crean una relación para cada uno de los subtipos, la llave primaria del supertipo será la llave primaria de cada uno de los subtipos.



## Caso 3



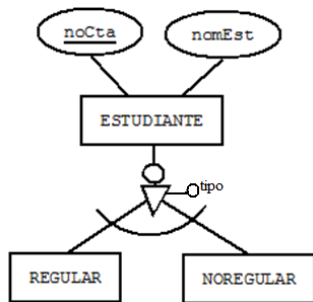
Generalización. No excluyente y sin atributos en los subtipos

PERSONAL= {noTrab(PK), nomPer}

TIPOPERSONAL= {noTrab (FK), tipoPersonal(CK)}

\* Ck verifica los valores de ACADEMICO o ADMINISTRATIVO

## Caso 4

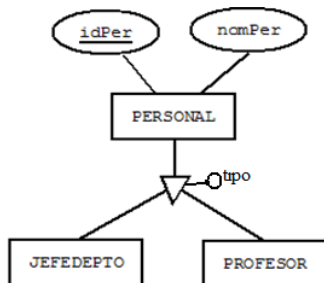


Generalización. Excluyente y sin atributos en los subtipos

ESTUDIANTE={noCta(PK), nomEst, tipoEstud(CK)}

\* Ck verifica los valores de REGULAR o NOREGULAR

## Caso 5



Especialización No excluyente y sin atributos en los subtipos

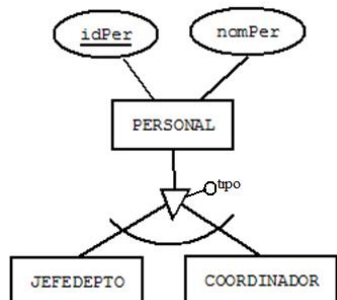
PERSONAL={idPer(PK), nomPer, ...}

TIPO={idTipo(PK), descripTipo}

TIPOPERSONAL={idPer(FK), idTipo(FK)}

Habrà PK del supertipo que no tengan asignado ningún tipo.

## Caso 6



Especialización. Excluyente y sin atributos en los subtipos

PERSONAL={idPer(PK), nomPer, idTipo(FK)}

TIPOPERSONAL={idTipo(PK), descripTipo}

La FK en el supertipo podrá tener valores nulos



## Modelo relacional empleando herramientas CASE.

### Relaciones Supertipo – Subtipo.

Existen 4 variantes para representar la cardinalidad entre un supertipo y su subtipo, las cuales se expresan a través de 2 tipos de restricciones:

- Restricción Excluyente/traslape (disjoint / overlapping).

Diseño conceptual	Diseño lógico
<b>Excluyente (Disjoint):</b> Una persona solo puede tener un rol: profesor, administrador o administrativo	
<ul style="list-style-type: none"> <li>Se emplea el arco para indicar una relación excluyente.</li> </ul>	<ul style="list-style-type: none"> <li>En IDEF1X Se emplea un círculo y al interior una letra “D” para indicar una relación excluyente (“Disjoint”).</li> <li>En Crow’s foot se emplea un semi-círculo con una cruz al centro :</li> </ul>

- Excluyente:** Cada instancia del supertipo se asocia a lo más con una instancia de alguno de sus subtipos.
  - Traslape:** Cada instancia del supertipo puede asociarse con más de una instancia de sus subtipos
- Restricciones Parciales o totales (partial / complete).
  - Total:** Cada instancia del supertipo se asocia por lo menos con una instancia de sus subtipos.



- **Parcial:** Cada instancia del supertipo puede o no asociarse con las instancias de sus subtipos.

**Actividades a realizar:**

1. Repasar la transformación de los elementos extendidos del modelo Entidad Relación a su representación en el modelo relacional
2. A partir de los diagramas Entidad Relación extendido realizados en prácticas anteriores, realizar la transformación al modelo relacional avanzado generando los objetos necesarios en la herramienta CASE seleccionada. Los ejercicios debe incluir jerarquías (supertipo - subtipos) con todas las posibles variantes.
3. Realizar la revisión de las propuestas de modelos, analizar y discutir posibles errores, ventajas y desventajas. Partir de un modelo ER con errores, corregir y generar el modelo relacional correcto.
4. A partir de una tabla de datos proporcionada, normalizar hasta la tercera forma normal y crear el modelo relacional resultante en la herramienta CASE seleccionada, opcionalmente aplicar FNBC y cuarta forma normal.

