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

CENTRO DE INFORMACION Y DOCUMENTACION
"ING. BRUNO MASCANZONI"

EL CENTRO DE INFORMACION Y DOCUMENTACION "ING. BRUNO MASCANZONI" TIENE - POR OBJETIVO SATISFACER LAS NECESIDADES DE ACTUALIZACION AL PROPORCIONAR LA ADECUADA INFORMACION QUE PERMITA A LOS PROFESIONALES INGENIEROS PROFESORES Y ALUMNOS, ESTAR AL TANTO DEL ESTADO ACTUAL DEL CONOCIMIENTO SOBRE-TEMAS ESPECIFICOS ENFATIZANDO LAS INVESTIGACIONES DE VANGUARDIA DE LOS -- CAMPOS DE LA INGENIERIA TANTO NACIONALES COMO EXTRANJERAS.

POR LO QUE SE PONE A DISPOSICION DE LOS ASISTENTES DE LOS CURSOS DE LA - D.E.C.F.I.; ASI COMO AL PUBLICO EN GENERAL.

EN DICHO CENTRO USTED TENDRA LOS SIGUIENTES SERVICOS:

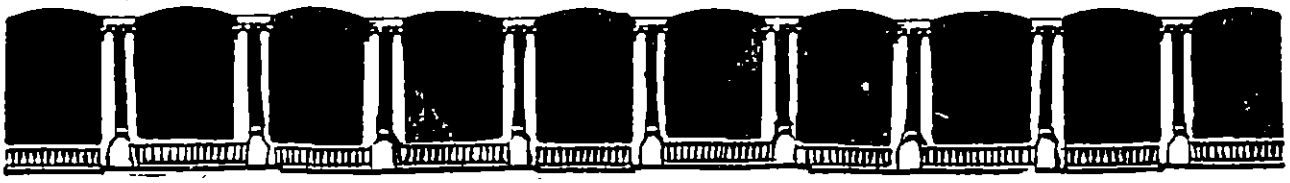
* PRESTAMO INTERNO
* PRESTAMO EXTERNO
* PRESTAMO INTERBIBLIOTECARIO
* SERVICIO DE FOTOCOPIADO
* CONSULTA TELEFONICA
* CONSULTA A LOS BANCOS DE DATOS: LIBRUNAM EN CD-ROM Y EN LINEA

LOS MATERIALES A SU DISPOSICION SON:

* LIBROS
* TESIS DE POSGRADO
* NOTICIAS TECNICAS
* PUBLICACIONES PERIODICAS
* PUBLICACIONES DE LA ACADEMIA MEXICANA DE INGENIERIA
* NOTAS DE LOS CURSOS QUE SE HAN IMPARTIDO DE 1971 A LA FECHA

EN LAS AREAS DE INGENIERIA INDUSTRIAL, CIVIL, ELECTRONICA, CIENCIAS DE LA-TIERRA, MECANICA Y ELECTRICA Y COMPUTACION.

EL C.I.D. SE ENCUENTRA UBICADO EN EL MEZZANINE DEL PALACIO DE MINERIA LADO ORIENTE. EN HORARIO DE SERVICIO DE 10:00 A 19:30 HORAS DE LUNES A VIERNES.



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

A LOS ASISTENTES A LOS CURSOS DE LA DIVISION DE EDUCACION CONTINUA

Las autoridades de la Facultad de Ingeniería, por conducto del Jefe de la División de Educación Continua, otorgan una constancia de asistencia a quienes cumplan con los requisitos establecidos para cada curso.

El control de asistencia se llevará a cabo a través de la persona que le entregó las notas. Las inasistencias serán computadas por las autoridades de la División, con el fin de entregarle constancia sólamente a los alumnos que tengan un mínimo del 80% de asistencias.

Pedimos a los asistentes recoger su constancia el día de la clausura. Estas se retendrán por el período de un año, pasado este tiempo la DECFI no se hará responsable de este documento.

Se recomienda a los asistentes participar activamente con sus ideas y experiencias, pues los cursos que ofrece la División están planeados para que los profesores expongan una tésis, pero sobre todo, para que coordinen las opiniones de todos los interesados, constituyendo verdaderos seminarios.

Es muy importante que todos los asistentes llenen y entreguen su hoja de inscripción al inicio del curso, información que servirá para integrar un directorio de asistentes, que se entregará oportunamente.

