■ BASES DE DATOS EN INFORMIX ■

Del 14 de junio al 2 de julio, 1994

DIRECTORIO DE PROFESORES

1. ING. JESSICA BRISEÑO CORTES (Coordinadora)
   Norte 86 B No. 4729
   Col. Nueva Tenochtitlan
   Tel . 751-92-56

2. ING. J. ANTONIO CHAVEZ FLORES
   SYBASE DE MEXICO, S.A.
   Tel. 282-80-78

3. ING. EDWIN NAVARRO PLIEGO
   AYUDANTE DE PROFESOR "B"
   FACULTAD DE INGENIERIA,
   U. N. A. M.
   CIUDAD UNIVERSITARIA
   TEL. 622-30-53
   622-30-54

EVALUACION DEL PERSONAL DOCENTE

CURSO: BASES DE DATOS EN INFORMIX
FECHA: 14 DE JUNIO AL 02 JULIO DE 1994.

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACION CON EL ASISTENTE	PUNTUALIDAD
ING. JESSICA BRISEÑO CORTES				
ING. J. ANTONIO CHAVEZ FLORES				
ING. EDWIN NAVARRO PLIEGO				
ING. ROMAN RAMIREZ HERNANDEZ				

EVALUACION DE LA ENSEÑANZA

ORGANIZACION Y DESARROLO DEL CURSO	
GRADO DE PROFUNDIDAD LOGRADO EN EL CURSO	
ACTUALIZACION DEL CURSO	
APLICACION PRACTICA DEL CURSO	

EVALUACION DEL CURSO

CONCEPTO	CALIF.
CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
CONTINUIDAD EN LOS TEMAS	
CALIDAD DEL MATERIAL DIDACTICO UTILIZADO	

ESCALA DE EVALUACION: 1 A 10

COORDINACION CURSOS DE COMPUTO
CENTRO DE INFORMACIÓN Y DOCUMENTACION

1.- ¿LE AGRADO SU ESTANCIA EN LA DIVISION DE EDUCACION CONTINUA?

SI	NO

SI INDICA QUE "NO" DIGA PORQUE.

2.- MEDIO A TRAVES DEL CUAL SE ENTERO DEL CURSO:

PERIODICO EXCELSIOR		FOLLETO ANUAL		GACETA UNAM		OTRO MEDIO	
PERIODICO EL UNIVERSAL		FOLLETO DEL CURSO		REVISTAS TECNICAS			

3.- ¿QUE CAMBIOS SUGERIRIA AL CURSO PARA MEJORARLO?

4.- ¿RECOMENDARIA EL CURSO A OTRA(S) PERSONA(S)?

SI		NO	

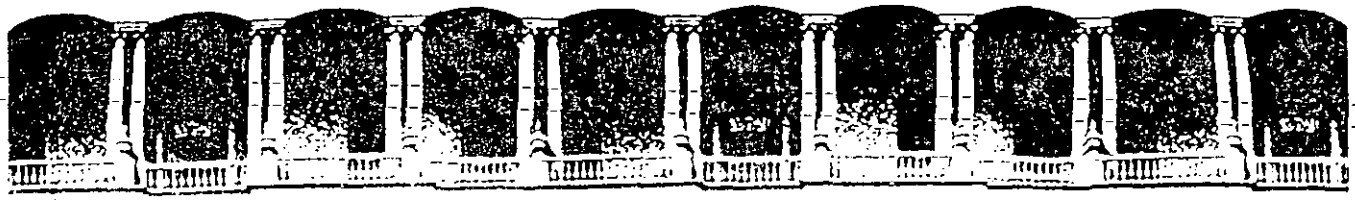
5.- ¿QUE CURSOS LE SERVIRIA QUE PROGRAMARA LA DIVISION DE EDUCACION CONTINUA.?

6.- OTRAS SUGERENCIAS:

7.- ¿EN QUE HORARIO LE SERIA CONVENIENTE SE IMPARTIERAN LOS CURSOS DE LA DIVISION DE EDUCACION CONTINUA?
MARQUE EL HORARIO DE SU AGRADO

LUNES A VIERNES DE 16 A 20 HORAS	MARTES Y JUEVES DE 17 A 21 HS SABADO DE 10 A 14 HS.	OTRO
LUNES, MIERCOLES Y VIERNES DE 17 A 21 HORAS	VIERNES DE 17 A 21 HS. SABADOS DE 10 A 14 HS	

COORDINACION CURSOS DE COMPUTO
CENTRO DE INFORMACIÓN Y DOCUMENTACION



FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

" BASES DE DATOS EN INFORMIX "

Del 14 de junio al 2 de julio, 1994

I N T R O D U C C I O N

ING. C. JESSICA BRISEÑO CORTEZ

JUNIO, 1994

Introducción

Coordinador:
Ing. C. Jéssica Briseño Cortez

BASES DE DATOS

- Conjunto de datos interrelacionados con redundancia controlada para servir a una o más aplicaciones; los datos son independientes de los programas que los usan .

- Conjunto de datos interrelacionados.

De estas definiciones podemos resumir los siguientes puntos importantes:

- Sistema de aplicación de la empresa.

- Colección de datos de operación.

- Datos interrelacionados.

- Facilidad de acceso, análisis y producción.

- Independencia programas-datos.

Problemas a solucionar:

- Redundancia e inconsistencia de los datos.

- Dificultad para tener acceso a los datos.

- Aislamiento de los datos.

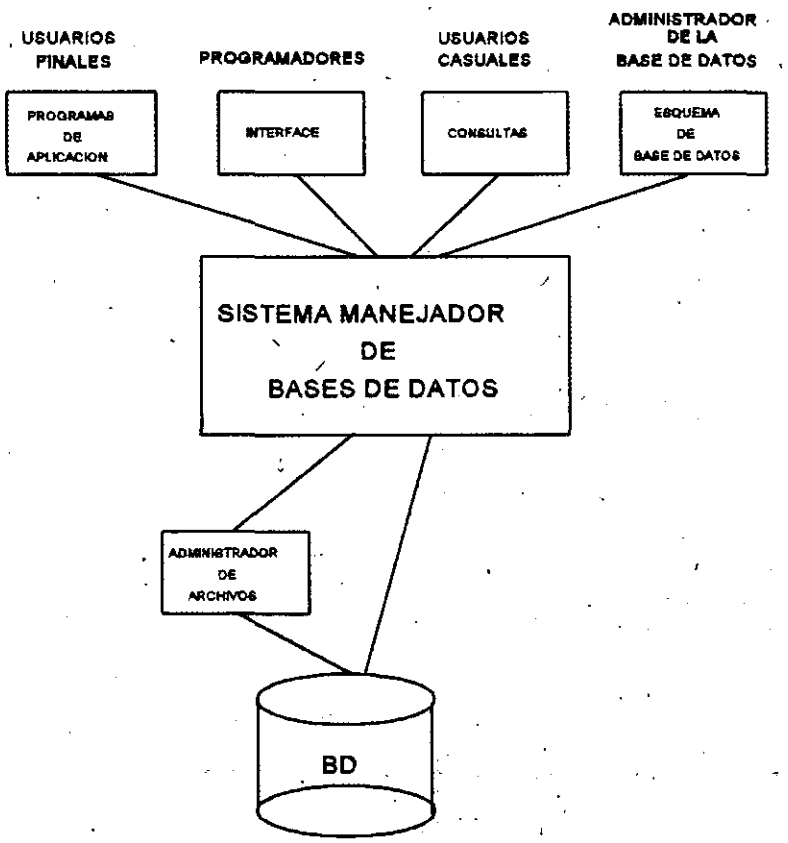
- Usuarios concurrentes.

- Seguridad.

- Integridad.

SISTEMA MANEJADOR DE BASES DE DATOS

Conjunto de programas que sirven para administrar, controlar, accesar y manipular una base de datos.



EJEMPLOS:

- SYBASE
- Oracle
- Informix
- Ingres

SISTEMAS DE PROCESAMIENTO DE ARCHIVOS

Sistemas en donde la información se guarda en un conjunto de archivos (conocidos comunmente como archivos planos) y se escriben varios programas de aplicaciones para obtener información de dichos archivos, así como para modificarlos.

Características:

- Los programas de aplicación se crean como respuesta a las necesidades de la empresa.

- Los programas de aplicación generalmente causan la creación de archivos de datos con información que muy probablemente se encuentre en otros archivos.

- Archivos con formatos diferentes.

- Programas escritos en diferentes lenguajes.

- Dependencia directa del sistema operativo para controlar la seguridad de los datos.

Desventajas:

- Redundancia e inconsistencia de los datos.

- Aislamiento de los datos.

- Usuarios múltiples sin control.

- Dificultad para tener acceso a los datos.

- Problemas de seguridad.

- Problemas de integridad.

SISTEMAS EN BASES DE DATOS

Los Manejadores de Bases de Datos nos ayudan a implementar sistemas de información bajo la perspectiva de Bases de Datos, proporcionandonos los siguientes servicios:

- Interacción con el sistema operativo para efectos del almacenamiento, recuperación y actualización de los datos en la base de datos.

- Implantación de la integridad.

- Seguridad.

- Respaldo y recuperación.

- Control de concurrencia.

- Herramientas para la explotación de los datos.

USUARIOS DE LA BASE DE DATOS

Existen diferentes puntos de vista y aplicaciones que los usuarios llevan a cabo sobre una Base de Datos, es por ello que el Manejador de Base de Datos debe contar con los mecanismos necesarios que esten de acuerdo con el tipo de usuario involucrado. Podemos definir los siguientes tipos de usuarios:

- Usuarios finales
- Usuarios casuales
- Programadores de aplicaciones
- Programadores especializados
- El Administrador de la Base de Datos

EL ADMINISTRADOR DE LA BASE DE DATOS

Una de las razones por las que se utilizan sistemas en Bases de Datos es el tener un control centralizado de la información que se maneja y de los programas que la accesan. El Administrador de la Base de Datos o DBA (*Data Base Administrator*) es un usuario especial que tiene como función controlar la Base de Datos y la forma en como esta es accesada.

Las funciones principales del DBA son:

- Definición del esquema de la Base de Datos
- Definición de las estructuras de almacenamiento y de los métdos de acceso
- Modificación del esquema y de la organización física
- Autorización para acceso a los datos
- Especificación de restricciones de integridad
- Especificación de politicas para con la Base de Datos

MODELOS LOGICOS

Una Base de Datos se compone esencialmente de datos; además existen mecanismos que permiten tener un acceso rápido y eficiente a los mismos; pero ¿cuándo se va ha definir qué datos estarán guardados y cuál será la organización que tendrán?

Tratando de encontrar una representación accesible, se han desarrollado modelos que ilustran la lógica del comportamiento, las restricciones, las relaciones entre los datos en una forma fácil de entender, organizar y modificar. Entre estos modelos, los más conocidos son:

- Modelos lógicos basados en registros

- Modelos lógicos basados en objetos

MODELOS LOGICOS BASADOS EN REGISTROS

Estos modelos representan a la realidad tomando como base la forma en como los datos son almacenados en la computadora y como son mantenidas las asociasiones entre ellos. En estos modelos se crea una dependencia con la forma en como se implementa la Base de Datos.

Dentro de los modelos lógicos basados en registros tenemos los siguientes:

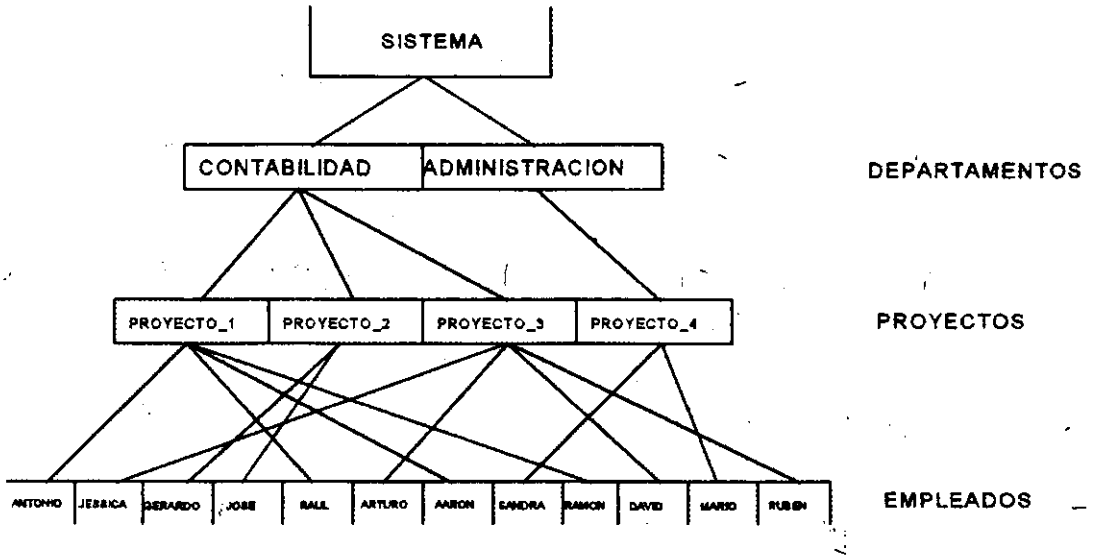
- Modelo jerárquico

- Modelo de red

- Modelo relacional

MODELO JERARQUICO

En este modelo, los datos se representan como una colección de registros, mientras que la relación entre ellos se da por medio de ligas o apuntadores.