## **PRACTICA 7. Lenguaje de definición de datos.**

### **Objetivo:**

El alumno aprenderá el uso del lenguaje SQL para la creación de tablas e índices en una base de datos. Comprenderá y confirmará el uso del lenguaje SQL para realizar la modificación y/o eliminación de la estructura de los principales objetos de una base de datos empleando las instrucciones ALTER y DROP.

### **Introducción**

#### **Lenguaje de definición de datos**

El Lenguaje de Definición de Datos (DDL Data Definition Language) nos permite realizar las tareas de creación, eliminación, modificación de un objeto de base de datos, además de la concesión y revocación de privilegios sobre estos objetos. Los ejemplos más comunes de comandos de este tipo son: CREATE, ALTER y DROP.

La primera fase de cualquier base de datos siempre comienza con órdenes DDL, para diseñar los objetos de la base de datos.

#### **Diseño de tablas**

Cuando diseñe una base de datos, deberá decidir qué tablas necesita, qué tipo de datos van destinados a cada tabla, quién puede tener acceso a cada tabla, etc. Cuando cree tablas y trabaje con ellas, seguirá tomando decisiones más específicas acerca de las mismas.

El método más eficiente para crear una tabla consiste en definir todo lo que se necesita en la tabla al mismo tiempo, incluidas las restricciones para los datos y los componentes adicionales. Entre las decisiones que deben tomarse, se incluyen:

- Los tipos de datos que debe contener la tabla.
- Las columnas de la tabla y los tipos de datos para cada columna (así como su longitud, si es preciso).
- Qué columnas aceptan valores NULL.
- Si deben utilizarse (y cuándo) restricciones o valores predeterminados y reglas.
- Los tipos de índices necesarios, dónde se necesitan y qué columnas son claves principales y claves externas.

## Creación de una tabla

La sintaxis para crear tablas en su forma más común es la siguiente<sup>1</sup>:

```
CREATE TABLE <nombre tabla>
(<nombre columna> <tipo de dato>
[NOT NULL] [UNIQUE] [CONSTRAINT <nombre restricción>] [PRIMARY
KEY] [DEFAULT] [CHECK]
| [PRIMARY KEY (<lista columnas>)]
| [FOREIGN KEY (<lista columnas>) REFERENCES <nombre tabla>]
| [UNIQUE (<lista columnas>)] [CONSTRAINT <nombre
restricción>], [... ] )
| [CHECK (condición de búsqueda)])
;
```

Se explican a continuación cada uno de los comandos:

- La sentencia CREATE TABLE se utiliza para crear una tabla.
- Dentro de la tabla podemos especificar una serie de columnas que contienen datos y restricciones que verifican datos y especifican otras características de la tabla.
- Cada columna debe tener asignado un tipo de dato válido.
- Puede especificarse NOT NULL para asegurar que la columna siempre tenga datos.
- También podemos especificar un valor por omisión para la columna que va a utilizarse cuando se inserte un nuevo registro en la tabla y no se especifique un valor, o se especifique como DEFAULT.
- Podemos definir una llave primaria (PRIMARY KEY) para la tabla o definir una columna particular o un conjunto de columnas que sean UNIQUE (que sus valores no se puedan repetir en la tabla). Además, podemos definir una llave foránea (FOREIGN KEY) tomando en cuenta, que el valor de la <lista de columnas> debe existir en otra tabla.

## Eliminar una tabla

La sentencia DROP suprime todas las restricciones de integridad referencial que hagan referencia a llaves de la tabla eliminada.

```
DROP TABLE <nombre tabla> [CASCADE CONSTRAINTS]
```

---

<sup>1</sup> < > Los corchetes triangulares deben ser reemplazados por el nombre del objeto correspondiente.

( ) Los paréntesis son elementos propios de la sintaxis.

[ ] Los corchetes cuadrados indican argumentos opcionales.

| La barra vertical separa distintas alternativas.