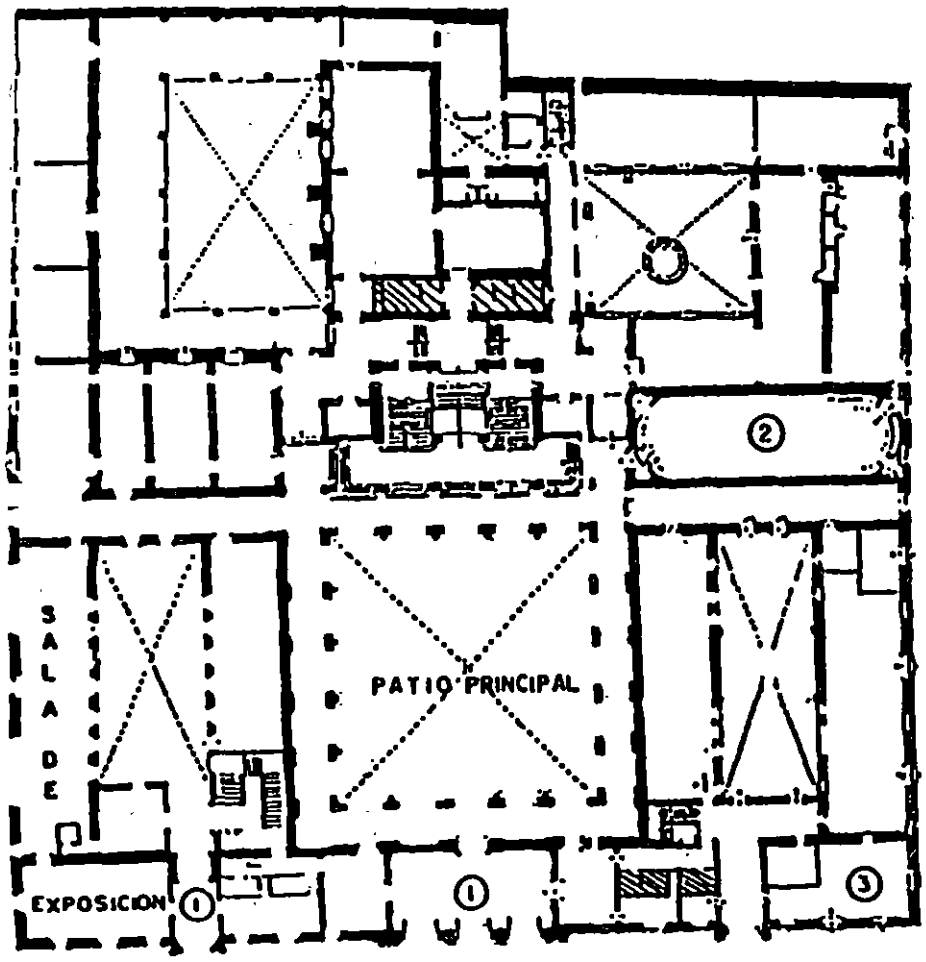
Con el objeto de mejorar los servicios que la División de Educación Continua ofrece, al final del curso deberán entregar la evaluación a través de un cuestionario diseñado para emitir juicios anónimos.

Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con ésto sean más fehacientes sus apreciaciones.

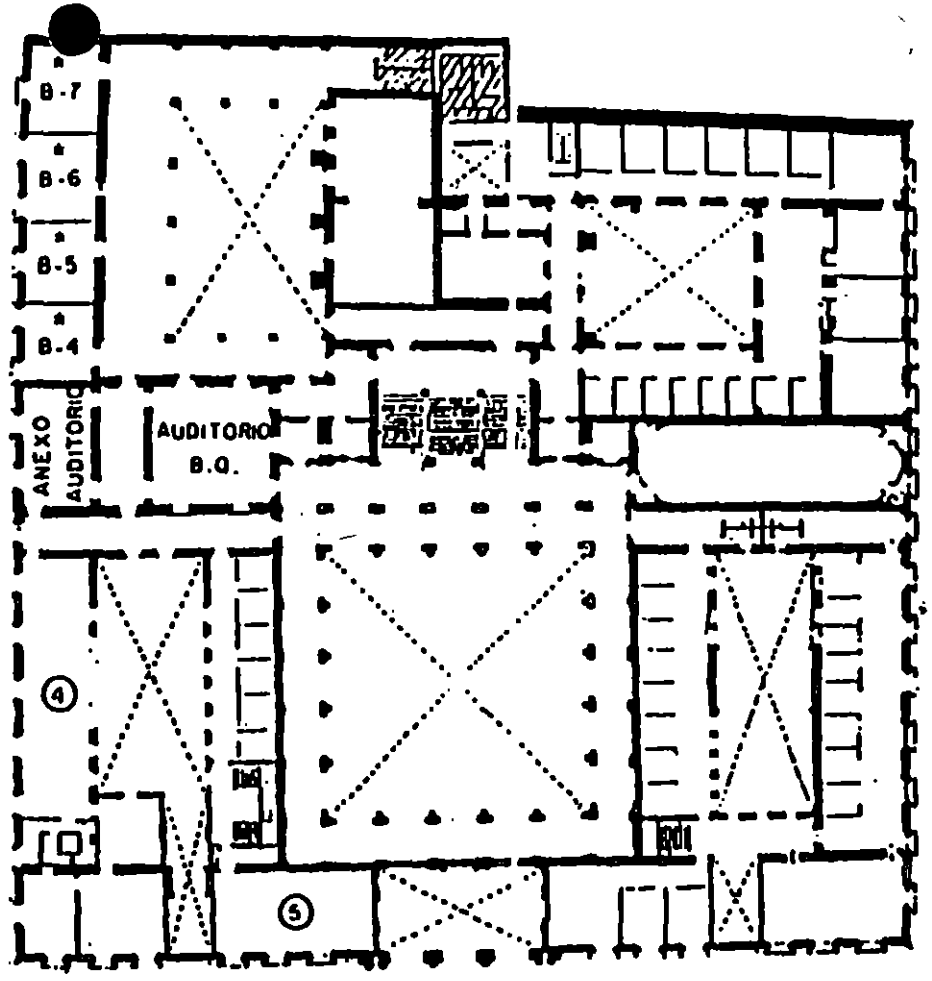
¡ G R A C I A S !