Desventajas:

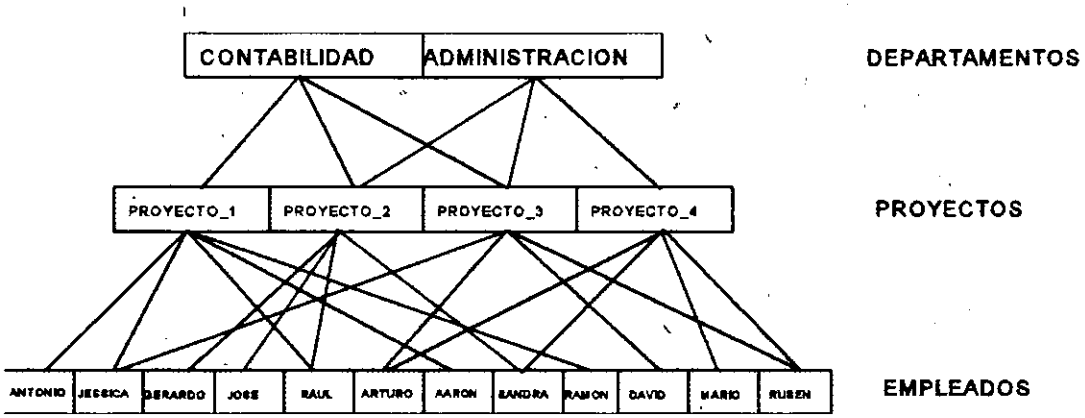
- No puede haber ciclos y sólo puede haber asociaciones 1:N y 1:1
- Los apuntadores o direcciones se deben almacenar junto con los datos
- Para recuperar información se debe recorrer el árbol
- No se puede obtener información no planeada antes de modelar
- Modificar la estructura de la Base de datos implica redefinir todo el esquema
- Se pueden representar asociaciones M:N manteniendo datos duplicados

Ventajas:

- Acceso rápido a los datos debido a los apuntadores

MODELO DE RED

Al igual que en el modelo jerárquico, en este modelo, los datos se representan como una colección de registros, la relación entre ellos se da por medio de apuntadores, la diferencia radica en que los registros de la Base de Datos se organizan en forma de gráficas arbitrarias.



Desventajas:

- Los apuntadores o direcciones se deben almacenar junto con los datos
- Para recuperar información se debe "navegar" a través de la gráfica
- No se puede obtener información no planeada antes de modelar
- Modificar la estructura de la Base de datos implica redefinir todo el esquema

Ventajas:

- Acceso rápido a los datos debido a los apuntadores.

MODELO RELACIONAL

En el modelo relacional, los datos y las relaciones entre los datos se representan por medio de una serie de tablas, cada una de las cuales esta compuesta por columnas con nombres únicos. Una columna de una tabla representa una relación entre un conjunto de valores. Existe una correspondencia entre el concepto de tabla y el concepto matemático de relación, del cual recibe su nombre el modelo relacional.

EMPLEADOS

NO_CTA	NOMBRE	DIR	TEL	NO_DEPTO
18456	Antonio	Revolucion 123	733-22-89	10
18272	Jessica	Norte 86B 98	657-28-92	40
72638	Gerardo	Rio chico 22	512-38-39	10
28289	Jose	Palmas 926	833-32-21	30
29829	Raul	Rosas 83-5	937-00-52	30
87719	Arturo	Churubusco 45	723-45-11	30
32983	Aaron	Taxqueña 372	632-73-21	20
32732	Sandra	Av. Central 13	743-03-01	40
32903	Ramon	Marina Nal. 86	732-37-77	20
95672	David	Almaraz 2-1	664-83-00	10
48568	Mario	Cozumel 11-3	731-82-83	40
84324	Ruben	Fco. Sosa 266	527-73-12	20

DEPARTAMENTOS

NO_DEPTO	NOMBRE
10	Sistemas
20	Finanzas
30	Contabilidad
40	Ingeniería

PROYECTOS

NO_PROY	DESCRIPCION
1	Diseño del Sistema de Inventarios
2	Sistema de Nomina
3	Sistema Integral de Servicios
4	Evaluación de Equipo de Cómputo

ASIGNADO

NO_CTA	NO_PROY
28289	2
95672	3
48568	4
84324	3
84324	4
18456	1
29829	2
32983	1
32732	4
18272	1
29829	1
72638	2
18272	3
87719	3
32732	2
32903	1
87719	4

Ventajas:

- Tiene una base matemática, conocida como Algebra y Cálculo Relacional

- Se pueden representar fácilmente asociaciones M:N

- No existen apuntadores u otro tipo de información que no sea la que creó la necesidad de la Base de Datos

- Las operaciones efectuadas para obtener información se realizan a nivel de la tabla completa y no a nivel de registros

- No es necesario diseñar el esquema de la Base de Datos de acuerdo a las operaciones o consultas que se van a llevar a cabo. Ees posible obtener información no prevista

- Se puede modificar la estructura de la Base de Datos sin que esto obligue a un cambio de las aplicaciones

- La forma de explotar la información es por medio de operaciones relacionales, a través de un lenguajes de cuarta generación

MODELOS LOGICOS BASADOS EN OBJETOS

Estos modelos son relativamente recientes, se caracterizan por ser muy flexibles e independientes de la forma en que los datos se almacenan y manipulan, además de que hacen posible especificar claramente las limitantes de los datos.

Los modelos basados en objetos siguen en desarrollo con vistas a hacer una representación más completa y versátil de la información. Se incluyen en estudios de Inteligencia Artificial y Bases de Conocimiento.

ESQUEMAS

El esquema de la Base de Datos lo constituye el diseño de la misma. Existen varios esquemas en la Base de Datos de acuerdo al nivel que describen, el objetivo es que estos esquemas sean independientes. A la capacidad de modificar una definición de esquema en un nivel sin afectar la definición del esquema en el nivel inmediato superior se denomina *independencia de datos*.

Los esquemas que existen en una Base de Datos son:

- Esquema fisico

En este esquema se define la forma en como se almacenan realmente los datos.

- Esquema lógico

En este esquema se describen cuáles son los datos reales que están almacenados en la Base de Datos y que relaciones existen entre ellos. Un mismo modelo tendrá diferentes esquemas fisicos al ser implantado en diferentes DBMS.

LENGUAJES

Para definir el esquema de la Base de Datos, así como para explotar la información que se encuentra en ella es necesario contar con un medio de comunicación proporcionado por el DBMS. Existen dos tipos de lenguajes básicamente, los cuales en la mayoría de los casos están inmersos en uno:

- Lenguaje de Definición de Datos

- Lenguaje de Manipulación de Datos

El Lenguaje de Manipulación de Datos puede ser de alguno de los siguientes tipos:

- Procedurales: se especifica el cómo los datos son recuperados

- No Procedurales: se especifica únicamente los datos que serán recuperados

Un lenguaje será más productivo en tanto más No Procedural sea y más versátil para aplicaciones especiales en tanto más Procedural se comporte.

SQL (Structured Query Language)

SQL es un lenguaje tanto para la definición de los datos como para la manipulación de ellos.

Características:

- Es un lenguaje estándar reconocido por ANSI e ISO

- Se encuentra implementado en la mayoría de los DBMS más populares

- Es un 4GL

- Es muy fácil de utilizar

- No incluye referencias físicas de los datos

- Es utilizado desde muchos programas de aplicación que forman parte de un RDBMS

- Es utilizado para la obtención, modificación y definición de los datos, así como para la Administración de la Base de Datos

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

BASES DE DATOS EN INFORMIX

ALGEBRA RELACIONAL

JUNIO, 1994.

Implementación de una pequeña Base de Datos

ALUMNO

No_cta	Nom_alum	Cve_carr
105	Sandra	027
107	José	032
101	Luis	032
103	Lourdes	027
106	Juan	029
102	Santiago	029
108	Véronica	027

PROFESOR

Cve_prof	Nom_prof
acf000	Antonio
rrh000	Ramón
enp000	Edwin
jbc000	Jéssica

CARRERA

Cve_carr	Carrera
027	Industrial
032	Computación
039	Electrónica

MATERIA

Cve_mat	Materia
062	Base de Datos
011	Sistemas Operativos
030	Electrónica
045	Redes
039	Compiladores
058	Comunicaciones

INSCRITO

No_cta	No_gpo
107	05
103	08
102	02
108	07
105	03
105	07
101	1
107	1

GRUPO

No_gpo	Cve_mat	Cve_prof
01	030	rrh000
05	011	rrh000
07	058	enp000
08	062	acf000
02	011	enp000
03	030	enp000

INFORMIX-SQL

Comenzando

Informix-SQL se puede ejecutar para diferentes plataformas, en ambiente UNIX. los pasos a seguir para la ejecución de ISQL son los siguientes:

1. login: cursoN ↵

2. passwd: ■■■■■ ↵

3. cursoN prometeo> isql ↵

Informix-SQL Menú Principal

El Menú Principal muestra las siguientes opciones:

Form

Report

Query-Language

User-menu

Database

Table

Exit

Formato de los menus de ISQL

A excepción de los menus de Query-Language y User-menu; los demas siguen el siguiente formato:

INFORMIX-SQL: **Form** **R**eport **Q**uery-Lang **U**ser-menu **D**atabase **T**able **E**xit
Run, Modify, Create, or Drop a form

---------- Press CTRL-W for Help ----------

Opcion elegida

esta segunda línea es un comentario con respecto
al menú seleccionado

nombre del menú superior (en este caso es el menú princiupal)

Si se tiene alguna duda sobre el menú elegido o acerca de alguna de sus opciones, ISQL proporciona una pequeña ayuda oprimiento CONTROL-W

La última opción de cada menú es el comando EXIT esta opción regresa al menú anterior, en este caso al seleccionar esta opción se cierra la sesión de ISQL regresando al Sistema Operativo.

Usando los menús

Cualquiera de las opciones (menús) es seleccionada oprimiendo la primera letra del nombre de dicha opción, o bien, utilizando las flechas ó la barra espaciadora y moviendo el cursor hacia la opción que se desee y presionando ENTER ó RETURN.

Algunas opciones despliegan el simbolo >>, lo que indica que está que requiere de alguna información.

Para cancelar alguna opción presionar el comando de INTERRUPT, en este caso es CTRL-C

Si algún menú contiene más opciones de las que se puedan ver en una línea, aparecerán punto suspensivos ..., lo que indica que existen más opciones.

Creando una Base de Datos

Para crear una Base de Datos se debe de elegir la opción del Main Menu (Menú Principal) **Database**.

Posteriormente apareceran las siguientes opciones:

Select

Create

Drop

Exit

Después seleccionaremos la opción de **Create** y apareceran los simbolos >> lo que nos indica que está esperando el nombre de la Base de Datos.

El nombre de la Base de Datos debe cumplir con los siguientes puntos:

- Máximo debe tener 10 caracteres
- Debe comenzar con una letra
- Puede contener letras, números, y el subguión (_)
- No existe distinción entre mayusculas y minisculas

Una vez que se ha introducido el nombre de la base de datos presionar ENTER

Si la creación de la Base de Datos tuvo éxito ISQL regresa el control al menú de **Database**.

Para salir de este menú elegir la opción de **Exit**.

Tipos de datos en ISQL

En ISQL existen cuatro tipos de datos básicos :

- **CHAR** se utiliza para cadenas de caracteres , o bien, un caracter (Jéssica, 1-23-45-67, AB, C)
- **NUMERIC** para valores numericos (1, 3.1415, -100) :
  - INTEGER número enteros que tienen un rango de -2,147,483,647 a +2,147,483,647
  - SMALLINT número enteros con un rango más pequeño -32,76 7 a +32,757
  - DECIMAL números reales con un máximo de 32 digitos significativos
  - FLOAT número reales, los cuales tienen dos tipos:
    - FLOAT núeros reales con presición de hasta 16 digitos significativos
    - SMALLFLOAT núeros reales con presición de 8 digitos significativos
- **SERIAL** para valores secuenciales (1, 2, 3, ... , n)
- **DATE** para valores de tipo fecha (17-JUN-94, 17-06-94)
- **MONEY** es un valor numerico de tipo decimal ($ 1250.00)

Esto tipos de datos se utilizan para la definición de los atributos de las tablas.

Indices de ISQL

Sobre una tabla se pueden crear indices, los cuales pueden ser:

UNIQUE	no permite valores duplicados en una columna
CLUSTER	fisicamente, los registros son ordenados en la misma secuencia que los indices
DUPLICATE	permite que haya mas de una ocurrencia del valor de la columna especificada
COMPOSITE	permite crear un índice sobre dos o más columnas. Estos pueden ser también unique ó duplicate.

Valores NULL

Un valor NULL implica un valor desconocido:

- Un valor NULL no implica cero o una cadena vacía
- Un valor NULL no es igual a otro valor NULL
- Operaciones que involucran NULL dan como resultado NULL

Ejemplo:

Nombre	Empleo	Comisión
José	Manager	NULL
Luis	Vendedor	0
Juan	Vendedor	1,054
Gerardo	Analista	NULL

Creando una Tabla

Para crear una Tabla se debe de elegir la opción del Main Menu (Menú Principal) **Table**.

Posteriormente apareceran las siguientes opciones:

Create

Alter

Info

Drop

Exit

Después seleccionaremos la opción de **Create** y apareceran los simbolos >> lo que nos indica que está esperando el nombre de la Tabla.

El nombre de la Tabla debe cumplir con los siguientes puntos:

- Debe ser único
- Máximo debe tener 18 caracteres
- Debe comenzar con una letra
- Puede contener letras, números, y el subguión (_)
- No existe distinción entre mayusculas y minisculas

Una vez que se ha introducido el nombre de la tabla presionar ENTER

Si la creación de la Tabla tuvo éxito ISQL regresa el control al menú de **Table**.

Para salir de este menú elegir la opción de **Exit**.

Insertar Datos

Una manera sencilla de insertar la información en las tablas es haciendo uso del menú Form, esto es:

Una vez creada la tabla, seleccionar el menú **Form** del Main Menu

Posteriormente apareceran las siguientes opciones:

Run

Modify

Generate

New

Compile

Drop

Exit

Después seleccionaremos la opción de **Generate** , la cual genera una forma por default; apareceran los simbolos >> lo que nos indica que está esperando el nombre de la Tabla sobre la cual se va a generar la forma.

Una vez que se ha seleccionado la tabla presionar ENTER

Si la creación de la Forma tuvo éxito ISQL mostrará la forma, después se eligirá la opción **ADD** del menú en uso, con la cual podremos introducir la información.

Para validarla presionar la tecla **ESC** y con esto el renglon se habrá insertado en la tabla.

Laboratorio

1. Crear una Base de Datos llamada CursoN, si ya existe seleccionarla.

2. Crear la tabla una tabla llamada EMP la cual contendrá los siguientes atributos:

Columna	Tipo de dato
EMPNO	INTEGER UNIQUE INDEX NOT NULL
ENAME	CHAR (10)
JOB	CHAR(15)
MGR	INTEGER NULL
HIREDATE	DATE
SAL	MONEY
COMM	MONEY NULL
DEPTNO	SMALLINT NOT NULL

3. Insertar los datos de la tabla de EMP

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

BASES DE DATOS EN INFORMIX

Del 14 de junio al 2 de julio, 1994.

D I S E Ñ O

ING. C. JESSICA BRISEÑO CORTES

DISEÑO

OBJETIVO:

En este capítulo se estudiaran las técnicas para la obtención de tablas a partir del modelado representado por medio de un Diagrama de Entidades y Relaciones.