... Los puntos suspensivos indican repetición del elemento sintáctico precedente.

## Modificar una tabla

La sentencia ALTER TABLE se utiliza para modificar la estructura de una tabla existente. Dentro de la tabla podemos añadir (ADD) o borrar (DROP) columnas, restricciones, condiciones de comprobación, llaves primarias y foráneas.

### Actividades a realizar:

Con base en un modelo relacional proporcionado que contenga llaves primarias compuestas y relaciones dependientes.

1. Crear las tablas, considerando las diferentes cardinalidades (1:1, 1:m, m:m)
2. Restricciones de llave primaria, foránea, unicidad, valores obligatorios y de dominio. Contemplar el uso de valores por omisión y nulos. Considerar el uso de creación de restricciones tanto a nivel columna como a nivel tabla.
3. Generar índices (non unique, unique)
  - a. Explicar y analizar los casos donde es recomendable crear índices, por ejemplo, en llaves foráneas, atributos que se emplean con alta frecuencia en búsquedas, etc.
4. Mostrar la estructura de las tablas.
5. A partir de un caso de estudio nuevo o existente proponer una serie de cambios como:
  - Aumentar el tamaño de una columna.
  - Modificar un tipo de dato numérico a varchar conservando la misma longitud
  - Modificar una columna nula a obligatoria.
6. Utilizando el comando ALTER
  - Adicionar tres constraints a una tabla (Primary Key, Foreign key y unique)
  - Inhabilitar el constraint foreign key
  - Modificar objetos autoincrementables
  - Modificar un índice para deshabilitarlo, renombrarlo y reconstruirlo
7. Realizar operaciones de borrado como:
  - Eliminar una columna
  - Eliminar el constraint unique,
  - Eliminar una tabla
8. Realizar operaciones restringidas, en cascada y puesta en nulo
  - UPDATE
  - DELETE

## PRACTICA 8. Lenguaje de manipulación de datos y transacciones

### Objetivo:

El alumno comprenderá y confirmará el uso del lenguaje SQL para realizar la administración de los datos empleando las instrucciones INSERT, UPDATE y DELETE. Comprenderá las principales características de una transacción, comprobará y configurará los diferentes niveles de aislamiento empleados por un RDBMS.

### Introducción

#### Lenguaje de manipulación de datos

El lenguaje de manipulación de datos (DML) nos permite acceder, crear, modificar o eliminar los datos de un esquema de base de datos. Las órdenes que se utilizan son: insertar (INSERT), actualizar (UPDATE), borrar (DELETE) y seleccionar (SELECT) los datos en una base de datos.

Insertar valores a una tabla

#### INSERT

Una vez que se han creado las tablas, utilizamos el comando INSERT para agregar información a las tablas, su sintaxis es la siguiente:

```
INSERT INTO <nombreTabla> [(campo1 [,campo2, ...])]
VALUES (valor1, valor2,...);
```

O bien,

```
INSERT INTO <nombreTabla>
VALUES (valor1, valor2,...);
```

En esta opción debemos tener cuidado de colocar los valores según el orden correspondiente en la tabla.

Los valores de tipo DATE o CHAR se deben encerrar entre comillas simples. Los valores insertados deben ser de un tipo compatible con el de las columnas de la tabla.

También se puede realizar un INSERT combinado con un SELECT

```
INSERT INTO <nombreTabla> [(campo1 [,campo2, ...])]
SELECT <columna1, columna2, ...>
FROM <nombreTabla>
```

Modificar datos de una tabla

#### UPDATE

Se emplea para modificar los datos existentes en una tabla, su sintaxis es la siguiente:

```
UPDATE <nombre tabla> SET
{<columna> = <expresión> [, ...]}
```

```
| {( <lista-de columnas> ) | *} = ( <lista de expresiones> ) }
[WHERE <condición>]
```

Una expresión puede estar formada por una subconsulta SELECT entre paréntesis, cuyo resultado es un único dato de una sola columna (un único valor simple). La lista de columnas está formada por aquellas columnas a las cuales se les modificará su valor. Si se omite la cláusula WHERE entonces se actualizan todos los datos de la tabla.