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

PALACIO DE MINERIA

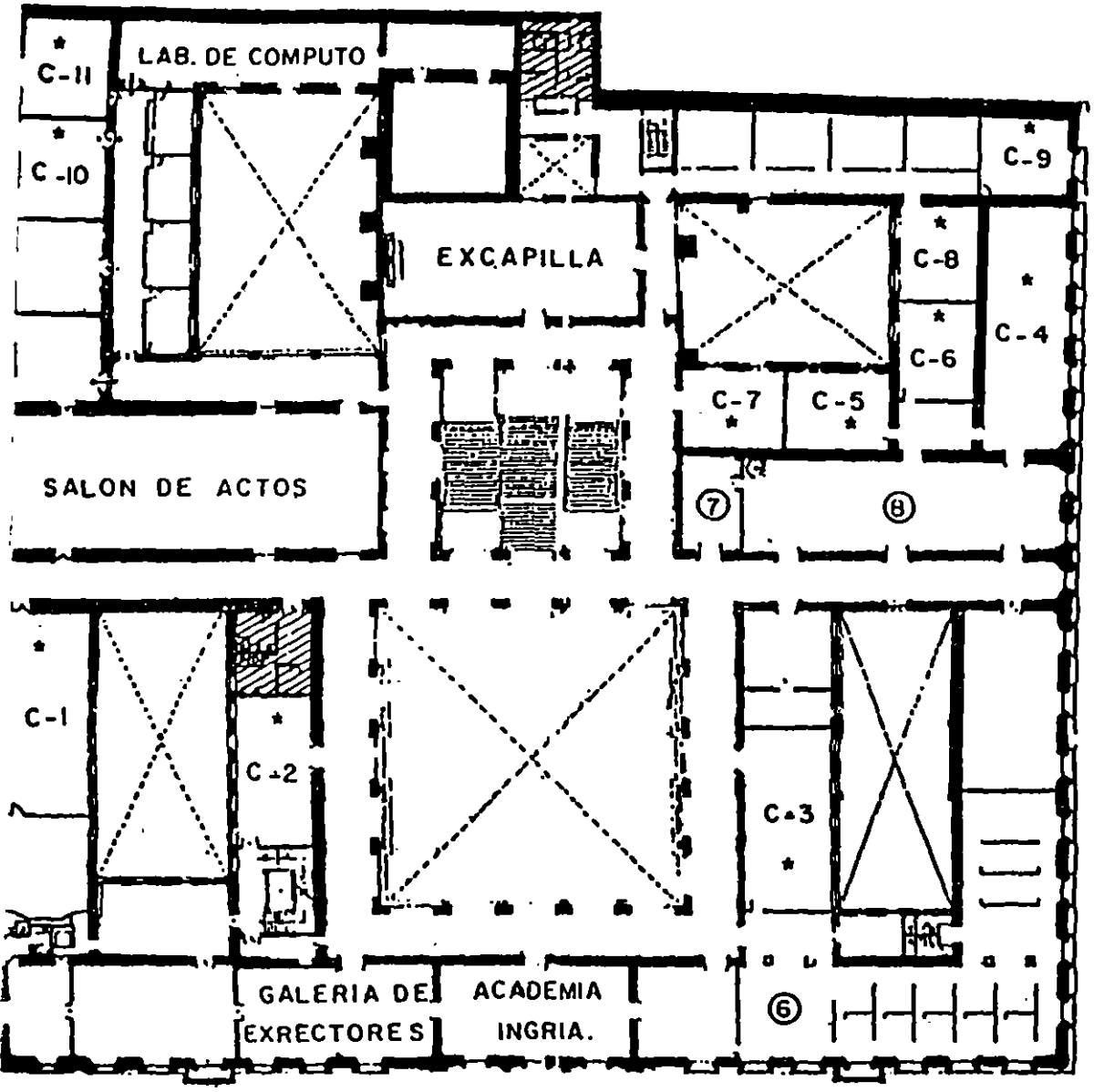


PLANTA BAJA



MEZZANINNE

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



1er. PISO

GUIA DE LOCALIZACION

1 – ACCESO

2 – BIBLIOTECA HISTORICA

3 – LIBRERIA U N A M

4 – CENTRO DE INFORMACION Y DOCUMENTACION "ING. BRUNO MASCANZONI"

5 – PROGRAMA DE APOYO A LA TITULACION

★ AULAS

6 – OFICINAS GENERALES

7 – ENTREGA DE MATERIAL Y CONTROL DE ASISTENCIA.