En este capítulo el asistente:

- Aprenderá a obtener las tablas que componen la Base de Datos a partir de entidades.

- Aprenderá a diseñar las relaciones entre tablas por medio de llaves foráneas o tablas de relación.

- Enetenderá el concepto de redundancia.

- Comprenderá el proceso de normalización de un diseño de Bases de Datos.

Entidades

Es un objeto que existe y que es distinguible de otros objetos.

Es algo (persona, lugar, objeto, concepto) a lo que la Empresa le reconoce poder existir en forma independiente y que puede ser definido en forma única.

Ejemplo: película, cliente, empleado, departamento, máquina, etc.

Una instancia de una entidad es un elemento de ese tipo de entidad, por ejemplo: "Los gritos del silencio" (película), "Sistemas" (departamento), etc.

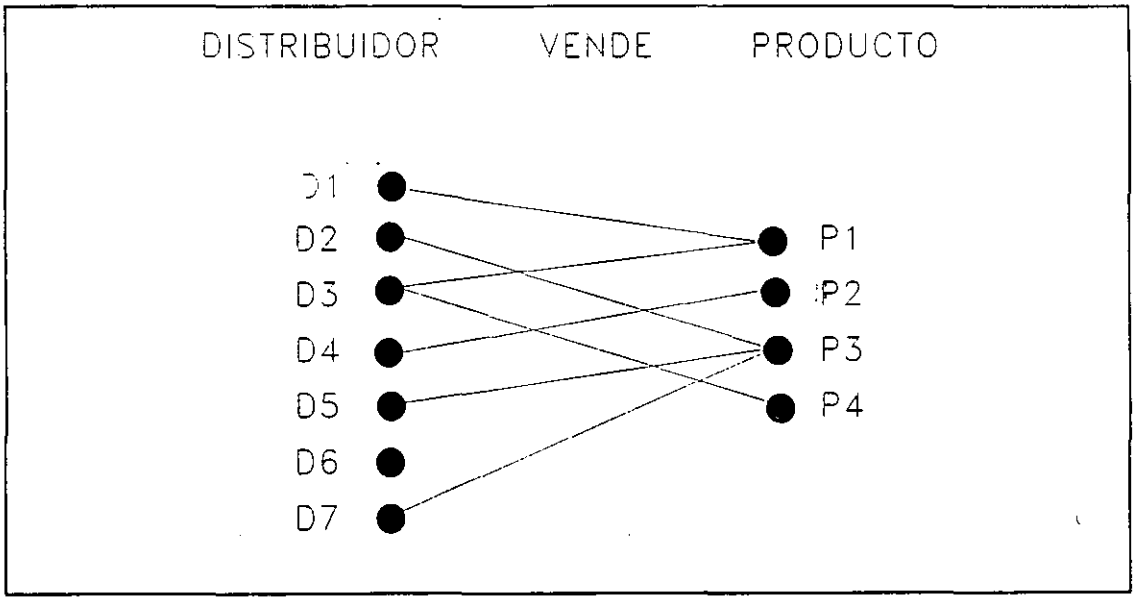
Las entidades representan tablas en el diseño relacional.

Asociaciones

Es el vínculo que existe entre dos o más Entidades.

Por ejemplo, la Entidad departamento puede asociarse con la Entidad empleado vía la Asociación emplea.

Ejemplo:



Una instancia de una asociación es un elemento de esa relación entre entidades, por ejemplo:.Juan Pérez TRABAJA Sistemas.

Llave primaria

Una llave primaria es una llave candidato que ha sido seleccionada por el diseñador de la base de datos como el medio de identificar Entidades.

Las características necesarias para una llave primaria son las siguientes:

- Unica
- Conocible en cualquier tiempo

Las características deseables de una llave primaria son las siguientes:

- Estable
- No descriptiva
- Pequeña y simple

Llave foránea

Una llave foránea en una tabla son las columnas que conforman la llave primaria de otra tabla.

Es conveniente que la llave foránea tenga el mismo nombre y tipo de la llave primaria de la que proviene.

La llave primaria de una tabla puede estar formada en parte por una llave foránea.

Atributos

Es una propiedad de una Entidad.

Por ejemplo, número, nombre y RFC pueden ser atributos de la Entidad cliente.

Los atributos generalmente describen a una entidad.

Hay que decidir el tipo asociado a los atributos, por ejemplo: entero, real, carácter, etc.

Hay que determinar cuales son atributos indirectos en una relación, por ejemplo, en la relación EMPLEADO TRABAJA DEPARTAMENTO, el atributo número de sucursal es un atributo indirecto de EMPLEADO en la relación.

Diagramas de Entidades y Asociaciones

El análisis de Entidades y Asociaciones cuenta con una herramienta gráfica para cumplir sus objetivos.

El proceso se realiza dibujando diagramas conocidos como Diagramas de Entidades y Asociaciones (DEA).

Las convenciones al dibujar DEA son:

1. Las Entidades serán representadas por rectángulos.
2. Las Asociaciones serán rombos.
3. Las líneas de conexión mostrarán qué Entidades son vinculadas por cuál Asociación.
4. Los atributos de las Entidades y las Asociaciones se muestran como círculos o elipses conectados al rombo o rectángulo correspondiente. Generalmente no se dibujan.
5. El grado de la Asociación será representado por 1, M ó N, sobre las líneas de conexión.
6. La obligatoriedad de una entidad débil se indicará terminando la correspondiente línea de conexión dentro de un pequeño rectángulo que forme parte de la Entidad.

Grado de asociación

El grado de asociación o cardinalidad, representa en forma cualitativa, el número de ocurrencias de una entidad con las que puede estar asociada una instancia de otra entidad.

Una asociación puede tener tres tipos de grados:

TIPO		PUEDE INCLUIR					
1:1	(uno a uno)	1:0	0:1	1:1			
1:N	(uno a muchos)	1:0	0:1	1:1	1:N		
M:N	(muchos a muchos)	1:0	0:1	1:1	1:N	N:1	M:N

El grado de asociación puede ser obtenido de las reglas de empresa.

Obtención de tablas

La primera etapa de Diseño es la obtención de esqueletos de las tablas que componen el modelo de Bases de Datos.

El esqueleto de una tabla se compone de: el nombre de la tabla, que usualmente es el nombre de la Entidad o Asociación; una lista de atributos mínimos que debe contener esa tabla, que por lo regular son una llave candidato y las llaves foráneas necesarias para mantener el vínculo con otras tablas; y grupos de tres puntos, que indican la futura presencia de otros atributos de la tabla.

La llave candidato se coloca al principio de la lista y se subraya para indicar su calidad de identificador de la Entidad.

La segunda etapa es la asignación del resto de atributos, colocándolos en la tabla que les corresponde, y cumpliendo siempre con las reglas de normalización.

Es un hecho que el conjunto de tablas resultantes puede ser implantado directamente dentro del ambiente de un manejador de bases de datos relacionales, puesto que cada tabla será una relación bien normalizada.

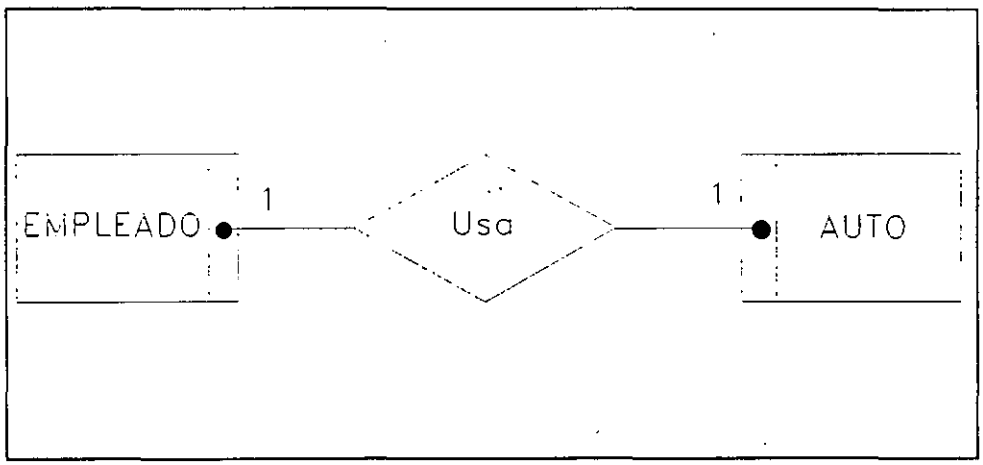
Representación de Entidades

Cada entidad independiente representa una tabla.

La llave primaria de tablas independientes no contiene llaves foráneas.

Representación de Realaciones 1:1

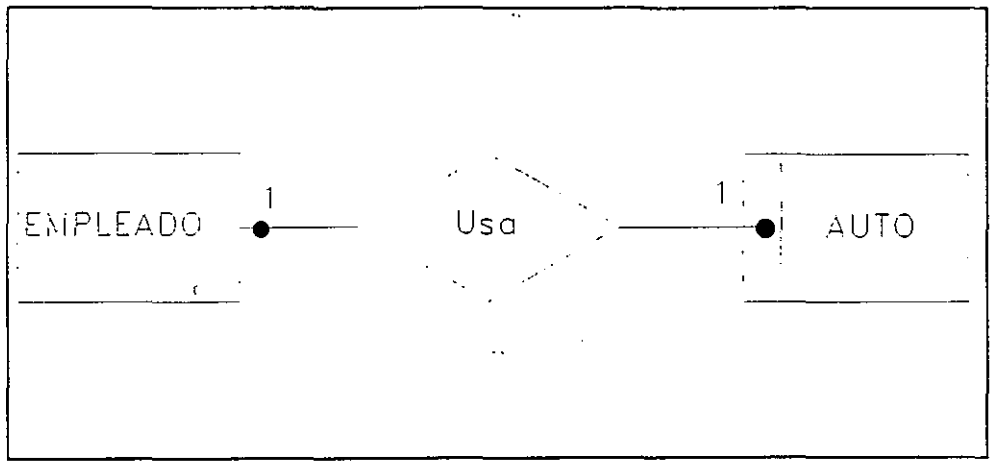
Obligatoriedad para ambas Entidades:



EMPLEADO(#empleado, #auto, ...)

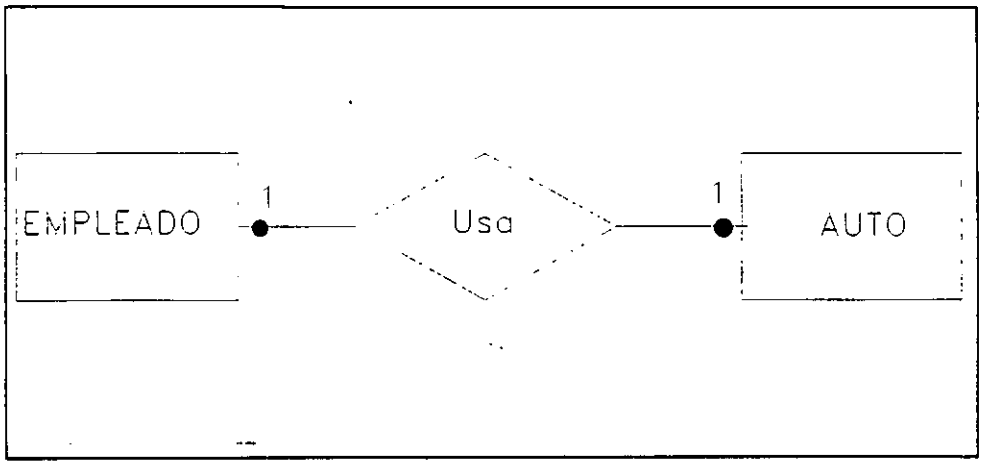
#empleado y #auto son la llave primaria de la Entidad.

Obligatoriedad sólo para una Entidad:



EMPLEADO(#empleado, ...)
AUTO(#auto, ... , #empleado)

No obligatoriedad para las dos Entidades:

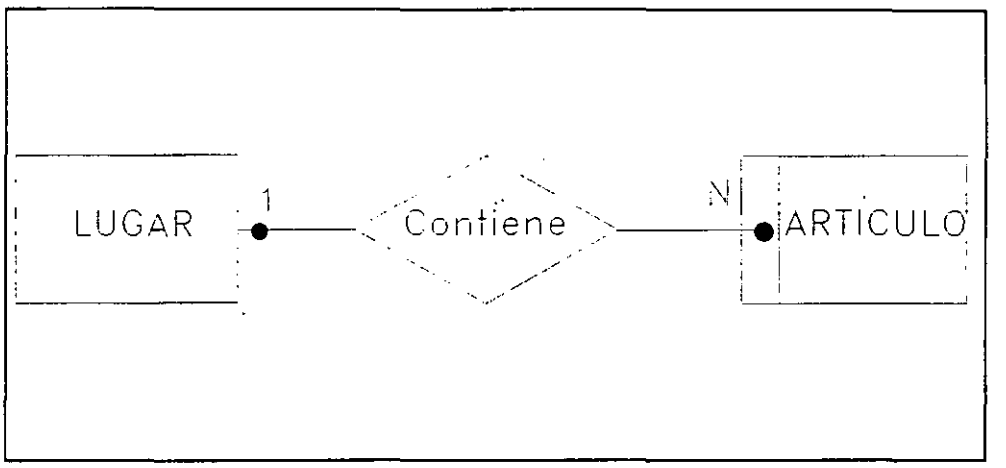


Se crea una tabla para la Relación:

EMPLEADO(#empleado, ...)
AUTO(#auto, ...)
USA(#empleado,#auto, ...)