Eliminar datos de una tabla

## DELETE

Permite eliminar datos de una tabla. Su sintaxis es la siguiente:

```
DELETE [FROM] <nombre tabla> [WHERE <condición>]
```

Si no se pone condición, se borran todos los datos de la tabla.

Seleccionar datos de una tabla

## SELECT

Para recuperar la información guardada en una base de datos utilizamos el comando SELECT.

La sintaxis de la orden SELECT consta básicamente de las cláusulas SELECT y FROM como obligatorias y la cláusula WHERE como opcional, su sintaxis es la siguiente:

```
SELECT [ALL | DISTINCT | UNIQUE ]
[<tabla|aliasTabla>.] <columna1> [AS <nombreColumna>]
[, [<tabla|aliasTabla>.]< columna2> [AS <nombreColumna>]..., |*]
FROM [<nombreBasedeDatos>] Tabla [<aliasTabla>]
[WHERE <condición>]
```

Las columnas ambiguas (iguales) se preceden del nombre de la tabla: <tabla>.<columna> o de un sobrenombre corto para la tabla <aliasTabla>.<columna>

AS Se usa para cambiar el nombre de una columna del conjunto de resultados o para asignarle un nombre a una columna derivada.

El \* hace referencia a todas las columnas de la tabla.

Cláusula FROM

Permite indicar las tablas que contienen los datos.

FROM {<nombre de tabla> [alias de tabla] } [,...]

<alias de tabla>: Es un sinónimo activo sólo en el ámbito de la sentencia SELECT.

#### Cláusula WHERE

Sirve para indicar la condición que deben cumplir las filas resultantes.

WHERE <condición>

Una condición está formada por una o varias expresiones condicionales conectadas por los operadores lógicos AND, OR y NOT. Una expresión condicional tiene la forma siguiente:

Expresión de comparación

<expresión1> <operador relacional> <expresión2>

Verifica si las dos expresiones satisfacen la comparación.

#### Actividades a realizar:

1. Repasar las instrucciones insert, update y delete
2. A partir de un modelo relacional proporcionado y una serie de requerimientos realizar lo siguiente:
  - Inserciones de renglones con y sin encabezados de columnas
  - Inserción con parámetros
  - Inserción con selección
  - Actualizar dos columnas en un comando
  - Actualizar una columna con una condición
  - Eliminar todos los renglones, utilizar opcionalmente TRUNCATE
  - Eliminar un renglón utilizando una condición
3. Ejecutar un conjunto de instrucciones para confirmar el comportamiento transaccional empleando las instrucciones COMMIT, ROLLBACK y puntos de salvado.
4. Diseñar ejercicios que le permitan al alumno comprobar las propiedades ACID de las transacciones.
5. Diseñar ejercicios que le permitan al alumno comprobar los diferentes niveles de aislamiento que ofrecen los RDBMS.

## PRACTICA 9. Álgebra relacional

### Objetivo:

El alumno conocerá los operadores del álgebra relacional correspondientes a la unión, intersección, diferencia y producto cartesiano. Aplicará el uso de la instrucción SELECT en su forma básica para realizar consulta de datos. Usará alias.

### Introducción

La manera en que los usuarios solicitan información de la base de datos es a través del lenguaje de consultas, que es un lenguaje de nivel superior a los de los lenguajes de programación, pues a través del menor número de instrucciones se puede obtener la información solicitada. Dentro del lenguaje de consulta se encuentra el lenguaje procedimental, también llamado álgebra relacional, el cual consiste en un conjunto de operaciones que toman una o dos relaciones como entrada y generan otra relación nueva como resultado.

Dentro del álgebra relacional tenemos dos tipos de operadores:

- Operadores básicos o fundamentales que son selección, proyección, unión, diferencia y producto cartesiano.
- Operadores compuestos o derivados: intersección y unión natural (join).

### OPERADORES BÁSICOS

Estos operadores son utilizados por cualquier manejador de bases de datos para la obtención de consultas y se encuentran dentro del lenguaje de manipulación de datos.