8 – SALA DE DESCANSO

▨ SANITARIOS

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

I N F O R M I X S Q L

MATERIAL DIDACTICO

04-14 SEPT.-95

Palacio de Mineria Calle de Tacuba 5 Primer piso Deleg. Cuauhtemoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

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:

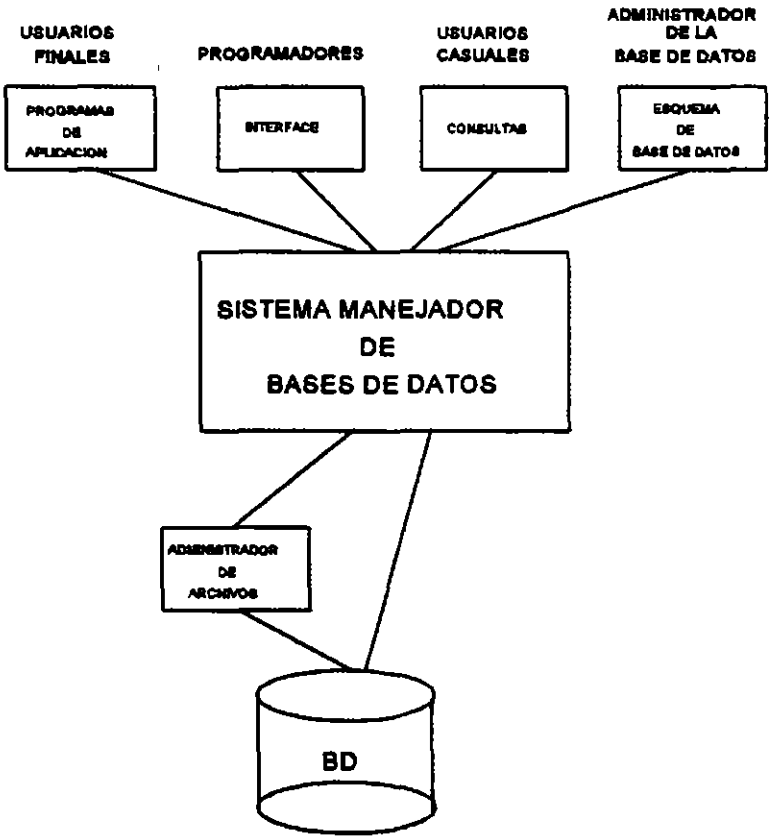
- Sistema de aplicación de la empresa.
- Colección de datos de operación.
- Datos interrelacionados.
- Facilidad de acceso, análisis y producción.
- Independencia programas-datos.

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 fisica
- 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

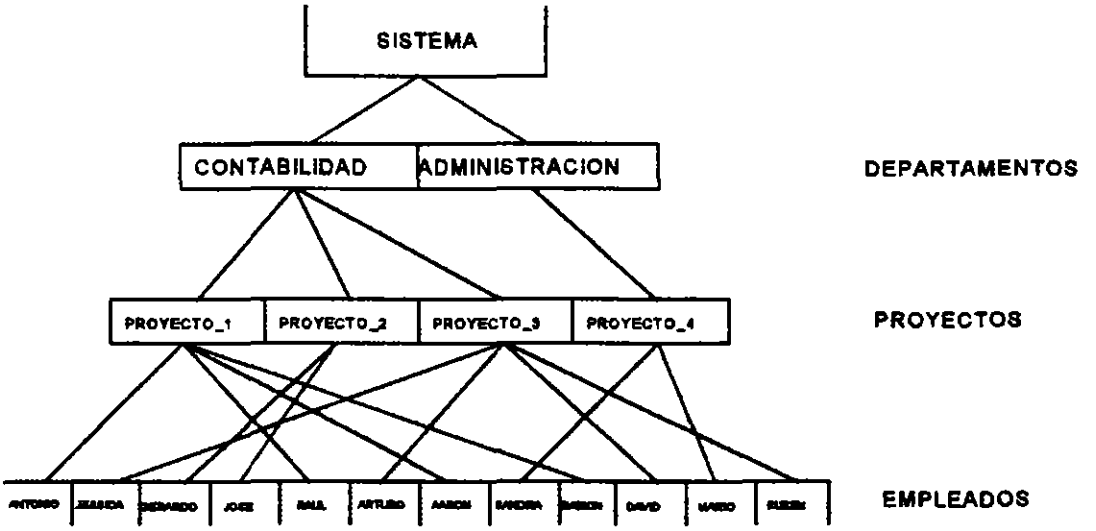
Estos modelos representan a la realidad tomando como base la forma en como los datos son almacenados en la computadora y como son mantenidas las asociasiones entre ellos. En estos modelos se crea una dependencia con la forma en como se implementa la Base de Datos.