Representación de Realaciones 1:N

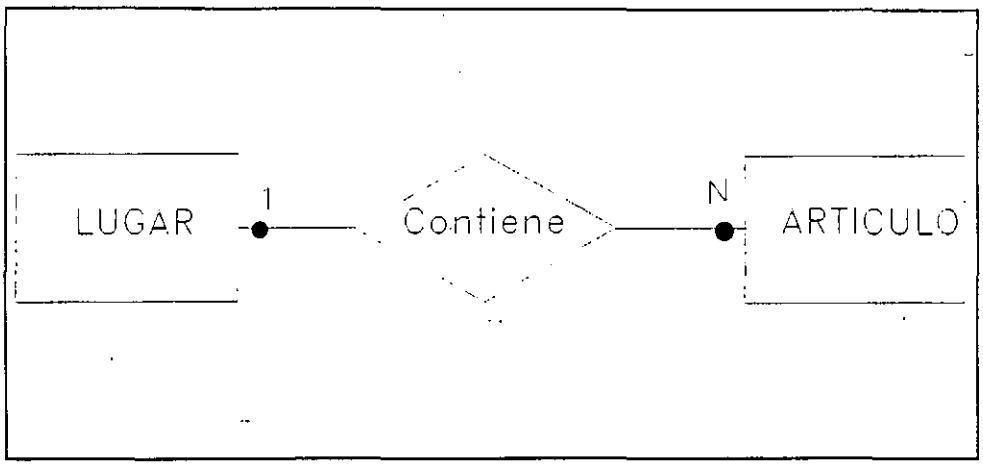
Obligatoriedad en la Entidad de grado N:



LUGAR(nombre, ...)
ARTICULO(#artículo, ..., nombre)

Nótese que la llave primaria de la Entidad de grado 1 se convierte en llave foránea de la tabla que representa la Entidad de grado N, **en ese orden**. De otra forma la llave de ARTICULO podría tomar valores nulos.

Sin obligatoriedad en la Entidad de grado N:



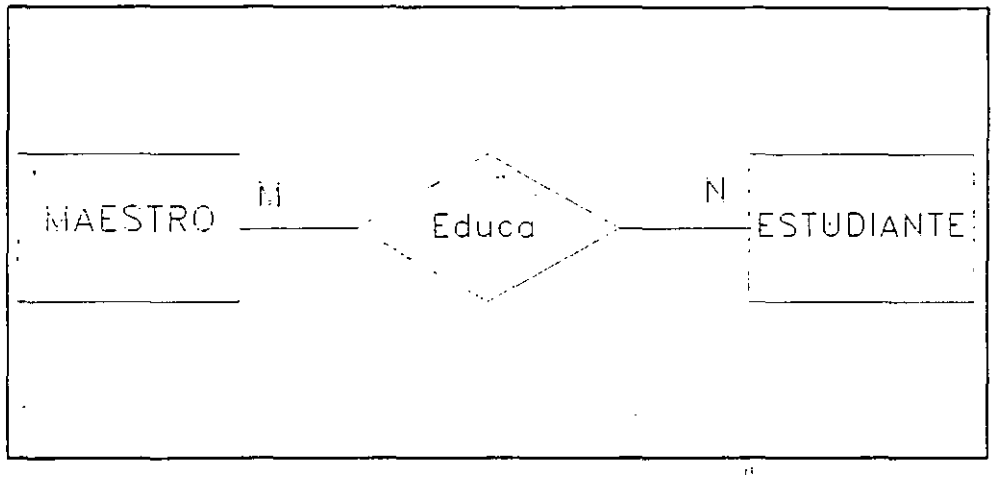
LUGAR(nombre, ...)
ARTICULO(#artículo, ...)
CONTIENE(#artículo, nombre, ...)

La llave primaria de la Entidad de grado 1 se convierte en la llave foránea de la Tabla que representa a la Asociación, **en ese orden.**

Los casos en que existe o no obligatoriedad en la Entidad de grado 1 no modifican nada.

Representación de Realaciones M:N

No importa la existencia o no de obligatoriedad en las Entidades:



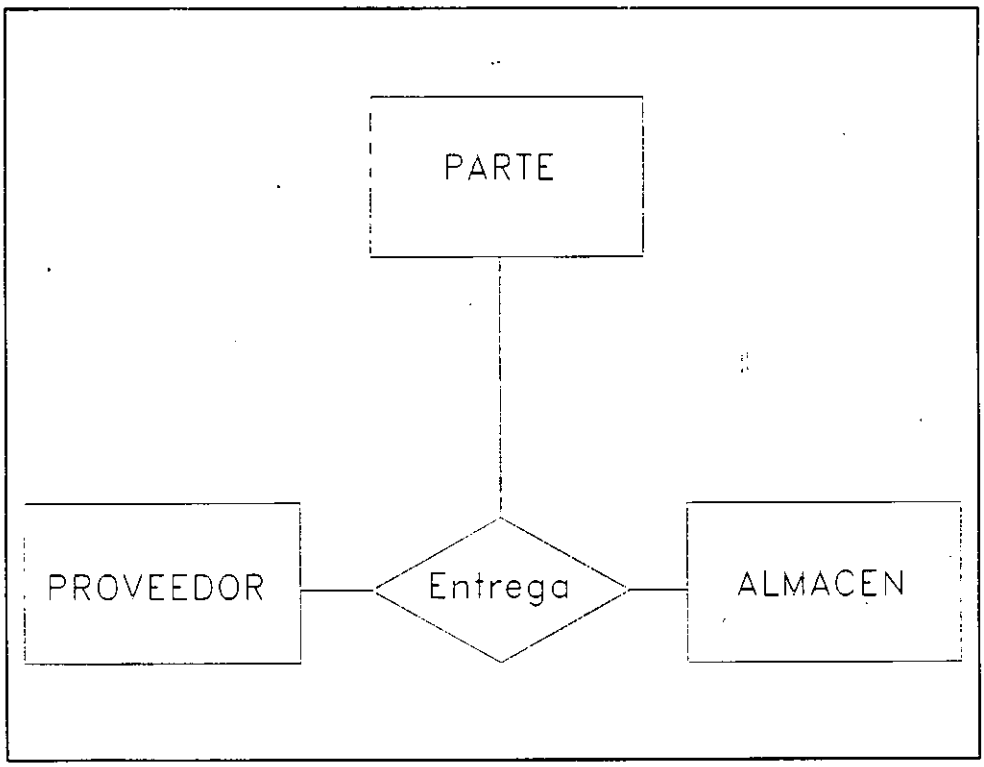
MAESTRO(nombre, ...)
ESTUDIANTE(#estudiante, ...)
EDUCA(nombre, #estudiante, ...)

La tabla que representa la Asociación debe tener como llave primaria la concatenación de las llaves de las Entidades.

Representación de Realaciones N-arias

Las relaciones N-arias, son aquellas que existen entre más de dos Entidades.

No importan mucho los grados de asociación o de pertenencia, de cualquier forma se debe representar la Asociación y formar una llave primaria con la concatenación de los identificadores de las Entidades que participan.

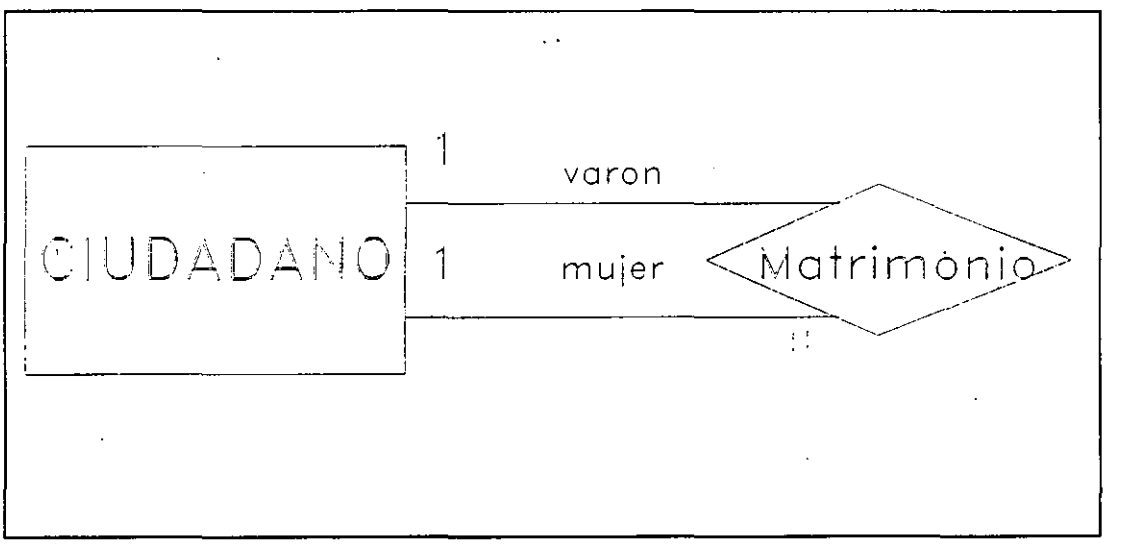


PARTE(#parte, ...)
ALMACEN(#almacén, ...)
PROVEEDOR(#proveedor, ...)
ENTREGA(#proveedor,#parte,#almacén, ...)

Relaciones recursivas

Son las que se dan entre Entidades del mismo conjunto de Entidades.

Un ejemplo lo podemos ver considerando al conjunto de Entidades CIUDADANO y la Asociación MATRIMONIO, que debe ser entre dos CIUDADANOs.



FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

MATERIAL DE APOYO

(SQL)

C U R S O :

BASES DE DATOS EN INFORMIX

21 - JUNIO - 1994

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. tel.: 521-40-20 Apdo. Postal M-2285

EL LENGUAJE SQL

SQL (*Structure Query Languaje*) es un lenguaje estandar que podemos definir en dos categorias:

1 Lenguaje de definición de datos (DDL)

2 Lenguaje de manipulación de datos (DML)

3 Lenguaje de control de datos (DCL)

Características:

- Es un 4GL
- Es un estandar
- No incluye ninguna referencia fisica a los datos que accesa
- Es un camino sencillo para obtener y manipular información de una Base de Datos
- Puede ser utilizado interactivamente
- Puede ser utilizado desde un lenguaje de tercera generación
- Todas las herramientas de explotación de la Base de Datos utilizan una interface de SQL
- Incluye instrucciones para administrar y definir la Base de Datos

Utilizando SQL

Para poder hacer uso del lenguaje de consulta SQL, se debe de elegir la opción del Main Menu **Query-Language**.

Posteriormente apareceran las siguientes opciones:

New

Run

Modify

Use-editor

Output

Choose

Save

Drop

Exit

Después seleccionaremos la opción de **New** y apareceran un editor de línea, en el cual haremos las consultas.

Una vez que se ha introducido la consulta presionar **ESC** para regresar al menú anterior, donde el cursor estará posicionado en la opción de **RUN**, esto es la consulta está lista para ejecutarse con solo presionar ENTER.

Si la consulta tuvo éxito ISQL regresará el resultado de la consulta; en caso contrario mandará el control al menú de **Modify**, en el cual se podrá corregir la consulta.

Para salir de este menú elegir la opción de **ESC seguido de EXIT.**

Creación de la Base de Datos

La creación de la Base de Datos, también se puede llevar a cabo desde SQL. El comando utilizado es:

CREATE DATABASE nombre_de _la _base_de_datos

La Base de Datos se creará con un tamaño por dedault asignado por el DBMS de que se trate. La opción para indicar un tamaño en especial, es sintáxis propia de los diferentes DBMS´s.

Creación de Tablas

Las tablas también se pueden crear desde SQL:

```
CREATE TABLE nombre (
        column_name datatype [NOT NULL],
        ...)
```

- Por default las columnas se crean como NULL

Ejemplos:

```
CREATE TABLE clientes (
cve_cli         numeric not null,
nombre          char(25),
dir             char(25),
tel             char(10)
)

CREATE TABLE peliculas (
cve_pel         numeric not null,
titulo          char(25),
genero          char(12),
cve_costo       char,
precio          numeric
)

CREATE TABLE copias (
cve_copia       numeric not null,
cve_pel         integer not null,
formato         char(5)
)

CREATE TABLE costos (
cve_costo       char not null,
costo           numeric
)

CREATE TABLE renta (
cve_copia       numeric,
cve_pel         numeric,
cve_cli         numeric,
fecha           date not null,
fecha_dev       date,
estado          char
)
```

Creación de Indices

Sobre una tabla se pueden crear indices, los cuales pueden ser:

UNIQUE	no permite valores duplicados en una columna
CLUSTER	fisicamente, los registros son ordenados en la misma secuencia que los indices

La sintáxis es :

CREATE [UNIQUE] [CLUSTER] INDEX index_name
ON nombre_tabla (nombre_columna [ASC | DESC], ...)

- Se pueden incluir hasta 8 columnas en un indice compuesto

- La longitud de las columnas indexadas no puede exceder los 120 bytes

Ejemplo:

```
CREATE UNIQUE INDEX  indi_cve_pel
ON peliculas (cve_pel)
```

Inserción de Datos

La instrucción INSERT permite insertar datos en una tabla ya existente.

- Los valores en las columnas se deben especificar en el orden en como se definierón las columnas al crear la tabla.

- Si solamente se especifican algunos valores, los no especificados toman valores nulos.

- Por la consideración anterior, para todas las columnas que no acepten nulos se debe indicar un valor.

- Los valores indicados deben ser del mismo tipo de la columna que afectan.

- Los valores de tipo CHAR y DATE deben ser encerrados entre comillas.

Sintáxis:

```
INSERT [into] nombre_table
        [(lista_de_columnas)]
        { values (lista_valores) | bloque_select}
```

Ejemplos:

```
INSERT into clientes
        values(100, 'J. Antonio Chavez', 'Almaraz 2-1','1234567')


INSERT into clientes (nombre, cve_cli, dir, tel)
        values('C. Jessica Briseño C.', 101, 'Norte 86 37',
                '7654321')


INSERT into clientes
        values(102, 'Edwin Navarro Pliego', NULL, NULL)


INSERT into clientes (cve_cli, nombre)
        values(103, 'Ramón Ramírez H.')
```

Proyección de Columnas

La instrucción SELECT es la más utilizada para llevar a cabo consultas sobre la Base de Datos. Una de las funciones básicas que lleva a cabo es la proyección de columnas de una tabla.

La sintáxis simplificada de SELECT que permite la proyección de columnas es:

SELECT lista_columnas
FROM nombre_tabla

El orden de las columnas en la instrucción determina el orden de estas en el resultado

Ejemplo: Listar todas las peliculas existentes

```
select cve_cli, nombre
    from clientes
```

cve_cli	nombre
105	Sandra Sosa Aragón
107	José Guevara Briones
101	C. Jessica Briseño C.
103	Ramón Ramírez H.
106	Juan C. Pacheco R.
102	Edwin Navarro Plego
100	J.Antonio Chávez F.

Registros Duplicados

La palabra **DISTINCT** permite eliminar registros repetidos

Ejemplos:

```
SELECT  genero                    /* Obtiene los generos    */
        FROM peliculas            /* de todas las peliculas */


SELECT  distinct genero           /* Obtiene los generos de */
        FROM peliculas            /* peliculas              */
```

Selección de Registros

Con ayuda de la palabra WHERE dentro de la instrucción SELECT, podemos condicionar los registros que se desean obtener como resultado.