#### Selección

Extrae las tuplas específicas de una relación dada presentando sólo aquellas que satisfagan las condiciones específicas.

```
SELECT <columna>  
FROM tabla  
WHERE <condición>;
```

#### Proyección

Extrae los atributos especificados de una relación dada.

```
SELECT <columna>  
FROM tabla;
```

#### Unión

Construye una relación formada por todas las tuplas que aparecen en las dos relaciones especificadas sin repeticiones.



```
SELECT <columna>
FROM tabla1
UNION
SELECT <columna>
FROM tabla2;
```

## Diferencia

Construye una relación formada por todas las tuplas de la primera relación que no aparezcan en la segunda, de las dos relaciones especificadas. EXCEPT o MINUS devuelve los valores de la primera consulta que no se encuentran en la segunda. Así podemos averiguar qué registros están en una consulta pero no en la otra, calculando la diferencia entre dos conjuntos de registros

```
SELECT <columna>
FROM tabla1
MINUS
SELECT <columna>
FROM tabla2;
```

## Producto cartesiano

Combina cada dato de una tabla con cada dato de otra tabla.

```
SELECT <columnaA>,<columnaB>
FROM tabla1, tabla2;
```

## OPERADORES COMPUESTOS

Estos operadores simplifican las consultas habituales, pues existe una operación equivalente usando sólo los operadores básicos.

## Intersección

De dos relaciones específicas se obtienen las tuplas que coinciden en ambas.

```
SELECT <columna>
FROM tabla1
INTERSECT
SELECT <columna>
FROM tabla2;
```

## JOIN

Es usado para consultar datos de más de una tabla. Los datos se combinan relacionando valores comunes, típicamente valores de primary key y foreign key. Su combinación está

basada en todas las columnas comunes entre ambas tablas. Si dos tablas en una consulta no tienen ninguna condición de combinación, entonces SE devuelve su producto cartesiano.

<nombre tabla> JOIN <nombre tabla> ON <condición>

**Actividades a realizar:**

1. A partir de un script de una base de datos ejemplo proporcionada y un conjunto de consultas expresadas con álgebra relacional, transformarlas al lenguaje SQL y ejecutar cada consulta en el manejador para confirmar su resultado.
2. Realizar consultas utilizando teoría de conjuntos.

## PRACTICA 10. Consultas básicas en SQL y funciones de agregación

### Objetivo:

El alumno utilizará instrucciones como DISTINCT, ORDER BY, BETWEEN, LIKE, IN. Hará uso de funciones de agregación empleando las instrucciones GROUP BY y HAVING.

### Introducción

SQL permite realizar consultas con condiciones de búsqueda, de tal forma que los resultados son precisamente aquellos que cumplen con dichas condiciones. También podemos ordenarlos y agruparlos según se desee.

### CONSULTAS CON CONDICIONES DE BÚSQUEDA

Las condiciones de búsqueda se agrupan en distintos contrastes: comparación, rango, correspondencia con patrón, de valor nulo y compuestos.

- Comparación

Calcula y compara los valores de dos expresiones SQL, utilizando los operadores de comparación (=, <>, <,>, <=,>=).

EJEMPLO 1. Obtener información de los departamentos que se encuentran ubicados en la ciudad de NEW YORK:

```
SELECT deptno, dname
FROM dept
WHERE loc = 'NEW YORK';
```

- Rango

### Cláusula BETWEEN...AND

Comprueba si un valor se encuentra entre dos valores especificados.

EJEMPLO 2. Nombre de los empleados y el puesto de aquellos que tienen un salario entre 1200 y 3000

```
SELECT ename, job
FROM emp
WHERE sal BETWEEN 1200 AND 3000;
```

- Pertenencia a un conjunto

### Cláusula IN

Comprueba si un valor dado coincide con uno de una lista de valores especificada.

EJEMPLO 3. Nombre de los empleados y el puesto de aquellos que ganan 2975, 3000 o 5000.

```
SELECT ename, job
FROM emp
WHERE sal IN (2975, 3000, 5000);
```

- Correspondencia con patrón

#### Cláusula LIKE

Obtiene las filas para las que el contenido de una columna de texto coincide con un patrón especificado. En este tipo de consulta se utiliza el símbolo % como comodín. También se utiliza el símbolo subrayado bajo (\_), que representa exactamente una posición de carácter, sólo un carácter.