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:

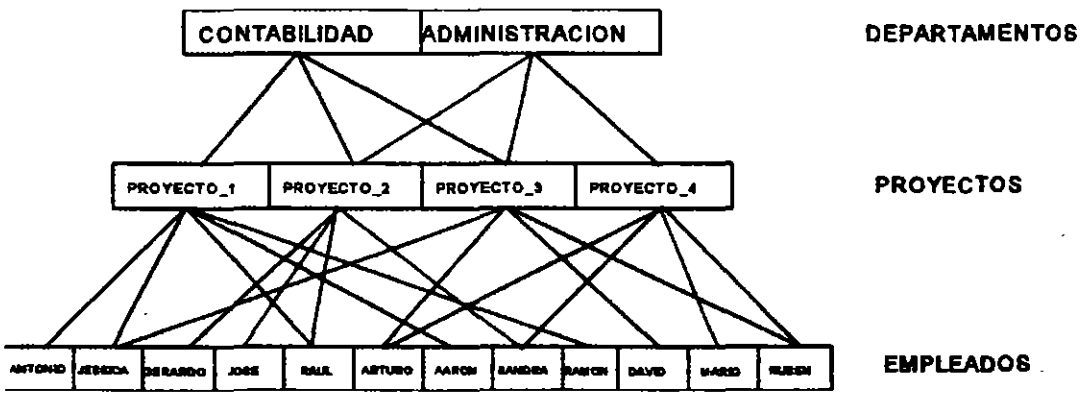
- No puede haber ciclos y sólo puede haber asociaciones 1:N y 1:1
- Los apuntadores o direcciones se deben almacenar junto con los datos
- Para recuperar información se debe recorrer el árbol
- No se puede obtener información no planeada antes de modelar
- Modificar la estructura de la Base de datos 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

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.

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.

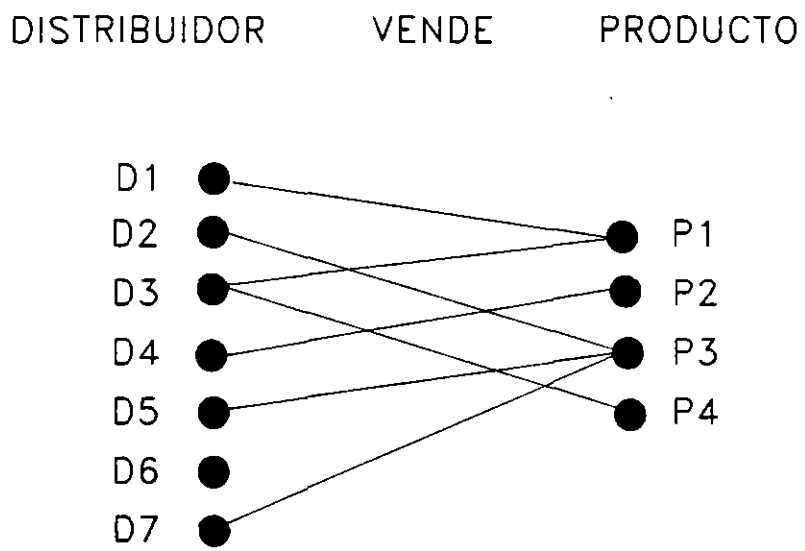
Las entidades representan tablas en el diseño relacional.

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.

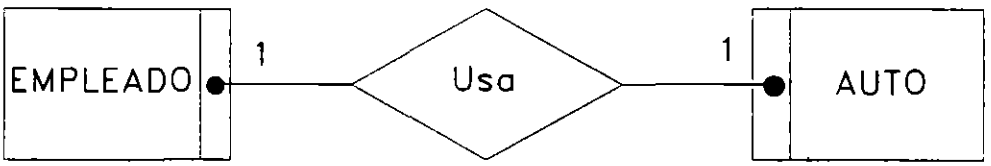
Representación de Entidades

Cada entidad independiente representa una tabla.

La llave primaria de tablas independientes no contiene llaves foráneas.

Representación de Relaciones 1:1

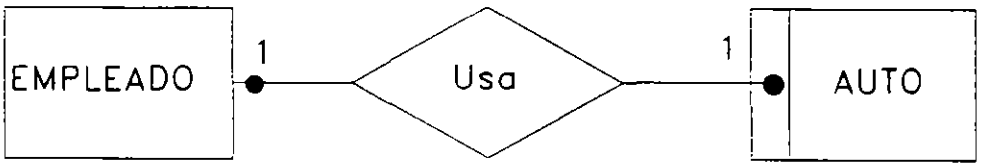
Obligatoriedad para ambas Entidades:



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

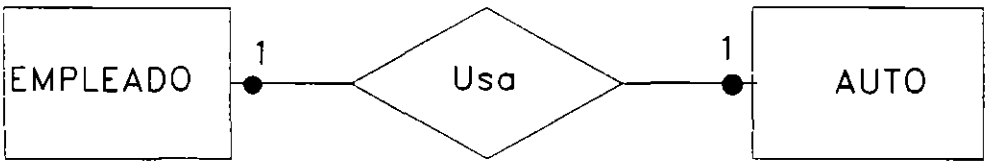
#empleado y #auto son la llave primaria de la Entidad.