Sintáxis simplificada:

SELECT lista_columnas
FROM nombre_tabla
WHERE expresión

La expresión puede involucrar:

- Operadores de comparación
- Rangos (BETWEEN y NOT BETWEEN)
- Patrones de caracteres (LIKE y NOT LIKE)
- Valores desconocidos (NULL y NOT NULL)
- Listas de valores (IN y NOT IN)
- Operadores lógicos (AND y OR)

- El operador NOT niega una expresión lógica

Operadores de Comparación

Los operadores de comparación son los siguientes:

=	igual
!= ó <>	diferente
>	mayor que
<	menor que
>=	mayor o igual
<=	menor o igual

En campos tipo CHAR los operadores >, >=, < y <= comparan lexicográficamente las cadenas

En campos tipo DATE los operadores comparan tiempos

Ejemplos:

```
SELECT titulo                    /* obtiene las peliculas */
        FROM peliculas           /* de comedia */
        WHERE genero='comedia'
```

```
SELECT titulo                    /* obtiene las peliculas */
        FROM peliculas           /* no sean de comedia */
        WHERE genero!='comedia'
```

```
SELECT nombre, dir               /* obtiene el nombre y la */
        FROM clientes            /* direccion de los clientes */
        WHERE cve_cli>120        /* cuya clave sea mayor a 120 */
```

Rangos

La palabra BETWEEN en la instrucción SELECT permite especificar rangos de valores.

Ejemplos:

```
/* Obtiene el nombre y dirección de los clientes con clave */
/* entre 120 y 150      */

SELECT nombre, dir
        FROM clientes
        WHERE cve_cli between 120 and 150


/* Obtiene el nombre y dirección de los clientes cuya clave */
/* no esta entre 120 y 150      */

SELECT nombre, dir
        FROM clientes
        WHERE cve_cli not between 120 and 150
```

Patrones de Caracteres

La palabra LIKE en la instrucción SELECT permite especificar valores que cumplen con un patrón. Se utilizan en campos tipo CHAR o DATE.

Los metacaracteres o *wildcars* para LIKE son:

%	especifica cero o más caracteres
_	especifica un caracter

propios de ISQL:

[..]	especifica un caracter en un rango [a-z] un caracter de la **a** a la **z**
*	especifica uno o más caracteres
?	especifica un caracter

Ejemplos:

```
SELECT nombre                          /* clientes cuyo nombre  */
        FROM clientes                  /* comience con A, B O C */
        WHERE nombre like '[ABC]%'

SELECT nombre                          /* clientes cuyo nombre no */
        FROM clientes                  /* comience con A, B O C   */
        WHERE nombre not like '[ABC]%'
```

Listas de Valores

La palabra IN en la instrucción SELECT permite especificar una lista de valores para una columna.

Ejemplos:

```
SELECT nombre                          /* Obtiene las peliculas  */
      FROM peliculas                   /*de comedia y de terror    */
      WHERE genero IN ('comedia', 'terror')
```

Operadores Lógicos

Los operadores lógicos sirven para unir expresiones.

- Con AND la expresión es verdadera si las expresiones que involucra lo son

- Con OR la expresión es verdadera si alguna de las expresiones que involucra es verdadera

La prescedencia de los operadores es:

AND
OR
NOT

La presedencia se puede cambiar incluyendo parentesis ( )

Ejemplos:

```
/* Obtiene el nombre y dirección de los clientes con clave */
/* entre 120 y 150                                          */

SELECT nombre, dir
        FROM clientes
        WHERE cve_cl >= 120 and cve_cli <= 150


/* Obtiene las peliculas cómicas de costo A */

SELECT nombre
        FROM peliculas
        WHERE genero = 'comica' AND cve_costo = 'A'


/* Obtiene las peliculas cómicas o de costo A */

SELECT nombre
        FROM peliculas
        WHERE genero = 'comica' OR cve_costo = 'A'
```

/* Obtiene las peliculas cómicas cuya clave este entre 315 y 400 o las peliculas de vaqueros */

```
SELECT nombre
        FROM peliculas
        WHERE genero = 'comedia' AND
                clave_pel >= 315 AND clave_pel <= 400
                OR genero = 'western'
```

/* Obtiene las peliculas cómicas o aquellas cuya clave este entre 315 y 400 y las peliculas de vaqueros */

```
SELECT nombre
        FROM peliculas
        WHERE (genero = 'comedia' OR
                clave_pel >= 315 AND clave_pel <= 400)
                AND genero = 'western'
```

Renombrando Columnas

En los resultados se permite utilizar otro nombre para las columnas listadas en la instrucción SELECT.

```
SELECT nombre "Nombre Cliente", dir "Dirección"
    FROM clientes
```

Nombre del Cliente	Dirección
Sandra Sosa Aragón	Valle de Guadaiana 272
José Guevara Briones	San Angel 354
C. Jessica Briseño C.	Norte 86 37
Ramón Ramírez H.	Marina Nacional 435
Juan C. Pacheco R.	Apicultura 98
Edwin Navarro Plego.	Sur 33
J.Antonio Chávez F.	Almaraz 2-1

```
SELECT 'Cliente', nombre, dir
    FROM clientes
```

Cliente	Nombre	Dir
Cliente	Sandra Sosa Aragón	Valle de Guadaiana 272
Cliente	José Guevara Briones	San Angel 354
Cliente	C. Jessica Briseño C.	Norte 86 37
Cliente	Ramón Ramírez H.	Marina Nacional 435
Cliente	Juan C. Pacheco R.	Apicultura 98
Cliente	Edwin Navarro Plego.	Sur 33
Cliente	J.Antonio Chávez F.	Almaraz 2-1

Operadores Númericos

Los operadores aritméticos permitidos son:

+
-
*
/

Pueden ser utilizados en expresiones números en la lista de columnas del SELECT o bien en WHERE

Valores Nulos

- Un valor NULL implica un valor desconocido:
- Un valor NULL no implica cero o una cadena vacía
- Un valor NULL no es igual a otro valor NULL
- Para seleccionar registros con valores NULL en algúna columna se utiliza IS NULL
- Operaciones que involucran NULL dan como resultado NULL

Ejemplos:

```
SELECT 1500 + NULL
```

NULL

```
SELECT nombre                    /* Obtiene peliculas     */
       FROM peliculas            /* de genero desconocido */
       WHERE genero IS NULL
```

LABORATORIO

1. Liste toda la información sobre los empleados.

2. Liste el nombre y salario todos los empleados.

3. Genere un listado que tenga como encabezados "EMPLEADO" y "PERCEPCION TOTAL" ; donde la Percepción Total será el salario más la comisión.

4. Liste los diferentes tipos de puestos (trabajos) que existen.

5. Liste la información de todos los empleados del departamento 30.

6. Liste el nombre y salario de los "CLERK".

7. ¿Qué empleados perciben más comisión que salario?

8. Liste los "SALESMAN" del departamento 30 que tengan un salario mayor a 1500.

9. Liste aquellos empleados que sean "MANAGER" ó aquellos que ganan más de 3000.

10. Qué empleados son "MANAGER" ó que empleados del departamento 10 son "CLERK".

11. ¿Qué empleados del departamento 10 no son "MANAGER" ni "CLERK".

12. ¿Qué empleados ganan entre 1200 y 1400?

13. Liste los empleados cuyos nombres comiencen con M.

SELECT/ ORDER BY

La claúsula ORDER BY en la instrucción SELECT permite ordenar el resultado de la consulta.

- El ordenamiento es ascendente por default
- Los null se consideran al principio

Sintáxis simplificada:

```
SELECT [DISTINCT] lista_columnas
    FROM nombre_tabla
    [WHERE expresion]
    [ORDER BY {columna | expresion} [asc | desc] [,...]]
```

Ejemplos:

```
SELECT nombre                    /* Lista de clientes ordenada */
    FROM clientes                /* nombre                     */
    ORDER BY nombre

SELECT titulo                    /* Obtiene una lista de       */
    FROM peliculas               /* peliculas comedia ordenada*/
    WHERE genero='comedia'
    ORDER BY titulo
```

Funciones Agregadas

Las funciones agregadas permitidas son:

SUM	calcula la suma de los valores en una columna
AVG	calcula el promedio de los valores de una columna
MAX	obtiene el valor máximo en una columna
MIN	obtiene el valor mínimo en una columna
COUNT	calcula el número de registros

- Las funciones agregadas ignoran los valores NULL (excepto count(*)).
- Sum y avg sólo trabajan con valores numericos.
- Solamente se regresa un valor como resultado.
- Pueden aplicarse a todos los registros de una tabla o a un subconjunto con ayuda de WHERE.
- La palabra DISTINCT es permitida en las funciones sum, avg y count.
- La palabra DISTINCT es utilizada solamente con nombres de columnas.
- Se puede utilizar más de una función agregada en una instrucción SELECT

Ejemplos:

```
SELECT count(*)                 /* Obtiene el número de */
    FROM peliculas                      /* peliculas    */


SELECT max(precio)                      /* Obtiene la pelicula  */
    FROM peliculas                      /* más cara             */


SELECT min(costo)               /* Obtiene el precio más */
    FROM costos                 /* alto por una renta    */


SELECT avg(precio)              /* Obtiene el precio    */
    FROM peliculas                      /* promedio de las peliculas*/


SELECT avg(costo)               /* Obtiene el precio    */
    FROM costos                 /* promedio de una renta */


SELECT max(precio)                      /* Obtiene la pelicula  */
    FROM peliculas                      /* de comedia más cara  */
    WHERE genero='comedia'
```

```
SELECT min(costo), max(costo)     /* Obtiene el precio más */
        FROM costos               /* alto y el más bajo    */
                                  /* por una renta         */

SELECT count(*)                   /* Obtiene el número de  */
        FROM copias               /* copias de la película */
        WHERE cve_pel = 312       /* con clave 312         */
```

Agrupación de Registros

- La clausula GROUP BY en la instrucción SELECT permite agrupar registros.
- Generalmente es utilizada en combinación con una función agregada.
- Todos los valores NULL son tratados como un grupo.
- La clausula WHERE se lleva a cabo antes de formar los grupos.

Sintáxis simplificada:

```
SELECT [DISTINCT] lista_columnas
        FROM nombre_tabla
        [WHERE  expresión]
        [GROUP BY  expresión]
        [ORDER BY ...]
```

Ejemplos:

```
SELECT genero, avg(precio)          /* Mal uso. Todas los  */
        FROM peliculas              /* generos aparecen con*/
                                    /* el mismo precio     */
                                    /* promedio            */

SELECT genero, avg(precio)          /* Uso correcto        */
        FROM peliculas
        GROUP BY genero

SELECT genero, avg(precio)          /* Lista ordenada de   */
        FROM peliculas              /* promedios de peliculas*/
        GROUP BY genero             /* por genero          */
        ORDER BY avg(precio)
```

SELECT GROUP BY/HAVING

La clausula HAVING condiciona a los grupos que se generan como resultado. La condición se aplica después de que se han formado los grupos.

Sintáxis simplificada:

```
SELECT [DISTINCT] lista_columnas
        FROM nombre_tabla
        [WHERE  expresión]
        [GROUP BY  expresión]
        [HAVING expresion]
        [ORDER BY ...]
```

Ejemplos:

```
SELECT genero, avg(precio)          /* Obtiene una lista  */
        FROM peliculas              /* del promedio de    */
        GROUP BY genero             /* precio de las peli-*/
        HAVING avg(precio) > 100    /* culas por genero   */
                                    /* para los generos   */
                                    /* cuyo promedio es   */
                                    /* mayor a 100        */

SELECT genero, avg(precio)          /* Igual que la anterior */
FROM peliculas                      /* pero la lista es      */
GROUP BY genero                     /* ordenada              */
        HAVING avg(precio) > 100
        ORDER BY avg(precio)
```

LABORATORIO

1. Liste a los empleados ordenados por departamento

2. ¿Cuál es el salario promedio de los empleados?

3. ¿Cuá es el salario más alto que se paga a un empleado?

4. Liste el salario promedio por tipo de trabajo.

5. ¿Cuál es el puesto en donde en promedio se gana un mayor salario?

6. Listar el salario promedio de los puestos que tienen más de dos empleados.

7. Obtener un listado que contenga por departamento, los tipos de trabajo que existen y el salario promedio de cada uno de estos; indicando cuantos empleados existen en cada uno de los tipos de trabajo.

JOINS

El JOIN es la operación más importante en el modelo relacional de Bases de Datos.

Un JOIN permite obtener información de más de una tabla.

El JOIN es una selección sobre un producto cruz.

Para poder resolver una consulta mediante un JOIN se debe determinar:

- Tablas involucradas
- Columnas de relación
- Condición de selección

El resultado de un producto cruz contiene el nombre de todas las columnas de ambas tablas identificadas. El nombre de las columnas es de la forma: TABLA.NOMBRE_COLUMNA.

El resultado del producto cruz son nxm registros, donde m es el número de resgistros de la primera tabla y n el de la segunda.

Ejemplo:

Listar las peliculas y su costo de renta.

Para este caso las tablas involucradas son: PELICULA y COSTO

Las columnas de relación son CVE_COSTO en PELICULA y CVE_COSTO en COSTO.

La condición es que CVE_COSTO en PELICULA sea igual a CVE_COSTO en COSTO.

La consulta se expresaría como:

```
select titulo, costo
        from PELICULA, COSTO
        where PELICULA.cve_costo = COSTO.cve_costo
```

Condiciones para Joins

El JOIN se puede involucrar n tablas.

El nombre de la columna en el SELECT debe ir precedido por el nombre de la tabla si existe ambiguedad, por ejemplo, para dos columnas con el mismo nombre de diferentes tablas.

Las columnas involucradas en la condición del JOIN deben de ser de tipos compatibles.

Los valores NULL no participan en un JOIN.

Las columnas participantes en la condición del JOIN no necesariamente deben de aparecer en el resultado.

Se pueden incluir más condiciones de selección u otros comandos, por ejemplo ORDER BY, GROUP BY, etc.

La condición del JOIN no necesariamente debe de ser de igualdad, puede involucrar cualquier operador: !=, >, <, etc.