#### EJEMPLO 4

- a) Liste la clave y nombre del empleado, de aquellos cuyo puesto inicie con la letra S

```
SELECT empno, ename
FROM emp
WHERE job LIKE 'S%';
```

- b) Muestre todos los datos de los empleados donde en la tercera posición de su nombre tengan una letra A

```
SELECT *
FROM emp
WHERE ename LIKE '__A%';
```

- Valor nulo

#### Cláusula IS NULL o IS NOT NULL

Comprueba si existen valores nulos en las columnas.

EJEMPLO 5. Muestre la clave de los empleados sin comisión.

```
SELECT empno
FROM emp
WHERE comm IS NULL;
```

- Compuestos

#### Cláusulas AND, OR y NOT

Utilizando las reglas de la lógica se pueden combinar condiciones de búsqueda SQL simples para formar condiciones compuestas únicas.

EJEMPLO 6. Liste el nombre de los empleados que son administradores o analistas y su sueldo es mayor a \$2500

```
SELECT ename
FROM emp
WHERE (job='MANAGER'
      OR job='ANALYST')
      AND sal>2500;
```

## CONSULTAS CON CONDICIONES DE ORDENAMIENTO

### Cláusula ORDER BY

Ordena los registros resultantes de una consulta por un campo o unos campos especificados en orden ascendente o descendente. Su sintaxis es la siguiente:

```
SELECT <lista de campos>
FROM <tabla>
WHERE <criterios de selección >
[ORDER BY <campo1 [as|des], [,campo2 [as|des]]];
```

EJEMPLO 7. Nombre de los empleados y puesto, ordenados por este último.

```
SELECT ename, job
FROM emp
ORDER BY job;
```

Nombre de los empleados ordenados alfabéticamente en forma descendente.

```
SELECT ename
FROM emp
ORDER BY ename DESC;
```

## CONSULTAS CON FUNCIONES DE AGREGACIÓN

Las funciones de agregación generan valores de resumen en las consultas; procesan todos los valores seleccionados en una única columna para generar un único resultado. Estas funciones se pueden aplicar a todas las filas de una tabla, a un conjunto de una tabla especificado por una cláusula WHERE o a uno o varios grupos de filas de una tabla. Cuando se aplica una función de agregación se genera un valor individual por cada conjunto de filas.

Las funciones de agregación son las siguientes:

Función	Acción
COUNT(*)	Regresa el número de registros encontrados
COUNT(<campo>)	Regresa el número de registros cuyo valor del campo especificado no es nulo
SUM(<campo>)	Suma los valores de la columna especificada
AVG(<campo>)	Promedia los valores del campo especificado
MIN(<campo>)	Regresa el valor mínimo del campo especificado
MAX(<campo>)	Regresa el valor máximo del campo especificado

## EJEMPLO 8

a) Cantidad de empleados que son administradores.

```
SELECT COUNT(empno)
FROM emp
WHERE job='MANAGER';
```

b) Cantidad de empleados que reciben comisión.

```
SELECT COUNT(comm)
FROM emp;
```

c) El salario máximo pagado a un empleado.

```
SELECT MAX(sal)
FROM emp;
```

## CONSULTAS CON FUNCIONES DE AGRUPACIÓN

## Cláusula GROUP BY

En la cláusula GROUP BY se indica el o los campos por los cuales se desea agrupar un conjunto de registros. Comúnmente esta agrupación va acompañada con una serie de funciones agregación.

## EJEMPLO 9.

a) Clave del departamento y el monto total empleado para pagar a los empleados de dicho departamento.

```
SELECT deptno, SUM(sal)
FROM emp
GROUP BY deptno;
```

b) Sueldo promedio por cada clave de departamento.

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno;
```

c) Salario mínimo pagado por puesto.