Obligatoriedad sólo para una Entidad:



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

No obligatoriedad para las dos Entidades:

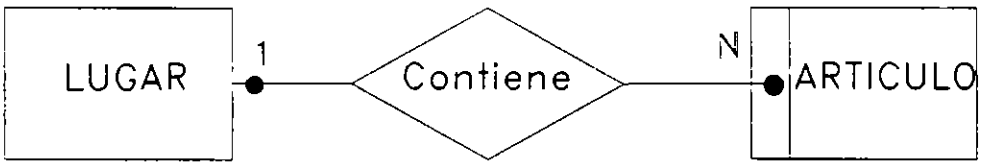


Se crea una tabla para la Relación:

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

Representación de Relaciones 1:N

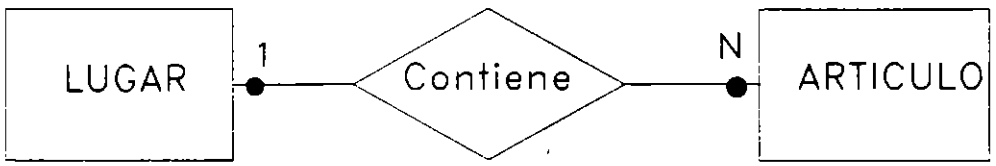
Obligatoriedad en la Entidad de grado N:



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

Notase 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:



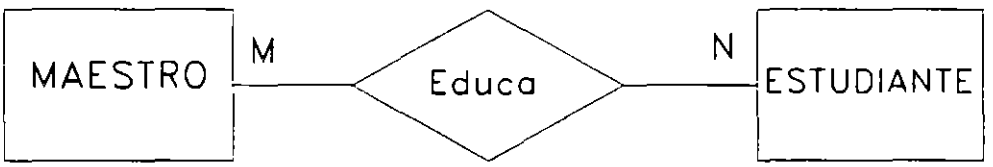
LUGAR(nombre, ...)
ARTICULO(#artículo, ...)
CONTIENE(#artículo, nombre, ...)

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 Relaciones M:N

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



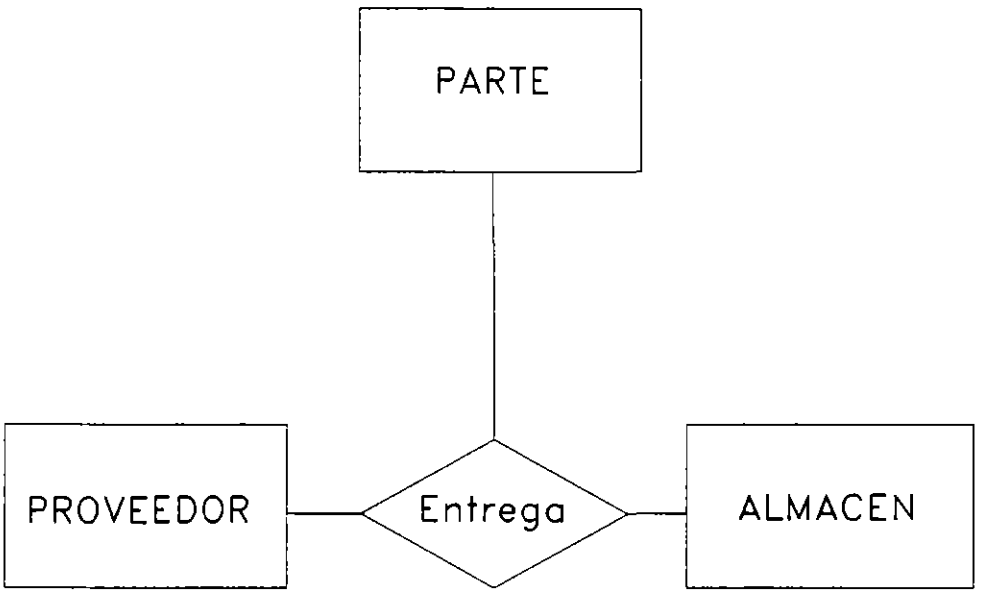
MAESTRO(nombre, ...)
ESTUDIANTE(#estudiante, ...)
EDUCA(nombre, #estudiante, ...)

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

Representación de Relaciones 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.

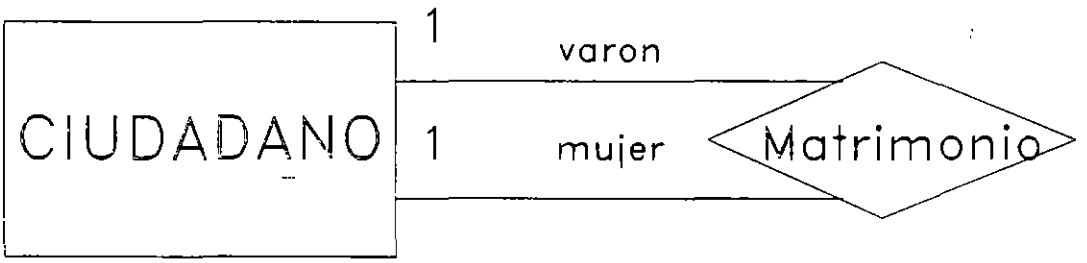


PARTE(#parte, ...)
ALMACÉN(#almacén, ...)
PROVEEDOR(#proveedor, ...)
ENTREGA(#proveedor,#parte,#almacén, ...)

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.