Ejemplos:

Listar las peliculas cuyo costo de renta sea mayor a 6.00:

```
select titulo, costo, PELICULA.cve_costo
        from PELICULA, COSTO
        where PELICULA.cve_costo = COSTO.cve_costo
        and costo > 6.00
```

Listar las peliculas que tienen más de cinco copias:

```
select titulo, count(*)
        from PELICULA, COPIAS
        where PELICULA.cve_pel = COPIAS.cve_pel
        group by titulo
        having count(*) > 5
```

Listar las peliculas rentadas por cliente indicando la fecha de renta.

```
select cve_cli, titulo, fecha
        from CLIENTES, PELICULA, RENTA
        where CLIENTES.cve_cli = RENTA.cve_cli
        and   RENTA.cve_pel = PELICULA.cve_pel
```

Alias y Selft JOIN

Para abreviar la escritura se pueden utilizar alias para las tablas participantes en un JOIN:

```
select cve_cli, p.cve_pel, titulo, fecha
        from CLIENTES c, PELICULA p, RENTA r
        where c.cve_cli = r.cve_cli
        and   r.cve_pel = p.cve_pel
```

Un SELFT JOIN es un JOIN en donde solamente participa una tabla.

En un SELFT JOIN es necesario utilizar alias.

Ejemplo:

Considere la siguiente tabla:

CIUDADANO(cve, nombre, direccion, tel, cve_conyuge)

Para listar el nombre de los ciudadanos y el de su conyuge:

```
select a.nombre, b.nombre
        from CIUDADANO a, CIUDADANO b
        where a.cve_conyuge = b.cve
```

LABORATORIO

1. ¿En donde trabaja Allen?

2. Listar el nombre de los empleados y de su departamento.

3. Listar todos los empleados que trabajan en CHICAGO.

4. ¿Qué empleados ganan más que JONES?

5. Obtener un listado que tenga como encabezados: EMPLEADO, JEFE ; las cuales contendrán el nombre de los empleados y el de su jefe, respectivamente.

6. ¿Cuántos empleados tiene a su cargo cada jefe?

Subconsultas

Muchas consultas intuitivamente se pueden resolver como subconsultas en lugar de resolverse por JOIN.

Es más fácil pensar en una solución mediante una subconsulta que con un JOIN.

Ejemplo:

¿Qué peliculas tienen un costo de 6.00?

Utilizando JOINS:

```
select titulo
        from PELICULAS, COSTOS
        where PELICULAS.cve_costo = COSTOS.cve_costo
        and costo = 6.00
```

Utilizando subconsultas:

```
select titulo
        from PELICULAS
        where cve_costo =
                (select cve_costo
                        from COSTOS
                        where costo = 6.00)
```

Una subconsulta es una sentencia SELECT utilizada en una expresión como parte de otra sentencia SELECT.

La subconsulta es resuelta y el resultado es sustituido en la consulta externa.

La columna involucrada en la selección de la consulta externa debe de ser de un tipo similar a la columna proyectada en la subconsulta.

La subconsulta no puede proyectar como resultado más de una columna.

El resultado de una subconsulta puede provenir de una función agregada.

El único resultado que aparece como salida es el que genera la consulta externa.

Ejemplos:

¿En que fechas ha rentado peliculas Jéssica Briseño?

```
select fecha
        from RENTA
        where cve_cli =
        (select cve_cli
                from CLIENTES
                where nombre = "Jéssica Briseño")
```

¿Cuál es la pelicula más cara de terror?

```
select titulo
        from PELICULAS
        where precio =
                (select max(precio)
                        from PELICULAS
                        where genero = "terror")
```

¿Qué peliculas cómicas tienen un precio mayor que cualquier película de terror?

```
select titulo
        from PELICULAS
        where precio =
                (select max(precio)
                        from PELICULAS
                        where genero = "terror")
        and genero = "comedia"
```

¿Cuál es la clave de la última película rentada?

```
select cve_pel
        from RENTA
        where fecha =
        ( select max(fecha)
                from RENTA )
```

Una consulta puede contener varios niveles de subconsultas.

Cuando una subconsulta genera un sólo resultado, se pueden utilizar los siguientes operadores en la consulta externa:

=, !=, >, >=, <, <=

Si se utilizan los operadores anteriores y la subconsulta genera como resultado varios registros, existe un error

Cuando la subconsulta genera varios registros, se pueden utilizar los operadores "in" y "not in" en la consulta externa.

Ejemplos:

¿Qué películas tienen copias en formato beta?

```
select titulo
        from PELICULAS
        where cve_pel in
        ( select cve_pel
                from COPIAS
                where formato = "beta")
```

¿Cuales son las películas para las que no hay copias en formato beta?

```
select titulo
        from PELICULAS
        where cve_pel not in
        ( select cve_pel
                from COPIAS
                where formato = "beta")
```

¿Qué peliculas se rentaron en el pasado mes de febrero?

```
select distinct titulo
        from PELICULAS
        where cve_pel in
        (select cve_pel
                from RENTA
                where fecha >= "1/2/1994"
                        and fecha <= "28/2/1994")
```

¿Qué peliculas ha rentado Jéssica Briseño?

```
select distinct titulo
        from PELICULAS
        where cve_pel in
        (select cve_pel
                from RENTA
                where cve_cli =
                (select cve_cli
                        from CLIENTES
                        where nombre = "Jéssica Briseño"))
```

¿En que fecha debe devolver Jéssica Briseño la película "Los gritos del silencio"?

```
select fecha_dev
        from RENTA
        where cve_pel =
        (select cve_pel
                from PELICULAS
                where titulo = "Los gritos del silencio")
        and cve_cli =
        (select cve_cli
                from CLIENTES
                where nombre = "Jéssica Briseño")
```

¿En que formato rento la película "Los gritos del silencio" Jéssica Briseño el 8/12/93?

```
select formato
        from COPIAS
        where cve_copia =
        ( select cve_copia
                from RENTA
                where cve_pel =
                (select cve_pel
                        from PELICULAS
                        where titulo = "Los gritos del silencio")
                and cve_cli =
                (select cve_cli
                        from CLIENTES
                        where nombre = "Jéssica Briseño")
                and fecha = "8/12/93")
        and cve_pel =
        (select cve_pel
                from PELICULAS
                where titulo = "Los gritos del silencio")
```

LABORATORIO

1. ¿Qué empleados tienen el mismo puesto de JONES?

2. ¿Qué empleados ganan más que algún ampleado del departamento 30?

3. ¿Qué empleados ganan más que cualquier empleado del departamento de SALES?

4. ¿Qué empleados tienen el mismo puesto y salario que FORD?

5. ¿Qué empleados tienen el mismo puesto que JONES o un salario mayor o igual al de FORD?

6. ¿Qué empleados del departamento 10 tienen el mismo puesto que algún empleado del departamento de SALES?

7. ¿Qué empleados tienen el mismo puesto que algún empleado de Chicago?

9. ¿Qué empleado es el que gana más?

10. ¿Quién es el empleado más joven?

11. ¿Qué empleado es el que gana menos?

Modificación de Datos

La instrucción UPDATE permite la modificación de las columnas de los renglones de una tabla.

La sintáxis es la siguiente:

```
update table_name
set columna = { expresion }
  [, columna = {expresion }] ...
   [ where condiciones ]
```

No se pueden modificar varias tablas con una sola instrucción UPDATE.

Ejemplos:

```
UPDATE emp set salary=2000
   where ename = "SMITH"

UPDATE COSTOS set costo = costo * 1.2

UPDATE CLIENTES set direccion = "Av. Centenario 12543",
                    tel = "367-82-82",
                        where cve_cli = 123
```

Borrado de Datos

La instrucción DELETE permite el borrado de registros dentro de una tabla.

La sintáxis es la siguiente:

```
delete from table_name
[where condiciones]
```

Si no se indica la claúsula where se borran todos los registros de la tabla.

Ejemplos:

```
DELETE from RENTA
        where fecha_dev < "1 Mar 1994"

DELETE from COPIAS
        where cve_pel = 38

DELETE from CLIENTES   /* Cuidado borra toda la información */
```

Borrado de Tablas

Para borrar una tabla (definición y datos) se utiliza la instrucción DROP TABLE.

La sintáxis es la siguiente:

DROP TABLE table_name [, table_name]

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

BASES DE DATOS EN INFORMIX

P E R F O R M

JUNIO 1994

Perform

Menú de Form

Perform es un program de ISQL, con el cual se pueden generar formas para captura de datos, consultas, validaciones, etc. Esto se puede realizar por medio del menú de **Form**, que se encuentra en el menú principal, el cual tiene las siguientes opciones:

Run. Se utiliza para ejecutar una forma

Modify. Se utiliza para hacer modificaciones a la forma

Generate. Con esta opción se puede generar un forma por default.

New. Crea una nueva forma

Compile. Esta opción se utiliza cuando ha ocurrido alguna modificación y se desea ver reflejada en la forma

Drop. Se utiliza para borrar un forma

Exit. Salir del menú de **Form**

ISQL nos proporciona una **forma por default**, la cual tiene las siguientes características:

- Contiene todas las columnas de la tabla sobre la cual se está generando la forma.
- Las columnas aparecen en el orden como se dió la definición de la tabla.
- Cada columna se encuentra en una línea.

- El nombre de la columna se encuentra justificado a la izquierda, seguido de corchetes [ ].

- Los corchetes [ ] nos indican que ahí se desplegará la información; además tiene un ancho, el cual corresponde con la definición de las columnas.

Una vez que se haya generado una forma se pueden realizar operaciones como inserción de datos, consultas, modificación de datos, etc.; esto se puede realizar con la opcion **Run** del menú de **Form**, el cual nos proporciona las siguientes opciones:

Query. Se pueden consultar la información que se tiene en esta tabla, registro por registro, en base a los siguientes operadores:

Operador	Significado	Tipo de Dato	Sintaxis
=	Igualdad	Todos	= X
>	mayor que	Todos	> X
<	menor que	Todos	>= X
>=	mayor o igual	Todos	>= X
<=	menor o igual	Todos	<= X
<>	diferente	Todos	<> X
|	OR	Todos	X | Y
:	Rango	Todos	? X,caracteres
?	sustituye un caracter	CHAR	* X,caracteres
*	sustituyecero o más caracteres	CHAR	

Next. Despliega el siguiente registro de una consulta

Previous. Despliega el registro anterior de una consulta

Add. Se agrega (inserta) un registro sobre la tabla

Update. Se puede modificar el registro desplegado

Remove. Se borra el registro desplegado

Table. Cambiar la tabla activa

Screen. Despliega más información sobre la forma, esto si la forma tiene más de 20 renglones.

Current. Despliega más información sobre el registro (además de la que mostró anteriormente)

Master. Esta opción despliega directamente la tabla activa MASTER que proviene de la tabla DETAIL (si existe alguna dentro de la forma)

Detail. En la sección de INSTRUCCIONS se pueden incluir una ó mas relaciones Master/Detail (maestro/esclavo) para joins que tienen una asociación de **1:M**. Esta opción automáticamente selecciona, despliega y hace consultas de la tabla activa DETAIL (si existe alguna dentro de la forma).

Output. Manda la información de una consulta a un archivo

Exit. Salir del menú, regresa al menú de **Form**

Archivo de especificaciones de una forma

Un archivo de especificaciones de una forma es un archivo en ASCII. Por ejemplo una forma por default tiene las siguientes especificaciones:

```
database ejemplo
screen
{
no_clien      [f000 ]
nombre        [f001         ]
apellidos     [f002                 ]
direccion     [f003                 ]
colonia       [f004         ]
delegacion    [f005]
cp            [f006 ]
telefono      [f007         ]
}
end

tables
cliente

attributes
f000 = cliente.no_clien
f001 = cliente.nombre
f002 = cliente.apellidos
f003 = cliente.direccion
f004 = cliente.colonia
f005 = cliente.delegacion
f006 = cliente.cp
f007 = cliente.telefono
end
```

Estructura de un archivo de especificaciones de una forma

▸ Database	Identifica la base de datos sobre la cual se realizó la forma
▸ Screen	Esta sección es la siguiente en aparecer y muestra exactamente el plan de la forma, como se quiere que esta aparezca en la pantalla. Si la forma tiene varias pantallas, esta sección incluye el plan para cada una de ellas, una tras otra.
▸ Tables	Lista todas las tablas a las que hace referencia la forma. Esta sección identifica las tablas cuyas columnas aparecen en la forma.
▸ Attributes	Esta sección describe cada campo sobre la forma. Incluyendo, por ejemplo, la apariencia, valores de entrada aceptables, comentarios y valores por default.
▸ Instruction	Esta sección es opcional para cálculos avanzados y características especiales como especificaciones de la relación master-detail (maestro-esclavo), joins compuestos, delimitadores and control de bloques.

Usando la palabra reservada **END** se puede marca el fin de cada sección en el archivo de especificaciones de una forma (es opcional, pero se recomienda).

Sección Database

La sección DATABASE de un archivo de especificación de una forma identifica la base da datos con la cual la forma es asignada para trabajar.

Su estructura de esta sección es la siguiente:

DATABASE nombre_base_de_datos [WITHOUT NULL INPUT]

donde :

DATABASE es una palabra reservada para definir esta sección
WITHOUT NULL INPUT es opcional y se utiliza solamente si se esta trabajando con una base de datos que permite valores NULL.

Por ejemplo:

database
cliente

Sección SCREEN

La seccion SCREEN describe como aparecerá la forma en la pantalla. Esta sección puede contener varias pantallas. Cada pantalla debe estar precedida por la palabra reservada *SCREEN* y encerrada entre llaves *{ }*. Cada pantalla esta formada por un arreglo de *display fields* y información textual como títulos y etiquetas.

Los *display fields* son indicados entre corchetes *[ ]* los cuales definen la longitud del campo y tienen un *field tag* que indentifica al campo.

Si se tienen dentro de una pantalla { } más de 20 líneas , FORMBUILD desplegará esta pantalla en dos paginas la segunda empezando apartir de la línea 21.

La sintaxis de esta sección sería la siguiente:

```
SCREEN
{

    display fields

}
SCREEN
{

    display fields

}
```

Display Fields

Los display fields indican donde va a desplegarse la información relacionada con algún campo, usando los corchetes [ ]. Cada campo tiene una etiqueta asociada (field tag) que identifica al campo en la sección de ATTRIBUTES y en la sección de INSTRUCTIONS.