```
SELECT job, MIN(sal)
FROM emp
GROUP BY job;
```

### Cláusula GROUP BY...HAVING

Esta cláusula es el equivalente a la cláusula WHERE, es decir, especifica un criterio o condición, pero la diferencia radica en que se ocupa únicamente cuando se desea especificar una función de agregación en la condición.

EJEMPLO 10. Clave de departamento y sueldo promedio de sus empleados, de aquellos departamentos cuyo salario promedio sea mayor que \$2000.

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
HAVING AVG(sal)>2000;
```

### Actividades a realizar:

A partir del script de una base de datos ejemplo y el modelo relacional proporcionado realizar

1. Consultas en las que se pida el uso de la instrucción SELECT involucrando las siguientes cláusulas: DISTINCT, ORDER BY, BETWEEN, LIKE, IN, ALL, AND, ANY, SOME, EXISTS, NOT, OR.
2. Consultas que hagan uso de funciones de agregación como son SUM, AVG, MAX, MIN, etc.
3. Tres consultas en las que se involucren las sentencias GROUP BY y HAVING, generar conjuntos de datos agrupados por más de una columna.

## PRACTICA 11. Utilización de distintos tipos de JOIN, subconsultas y vistas

### Objetivo:

El alumno pondrá en práctica el uso de los distintos tipos de JOIN, realizará diferentes tipos de subconsultas y creará vistas.

### Introducción

En SQL se tienen distintos tipos de JOINS disponibles.

- **INNER JOIN:** es el caso de unión interna clásico, combina dos o más tablas descartando todas las filas resultados que no se correspondan en ambas.
- **LEFT OUTER JOIN (o LEFT JOIN):** combina dos tablas con la diferencia que selecciona todas las filas de la primer tabla sin importar tengan o no coincidencia en la segunda. **RIGHT OUTER JOIN (o RIGHT OUTER):** es la opción inversa a la anterior. En la mayoría de las bases de datos actuales se puede omitir OUTER.
- **FULL OUTER JOIN:** combina los resultados de dos o más tablas, tengan o no coincidencia entre sí.
- **CROSS JOIN:** retorna el producto cartesiano de dos o más tablas, es decir, combina cada fila de una tabla con cada fila de otra tabla. CROSS JOIN no debería llevar la cláusula ON.
- **NATURAL JOIN:** Es un caso especial de INNER JOIN que compara por cuenta propia la equivalencia de columnas con el mismo nombre y tipo de dato entre dos o más tablas para hacer la combinación de ambas. Hay que tener especial cuidado con su uso, pues puede producir resultados ambiguos o generar problemas si se añaden, quitan, o renombran las columnas.

Una vista es una tabla cuyo contenido está definido por una consulta. Al igual que una tabla base, una vista consta de un conjunto de atributos y tuplas con un nombre; sin embargo, la vista no existe como conjunto de valores de datos almacenados en una base de datos. Sus atributos y tuplas provienen de una o varias tablas a las que se les hace referencia cuando se crea la vista y ésta se produce de forma dinámica cuando se hace referencia a ella.

Las vistas suelen utilizarse para restringir el acceso de usuarios a datos concretos de una tabla; además, se emplean para centrar, simplificar y personalizar la percepción de la base de datos para cada usuario. También pueden usarse como mecanismos de seguridad. Para crear vistas se utiliza la sentencia CREATE VIEW.

### Actividades a realizar:

1. A partir del script proporcionado de una base de datos ejemplo, realice subconsultas con distintos tipos de operadores en la cláusula SELECT, FROM, WHERE, HAVING y JOIN, además de utilizar funciones de agregación.
2. Consultas que ejemplifiquen los distintos tipos de JOIN por ejemplo: INNER JOIN, NATURAL JOIN y la cláusula USING; LEFT, RIGHT y FULL OUTER JOIN; CROSS JOIN, SELF JOIN.
3. Diferentes tipos de vistas empleando las consultas anteriores.



## PRACTICA 12. Programación con SQL. Parte 1

### Objetivo:

El alumno comprenderá los principales conceptos así como la estructura básica para realizar la construcción de scripts SQL empleando las extensiones de programación SQL ofrecidas por los manejadores. Aplicará los conceptos referentes a disparadores para reflejar la integridad de la información y hará uso de funciones creadas por el usuario

### Introducción

El lenguaje que se emplea para programar varía de un Sistema Manejador de Bases de Datos Relacional (RDBMS) a otro, este lenguaje de programación se usa para acceder y trabajar con bases de datos en desde distintos entornos.

Se crean subprogramas también llamados función, procedimiento o disparador.