Algebra Relacional

El álgebra relacional es un lenguaje de consulta abstracto procedural. Existen cinco operaciones fundamentales:

- Proyección □
- Elección ñ
- Producto Cartesiano □
- Unión **U**
- Resta de conjuntos -
- Intersección de conjuntos ∩

Todas estas operaciones dan como resultado una nueva relación

Como se había mencionado anteriormente el modelo lógico relacional se basa en este concepto de álgebra relacional; es por esto que haremos una equivalencia de la representación gráfica del modelo relacional y el algebra relacional para un mejor entendimiento.

Representación gráfica del modelo relacional	**Algebra relacional**
Tabla	Relación
Columna	Atributo
Renglón	Tuplo
Número de columnas	Orden
Número de renglones	Cardinalidad

Para ejemplificar estas operaciones haremos uso de las tablas que vienen al final del capitulo.

Proyección

Es una operación anuario, ya que actúa sobre una sola relación. Se representa con la letra griega □. Se utiliza para proyectar los atributos (columnas), su sintaxis es la siguiente:

$$\square_{\text{atributo 1, atributo 2, ..., atributo n}} \text{ (Relación ó Tabla)}$$

Ejemplo: Listar a los alumnos y su número de cuenta

$$\square_{\text{nom\_alum, no\_cta}} (\text{ALUMNO})$$

no_cta	nom_alum
105	Sandra
107	José
101	Luis
103	Lourdes
106	Juan
102	Santiago
108	Verónica

Ejemplo: Cuales son los profesores que se tienen

$$\square_{nom\_prof}(PROFESOR)$$

Nom_prof
Antonio
Ramón
Edwin
Jéssica

Elección

Al igual que la proyección la elección es una operación anuario. Se representa con la letra griega ή,y su función es elegir aquellos tuplos (renglones) que cumplan con la condición especificada; su sintaxis es la siguiente:

$$\acute{\eta}_{\text{condición}}(\text{Relación ó Tabla})$$

donde:

condición: es una expresión la cual puede tener los operadores de relación (>, <, >=, <=, =, !=) y operadores lógicos (AND, OR, NOT).

Ejemplo: Qué alumnos cursan la carrera cuya clave de carrera es 027

$$\acute{\eta}_{\text{cve\_carr=027}}(\text{ALUMNOS})$$

Nom_alum	Cve_carr
Sandra	027
Lourdes	027
Véronica	027

Ejemplo: Cual es la clave de la carrera de Computación

$$\pi_{\text{carrera= "Computación"}} (\text{CARRERA})$$

Cve_carr
027

Producto Cruz

Es una operación binaria ya que actúa sobres dos relaciones (tablas), se representa por una cruz □. Esta operación me permite combinar ó relacionar informacion de dos o mas relaciones (tablas). Su sintaxis es la siguiente:

(Relación ó Tabla) □ (Relación ó Tabla)

Esta operación nos da como resultado una relación que tiene **nXm** renglones y todas las columnas de cada relación (tabla).

donde:

n= número de renglones de la relación 1 (tabla 1)
m= número de renglones de la relación 2 (tabla 2)

NOTA: Se recomienda que para hacer el producto cruz de dos relaciones se tenga una columna (atributo) en común.

Ejemplo:

(ALUMNO) □ (CARRERA)

ALUMNO. no_cta	ALUMNO. nom_alum	ALUMNO. cve_carr	CARRERA. cve_carr	CARRERA. carrera
105	Sandra	027	027	Industrial
105	Sandra	027	032	Computación
105	Sandra	027	029	Electrónica
107	José	032	027	Industrial
107	José	032	032	Computación
107	José	032	029	Electrónica
101	Luis	032	027	Industrial
101	Luis	032	032	Computación
101	Luis	032	029	Electrónica
103	Lourdes	027	027	Industrial
103	Lourdes	027	032	Computación
103	Lourdes	027	029	Electrónica
106	Juan	029	027	Industrial
106	Juan	029	032	Computación
106	Juan	029	029	Electrónica
102	Santiago	029	027	Industrial
102	Santiago	029	032	Computación
102	Santiago	029	029	Electrónica
108	Véronica	027	027	Industrial
108	Véronica	027	032	Computación
108	Véronica	027	029	Electrónica

Las siguientes operaciones: Union, Resta y la intersección para que se puedan realizar deben cumplir con dos condiciones:

1. Las relaciones involucradas deben tener el mismo múmero de atributos (columnas).
2. Los dominios de los atributos (columnas) i-ésimo de las relaciones (tablas) involucradas deben ser los mismos.

Unión

Es una operación binaria, con la cual podemos obtener todos los tuplos (renglones) de las relaciones (tablas) involucradas. Su sintaxies es la siguiente:

Relación 1 (tabla 1) $\cup$ Relación 2 (tabla 2)

Resta

Es una operación binaria, con la cual obtenemos todos los tuplos (renglones) que se encuentran en la primera relación (tabla 1), pero que no se encuentran en la segunda relacion (tabla 2). Su sintaxies es la siguiente:

Relación 1 (tabla 1) - Relación 2 (tabla 2)

Intersección

Esta operación es adicional ya que se puede obtener de las operaciones anteriores. Esta operación nos dá como resultado aquellos tuplos (renglones) que se encuentran en ambas relaciones (tablas).