Su sintaxis es la siguiente:

text [field-tag]

text	es el texto que se quiere desplegar en la pantalla.
[ ]	son los delimitadores para el campo. El ancho del campo es el número de espacios que estan entre los corchetes.
field-tag	es un identificador usado para el despliegue del campo.

```
screen
{

                    ALTA DE CLIENTE

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

        Número de cliente: [c1     ]

        Nombre de cliente: [c2              ]    Apellidos:   [c3              ]

        Dirección:   [c4                    ]    colonia:     [c5              ]

        delegacion:  [c6                    ]    cp:          [c7     ]

        Telefono:    [c8

--------------------------------------------------------------------------------

}

end
```

Sección TABLES

Esta es la tercera sección de un archivo de especificaciones, aquí se listan todas las tablas de donde pertececen todas las columnas que aparecen en la forma.

El número de tablas que pueden usarse en una forma depende de la máquina. En UNIX pueden tenerse hasta 12 tablas abiertas al mismo tiempo.

La sintaxis es la siguiente:

TABLES tabla1 tabla2 ...

Sección ATRRIBUTES

En esta sección se describe el funcionamiento y la apariencia de cada uno de los campos definidos en la sección de SCREEN. Se pueden utilizar atributos que describen como PERFORM podría desplegar el campo, especificar un valor por default, el valor limite que puede aceptar un campo y un conjunto de parámetros que se describiran en esta sección.

El orden en el cual los campos son descritos en esta sección, determina el orden por default que el cursor ira llevando en la forma. Los campos que se encuentran en la sección de SCREEN no tienen que estar asociados con alguna columna de las tablas involucradas. Un campo que no esta asociado con una columna es llamado un *display only field*. La sección ATTRIBUTES contiene dos tipos de ligado: unos cuando los *field tags* se ligan con las columnas de las tablas involucradas, y otra cuando los *field tags* se ligan con un *display only field*.

NOTA: en esta sección cada expresión al final llevan punto y coma ;

Display Field Order

Cuando se utilizan las opciones del menú de Form como Query, Add y Update el cursor avanza por default campo por campo de la tabla que se encuentra activa esto deacueredo con el orden en el cual los field tags aparecen en la sección de ATTRIBUTES. El cursor tiene una trayectoria circular, lo cual quiere decir que va avanzando a través de todos los campos y regresa al primer campo para seguir de nuevo esta trayectoria. El orden por default puede ser cambiado con el uso de los *conbtrol blocks*, los cuales se veran en la sección INSTRUCTIONS.

Table Order

Cuando una forma contiene campos correspondientes a diversas tablas, PERFORM pone las tablas en una lista ordenada. Con la opción Table del menú de Form , PERFORM indica la tabla activa, la cual puede ser cambiada por medio de esta opción, en base a la lista ordena que genera. PERFORM genera la lista ordenada de las tablas de acuerdo al orden de aparición de las columnas en la sección de ATTRIBUTES.

Ligado de los Fields con las columnas de las tablas

Existen dos tipos de campos que son ligados con las columnas en la sección ATRRIBUTES: los que aceptan datos de la entrada y los que no. Los campos que no permiten datos desde el teclado son llamados *lookup fields*. Este tipo de campos se pueden especificar con el atributo LOOKUP que se verá más adelante. Aquí se van a tratar los campos que permiten datos de entrada.

Sintaxis:

field tag = [table.] col [, atributos];

donde:

field tag	es el field tag usado en la sección SCREEN.
table	es el nombre de la tabla a la cual corresponde la columna. El nombre de la tabla solo es necesario indicarla cuando haya dos columnas con el mismo nombre.
col	es el nombre de la columna.
attributos	es un atributo o una lista de atributos separados por comas.

Display Only Fiels

Los display only fields no estan asociados con las columnas de las tablas y aparecen solamente en la forma. Ellos reciben sus valores como el resultado de un cálculo and/or lógico basados en los valores de otros campos.

Sintaxis:

field tag = DISPLAYONLY [ALLOWING INPUT]
TYPE tipo_de_dato [NOT NULL] [, atributos];

donde:

field tag	es el field tag usado en la sección SCREEN.
DISPLAYONLY	es una palabra reservada la cual indica que el campo no corresponde a una columna de alguna tabla de la base de datos utilizada. Se puede especificar como el campo podrá recibir valores en la sección INSTRUCTIONS.
ALLOWING INPUT	son palabras reservadas que se usan para que pueda recibir datos de la entrada.
TYPE	Es una palabra reservada
tipo_de_dato	Es el tipo de dato que se va a desplegar, excepto el SERIAL.
NOT NULL	Es una palabra reservada, la cual se utiliza si el campo permite valores de entrada, el usuario forsozamente debe de dar un valor de entrada.
atributos	es uno o más atributos separados por comas.

- Si el tipo de dato es CHAR no se necesita indicar la longitud ya que está determinada por el ancho del fiel tag (se indica con los corchetes [ ]).

- Si se especifica la presición para los tipos DECIMAL o MONEY, se debe de asegurar que el ancho del field tag sea lo suficiente grande para desplegarlo.

- Cuando el campo no permite valores de entrada, solamente se pueden utilizar los siguientes atributos :

DEFAULT	DOWNSHIFT
FORMAT	QUERYCLEAR
REVERSE	RIGHT
UPSHIFT	ZEROFILL

- Cuando se especifica uno ó más fields display only que permitan valores de entrada, PERFORM junta estos campos y crea una tabla llamada **displaytable** en donde los field tags seran los nombres de las columnas. Esta tabla puede ser usada en la sección INSTRUCTIONS.

Ejemplo:

d1 = displayonly type money;

Este displayonly es utilizado para desplegar el resultado del cálculo del importe de algunas películas.

Joining entre Columnas

Una forma que contiene información de varias tablas normalmente se utiliza un campo de despligue (*join*) que une dos (o más) columnas las cuales contienen información que se relaciona.

El join se realiza con columnas del mismo tipo de dato. Si el de tipo CHAR tienen que tener la misma longitud. No se puede realizar un join con dos columnas que sean de tipo SERIAL, lo que si se podría hacer es un join con una sola columna de tipo SERIAL y la otra de ellas fuera una columna INTEGER.

Un join será representado en la sección ATTRIBUTES de la siguiente manera:

field tag = col1 = col2

Ejemplo:

```
c11 = *cliente.no _clien renta.no _clien;
```

c11 es el field tag que contendrá el join, el asterisco antes de la columna cliente.no_clien indica un tipo especial de join que veremos adelante.

- La colocación de los atributos está determinada por el efecto que se desea obtener.
- Si se quiere aplicar un atributo a pesar de no saber cual tabla se encuentre activa en el join; tanto las columnas como el atributo deberan ir en la misma linea. Esto obliga a las dos columnas a tener el mismo atributo:

  field tag = col1 = col2, atributos;

- Si se quiere aplicar diferentes atributos para cada una de las columnas en el join, las columnas iran en lineas separadas con sus respectivos atributos:

  field tag = col1, atributo1;
  = col2, atributo2;

  donde:
  atributo1 es realizado cuando la tabla que contiene la columna1 es activada
  atributo2 es realizado cuando la tabla que contiene la columna1 es activada

  Ejemplo:

```
r13 = renta.cve_pel;
    = pelicula.cve_pel, noentry, noupdate, queryclear;
```

Verify Joins

Uno puede verificar que el valor de entrada de un campo corresponda con el de la columna de la tabla correspondiente, a esta columna se le llama columna dominante. Un verify join se realiza directamente por medio del caracter asterisco (*), el cual se deberá colocar antes de la columna dominante.

Sintaxis:

*field tag = columna1 = *columna2*

PERFORM valida la entrada de cualquier valor que tome el field tag, siempre y cuando exista en la columna2.

Sintaxis de los atributos

A continuación se presentarán los atributos existentes:

- ***AUTONEXT***

  Este atributo causa el avance del cursor al siguiente campo automaticamente, cuando el ancho del campo está lleno (el ancho de cada campo esta delimitado por los corchetes [ ]).

  Este atributo es usado normalmente con campos de tipo CHAR.

  Sintaxis:

  field tag = columna, AUTONEXT;

  donde:

  field tag es el field tag usado en la sección SCREEN

  columna es el nombre de la columna de la base de datos

AUTONEXT es una palabra reservada

Ejemplo:

```
c6 = delegacion, upshift, autonext;
c7 = cp, autonext;
```

Cuando se introduce la información al campo **c6** y este campo se llena, el cursor se mueve automaticamente al comienzo del siguiente campo (en este caso el campo **c7**). De igual manera cuando se introducen 5 caracteres del codigo postal en el campo **c7**, el cursor se mueve automaticamente al comienzo del siguiente campo.

- ***COMMENTS***

Este atributo causa que PERFORM despliegue un mensaje en la línea de comentarios que se encuentra arriba de la línea de estado (abajo de la pantalla). El mensaje es desplegado cuando el cursor se mueve al campo asociado con este atributo.

El comentario solo debe abarcar una sola línea.

Este atributo es utilizado cuando se desea dar información o alguna instrucción al usuario sobre el campo asociado.

Sintaxis:

field tag = columna, COMMENTS = " mensage";

donde:

field tag	es el field tag usádo en la sección SCREEN
columna	es el nombre de la columna de la base de datos

COMMENTS es una palabra reservada

mensaje es un string encerrado entre comillas

Ejemplo:

```
c2= nombre, comments =
"Por favor solo indicar un solo nombre";
```

- ***DEFAULT***

Este atributo asigna un valor por default, el cual es desplegado en pantalla.

Sintaxis:

field tag = columna, DEFAULT = value;

donde:

field tag es el field tag usado en la sección SCREEN

columna es el nombre de la columna de la base de datos

DEFAULT es una palabra reservada.

value es el valor por default.

Ejemplo:

```
r4  = fecha, default = today,
        format = "mm/dd/yyyy";
```

- ***DOWNSHIFT***

Este atributo es utilizado para campos de tipo CHAR cuando se quiere convertir las letras mayusculas a minusculas; con esto estaremos asegurando que toda nuestra información referente a la columna relacionada con este atributo este en minusculas.

Recordar que hay diferencia entre mayuscula y minusculas en el manejo de la información.

Sintaxis:

field tag = columna, DOWNSHIFT;

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos
DOWNSHIFT	es una palabra reservada que concierte un valor de entrada CHAR a minusculas.

Ejemplo:

```
c5 = colonia, downshift;
```

- ***FORMAT***

Controla el formato de las columnas de tipo decimal, small float, float y date.

Los valores número pueden ser formateados, utilizando el simbolo de gato (#), comas y el punto decimal.

PERFORM redondea los numeros antes de desplegarlos si es necesario.

Si el formato es más pequeño que el ancho del campo FORMBUILD mandará un warning, pero la forma puede usarse.

Formato para tipo Date	**Salida**
Formato por default	09/15/1994
FORMAT = "mm/dd/yy"	09/15/94
FORMAT = "yymmdd"	940915
FORMAT = "mmm dd, yyyy"	Sep 15, 1994
FORMAT = "dd-mm-yy"	15-09-94
FORMAT = "(ddd.) mmm. dd, yyyy"	(Sat.) Sep. 15,1994

Formato paera tipos Numericos	**Salida**
FORMAT = "###.###"	234.455
FORMAT = "###,###.##"	100,234.46

Sintaxis:

field tag = columna, FORMAT = "formato";

donde:

field tag es el field tag usado en la sección SCREEN

columna es el nombre de la columna de la base de datos

FORMAT es una palabra reservada.

formato es un string que especifica el formato del dato.

Ejemplo:

```
r12 = fecha_dev, FORMAT = "mmm dd, yyyy";
r4  = fecha, default = today,
        format = "mm/dd/yyyy";
```

- ***INCLUDE***

Este atributo especifica los valores aceptables que puede tener un campo, causando que PERFORM cheque el valor de entrada antes de aceptarlo.

Si se incluyen string como valores se debem de encerrar entre comillas.

Si un campo puede aceptar valores NULL, incluir la palabra reservada *null*.

Sintaxis:

field tag = columna, INCLUDE = (lista de valores);

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos
INCLUDE	es una palabra reservada
lista de valores	son los valores que puede tomar el campo, se puede especificar valor por valor (valor1, valor2, ...) ó un rango de valores (valor1 TO valorN) ó una combinación de estos separados por comas.

Ejemplo:

```
p4 = genero,
include = ("acción", "comedia", "comicas", "suspenso"),
comments ="' El genero puede ser : acción, comedia, comicas,
suspenso";
```

- ***LOOKUP***

Este atributo es utilizado para desplegar información de otra tabla mientras se introduciendo valores ó consultando la tabla activa. Este atributo se puede utilizar para prevenir si el valor que se esta introduciendo a la tabla activa no se existe en otra tabla.

El asterisco antes de la columna tabla2.col es opcional, se indica solo si se quiere que la columna tabla1.col solamente pueda tomar valores existentes en la columna table2.col.

Sintaxis:

```
field tag = tabla1.col,     LOOKUP [field tag1 = table2.col1
                                    [, field tag2 = tabla2.col2, ...]]
                            JOINING [*] tabla2.col;
```

donde:

field tag	es el field tag usado en la sección SCREEN
tabla1.col	es el nombre de la columna perteneciente a la tabla1
LOOKUP	es una palabra reservada
field tag1	es el field tag de la columna que desplegará su valor desde el LOOKUP.
tabla2.col1	es una columna de la tabla2 cuyo valor se desplegará en el field tag1.
JOINING	es una palabra reservada que identifica el join entre las columnas
*	es un caracter opcional, el cual va con una columna dominante para realizar un verify join.
table2.col	es el nombre de la columna perteneciente a la tabla2 y es el que realiza el join con table1.col

Ejemplo:

```
r16 = renta.cve_pel,
lookup  p1 = pelicula.titulo
joining *pelicula.cve_pel;
```

- ***NOENTRY***

Este atributo se utiliza para no permitir la entrada de valores cuando la tabla a la que corresponde la columna asociada al campo es activada y se hace uso de la operación **Add**.

Sintaxis:

field tag = columna, NOENTRY;

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos
NOENTRY	es una palabra reservada que no permite datos de entrada sobre este campo durante una operación **Add**.

- ***NOUPDATE***

Este atributo no permite la entrada de información durante la modificación de un renglon haciendo uso de la operación **Update.**

Sintaxis:

field tag = columna, NOUPDATE;

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos
NOUPDATE	es una palabra reservada que no permite valor en la entrada sobre este campo durante la operación de **Update**.

Ejemplo:

```
c14 = costo.cve_costo, noentry, noupdate;
c15 = costo, costo, noentry, noupdate;
```

- ***PICTURE***

Este atributo se utiliza para especificar un patrón de caracteres para la entrada de valores sobre un campo tipo CHAR.