Dentro del lenguaje de programación se puede hacer uso de funciones numéricas, de carácter, de fecha, de conversión de tipos de datos, excepto de agrupamiento (ya que éstas se aplican sobre una columna de una tabla)

Los operadores son los mismos que para SQL: aritméticos, lógicos, concatenación, y paréntesis.

También se utilizan estructuras de control, éstas permiten elegir la forma en la que se van a ejecutar las diferentes instrucciones dentro del programa. Las más importantes son las estructuras condicionales y las estructuras de repetición también llamadas ciclos.

Un disparador de base de datos es un procedimiento almacenado que se asocia con una tabla específica y que es llamado cuando un evento particular ocurre como una inserción, borrado o actualización. Se pueden usar disparadores para personalizar la reacción de un servidor de la base de datos. Los disparadores suelen utilizarse para restricciones de integridad complejas, auditoría de la información de una tabla o aviso a otros programas para ejecutar una acción. Además, tienen la restricción de que no pueden manejar parámetros ni ser llamados directamente.

Los disparadores no pueden contener órdenes SQL de control de transacciones (COMMIT, ROLLBACK o SAVEPOINT) y ningún procedimiento o función llamado por el disparador puede emitir órdenes de control de transacciones. Para crear un disparador se usa el comando CREATE TRIGGER.

Una definición de disparador incluye las siguientes partes únicas:

- Una lista de sentencias de disparador, incluyendo INSERT, UPDATE y/o DELETE, que llama el disparador. Un disparador está asociado con una y sólo una tabla.

- Un disparador puede configurarse para que se ejecute antes o después de realizarse una acción sobre la tabla asociada, para mantener la lógica de la aplicación

**Actividades a realizar:**

1. Creación de la estructura de un programa con las extensiones SQL del manejador utilizado.
2. Manejo de variables
3. Estructuras condicionales (if then, if then else, case, etc)
4. Estructuras de repetición (for, while, loop, etc)
5. Uso de parámetros
6. Crear disparadores para cada una de las operaciones DML aplicadas a una tabla.
7. Utilización de disparadores por ejemplo en la auditoría y seguridad
8. Reflejar el uso de funciones creadas por el usuario

## **PRACTICA 13. Programación con SQL. Parte 2**

### **Objetivo:**

El alumno pondrá en práctica los conceptos básicos de programación procedural en la construcción de procedimientos almacenados y cursores

### **Introducción**

Un procedimiento almacenado es un conjunto de comandos de SQL que pueden ser compilados y almacenados en el servidor. Una vez realizado esto, los clientes no necesitan volver a teclear todas las instrucciones sino únicamente hacer referencia al procedimiento. Esto mejora el rendimiento del servidor, ya que la instrucción de SQL solamente es revisada una sola vez y menos información debe ser enviada entre el cliente y el servidor. Los procedimientos son llamados también subprogramas.

Un cursor es un área de memoria utilizada para realizar operaciones con los registros devueltos tras ejecutar una sentencia SELECT. Existen dos tipos:

- Implícitos
- Explícitos

### **Actividades a realizar:**

1. Con base en un modelo relacional proporcionado implementar procedimientos de alta, baja y modificación de datos.
2. Realizar cursores para validación de información o reglas de negocio.
3. Opcionalmente utilizar cursores anidados

## Bibliografía

ELMASRI RAMEZ A., NAVATHE SHAMKANT B., Todos  
Fundamentos de Sistemas de Bases de datos,  
Pearson Prentice Hall,  
ISBN: 8478290516, 2003

DE MIGUEL MARTÍNEZ, Adoración, PIATTINI, Mario, ESPERANZA, Marcos  
*Diseño de bases de datos relacionales*  
México  
Alfaomega, 2000

DE MIGUEL, Adoración, PALOMA CASTRO, Elena  
*Diseño de bases de datos (Problemas Resueltos)*  
México  
Alfaomega, 2001

ROB, Peter; CORONEL, Carlos  
*Database systems (Design, Implementation and Management)*  
6th. Edition  
U.S.A. Course Technology, 2004

ARELLANO M., Lucila P. y Hernandez Hdez. Luciralia  
*Manual de prácticas de la asignatura de Bases de Datos*  
UNAM, Fac. De Ingeniería., DIE