Para la definición del patrón se hará uso de los siguientes caracteres:

Caracter	**Significado**
A	sustituye cualquier letra
#	sustituye cualquier digito
X	sustituye cualquier caracter

Sintaxis:

field tag = columna, PICTURE = "patrón";

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos
PICTURE	es una palabra reservada
patrón	es un string que especifica el patrón que se desea que cumplan los datos de entrada.

Ejemplo:

```
c8  = telefono, picture = "###-##-##";
```

producirá el siguiente despliegue en el campo antes de introducir los valores:

Telefono: [- -]

```
p1 = cve_pel, picture = "AA###-AA(X)";
```

este ejemplo aceptará cualquiera de los siguientes valores:

LF493-BB(*)
TG385-AS(3)
YG674-ZZ(D)

- ***QUERYCLEAR***

Este atributo limpia el campo con el join en la pantalla cuando se aplica la operación de **Query**.

Este atributo no es aplicable para los campos display only fields.

Sintaxis:

field tag = columna, QUERYCLEAR;

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos
QUERYCLEAR	es una palabra reservada

Ejemplo:

```
r3 = renta.no_clien;
   = *cliente.no_clien, noentry, noupdate, queryclear;
```

Cuando la tabla cliente es activada y se realiza un query, el campo no_clien será limpiado.

- ***REQUIRED***

Este atributo forza a que se le de un valor de entrada a un campo en particular durante la operación **Add**.

Sintaxis:

field tag = columna, REQUIRED;

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos
REQUIRED	es una palabra reservada que indica que el campo no puede tener valores NULL

Ejemplo:

```
c1 = no_clien, requiered;
```

- ***REVERSE***

Este atributo se utiliza cuando se quiere que un campo aparezca en video inverso.

Sintaxis:

field tag = columna, REVERSE;

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos
REVERSE	es una palabra reservada que permite desplegar el field tag en video inverso

Ejemplo:

```
c1 = no_clien, requiered, reverse;
```

- ***RIGHT***

Este atributo se aplica a aquellos campos que se quiera hacer una justificación a la derecha.

Sintaxis:

field tag = columna, RIGHT;

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos
RIGTH	es una palabra reservada que indica la justificación a la derecha del valor de entrada sobre el field tag.

Ejemplo:

```
p3 = pelicula.clas, right;
```

- ***UPSHIFT***

Este atributo es utilizado para campos de tipo CHAR cuando se quiere convertir las letras minusculas a mayusculas; con esto estaremos asegurando que toda nuestra información referente a la columna relacionada con este atributo este en mayusculas.

Recordar que hay diferencia entre mayuscula y minusculas en el manejo de la información.

Sintaxis:

field tag = columna, UPSHIFT;

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos

UPSHIFT	es una palabra reservada.que convierte una valor de entrada caracter a mayusculas.

Ejemplo:

```
p4 = genero, downshift,
include = ("ACCION", "cCOMEDIA, "COMICAS", "SUSPENSO");
```

- ***VERIFY***

Este atributo se utiliza cuando se quiere que el usuario verifique si es el valor correcto; esto es que el usuario tendrá que dar el mismo valor dos veces para verificar.

Sintaxis:

field tag = columna, VERIFY;

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos
VERIFY	es una palabra reservada que requiere que se le de el mismo valor de entrada dos veces sobre el field tag.

Ejemplo:

```
p6 = precio, right, verify;
```

- ***ZEROFILL***

Este atributo se aplica a los campos que se quiera una justificación a la derecha y que rellene los espacios blancos a la izquierda con blancos.

Este atributo es utilizado con campos de tipo numerico.

Sintaxis:

field tag = columna, ZERIFILL;

donde:

field tag	es el field tag usado en la sección SCREEN
columna	es el nombre de la columna de la base de datos
ZEROFILL	es una palabra reservada que indica la justificación a la derecha del valor de entrada sobre el field tag y si sobran espacios en blanco rellenarlos con ceros.

Ejemplo:

```
p6 = precio, zerofill;
```

Sección INSTRUCTIONS

Esta última sección de el archivo de especificaciones de la forma es opcional. Esta sección es utilizada para:

- Establecer joins compuestos
- Especificar los delimitadores de campo
- Crear una relación master/detail (maestro/esclavo)
- Definir bloques de control

Esta sección comienza con la palabra reservada INSTRUCTIONS

COMPOSITES

Se establece un join compuesto entre dos tablas cuando los valores involucrados proviene de más de una columna (la PK esta formada por dos o más columnas) y esto hace que el renglón sea unico.

Sintaxis:

COMPOSITES < tabla1.col1, tabla1.col2[, tabla1.col3, ...] >
[*] < tabla2.col, tabla2.col2[, tabla2.col3, ...] >;

donde:

COMPOSITES	es una palabra reservada que le siguen un conjunto de columnas encerradas entre picoparéntesis < >, las cuales son tratadas como un join compuesto.
tabla1.colN	es una columna de la tabla 1.
tabla2.colN	es una columna de la tabla 2.

Ejemplo de la forma **sample**:

```
composites <items.stock_num, item.manu_code>
                * <stock.stock_num, stock.manu_code>
```

DELIMITERS

Los delimitadores son los que se usa PERFORM para encerrar los campos en la sección SCREEN.

Los delimitadores por default son [ ], pero estos pueden ser sustituidos por cualquier otro caracter, incluyecdo espacios en blancos.

Esta instrucción le dice a PERFORM el simbolo que usara como delimitador cuando este despliegue el campo en la pantalla.

Cada delimitador es un solo caracter.

Sintaxis:

DELIMITERS "ab";

donde:

DELIMITERS	es una palabra reservada.
a	es el delimitador que abre.
b	es el delimitador que cierra.

MASTER OF

Se puede crear una relación Master/Detail (Maestro/Esclavo) entre dos tablas cuando **un (1)** renglón de una tabla (Master) es asociado con **muchos (M)** renglones de la otra tabla (Detail).

La relación Master/Detail simplifica las consultas cuando existe el tipo de asociación de **1:M**

Master/Detail puede estar definida en ambas direcciones.

Sintaxis:

tabla1 MASTER OF tabla2;

donde:

tabla1	es una tabla de la base de datos en uso que será designada como la tabla master (maestro).
MASTER OF	es una palabra reservada.
tabla2	es una tabla de la base de datos es uso que será designada como la tabla detail (esclavo).

Ejemplos de la forma **sample**:

```
customer master of orders;
orders master of items;
```

Control Blocks

Loscontrol blocks son usados para las siguientes funciones:

- Controla el movimiento del cursor cuando su realizá la operación de **Add** (agregar) ó **Update** (modificar) un renglón.
- Checa el valor del dato de entrada
- Modifica el dato en los campos después de las operaciones **Add**, **Update** y **Query**.
- PERFORM hace cálculos sobre el valor de un campo y deja el resultado en otro campo.
- Despliga información calculada en las columnas.

Cada bloque de control puede ser un BEFORE control ó un AFTER control. Los controles pueden ser tomados antes ó después de que las operaciones de PERFORM se lleven a cabo. Se pueden utilizar BEFORE blocks con las operacionaes **Add**, **Update** y **Remove**. Se puede utilizar el AFTER blocks con **Add**, **Update**, **Remove** y **Query**.

BEFORE

Un BEFORE control causa que PERFORM tome una serie de acciones antes que este ejecute una operación.

Sintaxis:

```
BEFORE lista_de_opciones OF tabla/lista_columnas
        acción
        acción
          .
          .
          .
        acción
```

donde:

BEFORE	es una palabra reservada
lista_de_opciones	es una ó más palabras reservadas EDTADD, EDITUPDATE, ó REMOVE separadas por un espacioen blanco.
OF	es una palabra reservada
tabla/lista_columnas	es una lista de tablas ó columnas; pueden ser hasta 16, dependiendo del sistema operativo.
acción	es uno de cinco tipos de instrucciones para PERFORM: asignar valores a campos, mover el cursor a un campo, desplegar un mensaje, salir del menú de PERFORM ó tomar una de estas acciones dependiendo de los valores de los campos.

AFTER

Un AFTER control causa que PERFORM tome una serie de acciones después que este ejecute una operación.

Sintaxis:

AFTER lista_de_opciones OF tabla/lista_columnas

acción
acción
.
.
.
acción

donde:

AFTER	es una palabra reservada
lista_de_opciones	es una ó más palabras reservadas ADD, UPDATE, QUERY, REMOVE, DISPLAY, EDITADD y EDITUPDATE separadas por un espacioen blanco.
OF	es una palabra reservada
tabla/lista_columnas	es una lista de tablas ó columnas; pueden ser hasta 16, dependiendo del sistema operativo.
acción	es uno de cinco tipos de instrucciones para PERFORM: asignar valores a campos, mover el cursor a un campo, desplegar un mensaje ó tomar una de estas acciones dependiendo de los valores de los campos.

EDITADD, EDITUPDATE

EDITADD y EDITUPDATE nos proporcionan la habilidad para ejecutar una o más acciones antes ó después del dato de entrada durante las operaciones de **Add** y **Update** respectivamente.
La acción ocurre antes de que el renglón se escriba en la tabla.

Sintaxis:

```
{BEFORE | AFTER} {[EDITADD] [EDITUPDATE]}
    OF tabla/lista_colmnas
    acción
    acción
      .
      .
      .
    acción
```

donde:

BEFORE	es una palabra reservada.
AFTER	es una palabra reservada.
EDITADD	es una palabra reservada refiriendose a la ción de editar durante una operción **Add**.
EDITUPDATE	es una operación reservada refiriendose a la acción de editar durante una operación **Update**.
tabla/lista_columnas	es una lista de tablas ó columnas; pueden ser hasta 16, dependiendo del sistema operativo.

Ejemplo de la forma **sample**:

```
after editadd editupdate of quantity
        let i19 = i18 * s15
        nextfield = o11
```

ADD

El uso de esta instrucción causa la ejecución de acciones después de la operación Add. La acción ocurre después de que el renglón se escriba en la tabla.

Sintaxis:

```
AFTER ADD OF lista_tablas
        acción
        acción
          .
          .
          .
        acción
```

donde:

AFTER ADD OF es una palabra reservada

lista_tablas es una lista de tablas separadas por un espacio en blanco.

Ejemplo de la forma de **sample**:

```
after add update query of items
        if (total of i19) <= 100 then
                let d1 = 7.50
        else
                let d1 = (total of i19) * 0.04
        let d2 = (total of i19) + d1
```

UPDATE

El uso de UPDATE ejecuta acciones después de la operación **Update**.

Sintaxis:

```
AFTER UPDATE OF lista_tablas
        acción
        acción
          .
          .
          .
        acción
```

donde:

AFTER UPDATE OF es una palabra reservada

lista_tablas es una lista de tablas separadas por un espacio en blanco.

Ejemplo de la forma **sample**:

```
after add update query of item
        if (total of i19) <= 100 then
                let d1 = 7.50
        else
                let d1 = (total of i19) * 0.04
        let d2 = (total of i19) + d1
```

QUERY

El uso de QUERY ejecuta acciones después de la operación **Query**.

Sintaxis:

AFTER QUERY OF lista_tablas
acción
acción
.
.
.
acción

donde:

AFTER QUERY OF es una palabra reservada

lista_tablas es una lista de tablas separadas por un espacio en blanco.

Ejemplo de la forma **sample**:

```
after add update query of item
        if (total of i19) <= 100 then
                let d1 = 7.50
        else
                let d1 = (total of i19) * 0.04
        let d2 = (total of i19) + d1
```

REMOVE

El uso de REMOVE ejecuta acciones antes ó después de la operación **Remove**.

Sintaxis:

[BEFORE | AFTER] REMOVE OF lista_tablas
acción
acción
.
.
.
acción

donde:

BEFORE	es una palabra reservada
AFTER	es una palabra reservada
REMOVE OF	es una palabra reservada
lista_tablas	es una lista de tablas separadas por un espacio en blanco.

Ejemplo:

```
instructions

before remove of customer
        comments reverse
        " Recordar enviar una noticia al departamento de ventas"
```

DISPLAY

El uso del DISPLAY ejecuta acciones después de cualquier operación que causa que los datos sean desplegados en pantalla.

Sintaxis:

AFTER DISPLAY OF lista_tablas
acción
acción
.
.
.
acción

donde:

AFTER DISPLAY OF es una palabra reservada

lista_tablas es una lista de tablas separadas por un espacio en blanco.

Ejemplo de la forma **sample**:

```
after display of orders
        let d1 = 0
        let d2 = 0
```

Sintaxis Acción

En esta sección se mostrarán las siguientes acciones:

Palabra Reservada	Acción
LET	Asigna valores a campos
NEXTFIELD	Mueve el cursor hacia un campo especificado o a la salida del menú PERFORM.
COMMENTS	Despliega un mensaje en la línea de Status
IF-THEN-ELSE	Ejecuta otras acciones dependiendo de una condición
ABORT	Sale del menú de PERFORM.

Estas acciones pueden estar incluidas en un BEFORE ó AFTER control block.

ABORT

El uso de ABORT en la sección INSTRUCTIONS en las acciones ADD, UPDATE ó REMOVE provoca que una salida del menú PERFORM.

Sintaxis:

ABORT

donde:

ABORT regresa al menú de PERFORM, cuando se ejecuta una de las acciones ADD, UPDATE ó REMOVE.

```
i18 = items.quantity, include = (1 to 50),
        comments = "Acceptable values are 1 through 50" ;
i19 = items.total_price;
o20 = po_num, required,
        comments = "If no P.O. Number enter name of caller" ;
a   = backlog, autonext;
o21 = ship_date, default = today, format = "mm/dd/yyyy";
o22 = paid_date, format = "mm/dd/yyyy";
o23 = ship_instruct;
d1 =  displayonly type money;
d2 =  displayonly type money;
```

instructions

```
customer master of orders;
orders master of items;
composites <items.stock_num, items.manu_code>
    *<stock.stock_num, stock.manu_code>

before editadd editupdate of orders
nextfield = o20

before editadd editupdate of items
nextfield = i13

after editadd editupdate of quantity
let i19 = i18 * s15
nextfield = o11

after add update query of items

  if (total of i19) <= 100 then
  let d1 = 7.50
  else
  let d1 = (total of i19) * .04

  let d2 = (total of i19) + d1
```

```
after display of orders
  let d1 = 0
  let d2 = 0

end
```