Relación 1 (tabla 1) $\cap$ Relación 2 (tabla 2) = Relación 1 - (Relación 1 - Relación 2)

NOTA: Esta operación no es conmutativa. $A - B \neq B - A$

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

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

I N F O R M I X SQL

MATERIAL DIDACTICO

SEPTIEMBRE 1995

Palacio de Mineria Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Telefonos 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

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. alfaN prometeo> isql ↵

Informix-SQL Menú Principal

El Menú Principal muestra las siguientes opciones:

Form

Report

Query-Language

User-menu

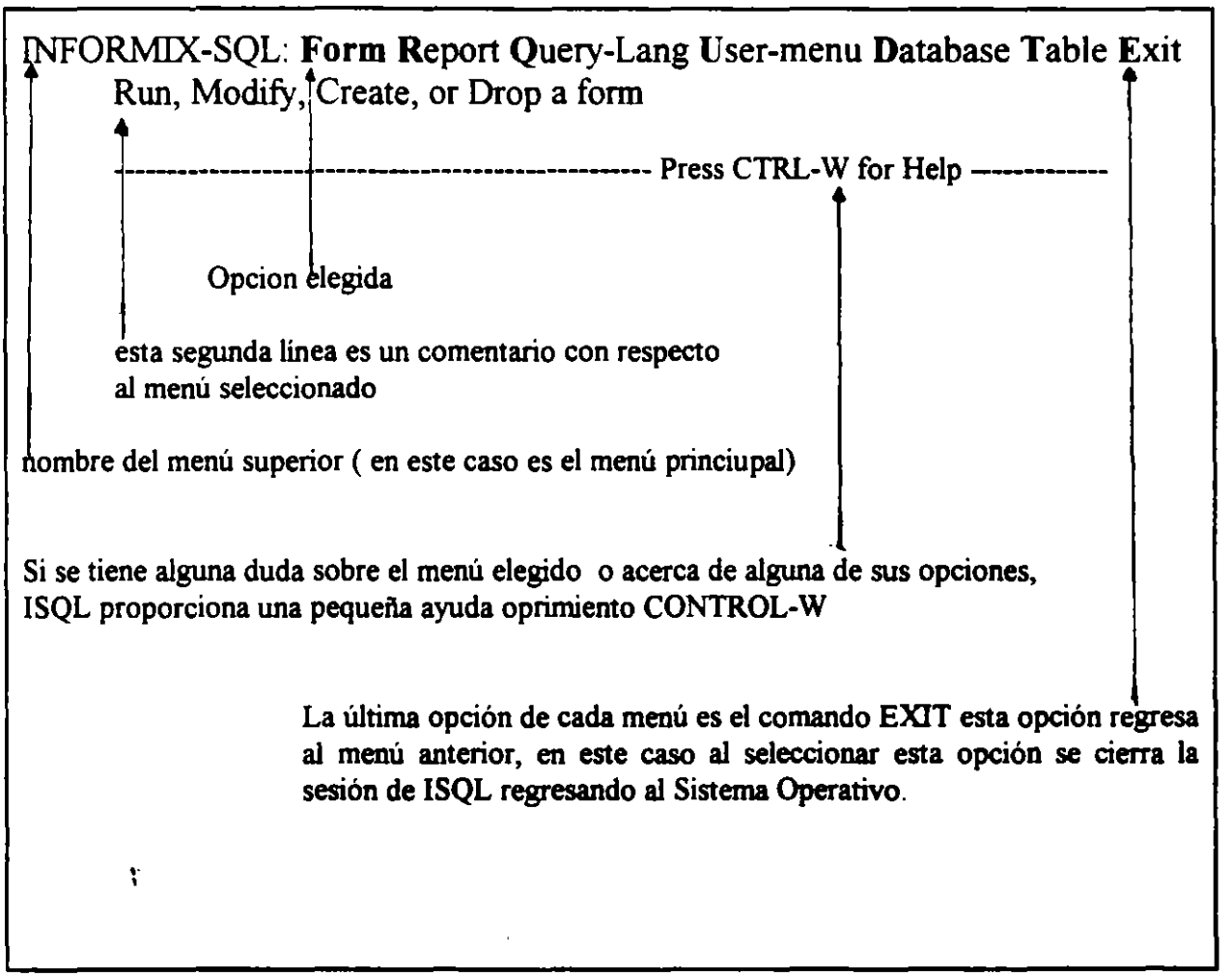
Database

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:



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 linea, 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 alfaN, si ya existe seleccionarla.

2. Crear 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

SQL
Structure Query Language

EL LENGUAJE SQL

SQL (*Structure Query Languaje*) es un lenguaje estandar que podemos definir en tres 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 númericas 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]

Apéndice A

Edición de archivos

El editor vi

El editor vi o editor Visual es un editor orientado a pantalla, es decir, aprovecha las ventajas de las terminales para mostrar los cambios que se realizan en los archivos al mismo tiempo que estos se efectúan. Además permite visualizar el archivo en pantalla desplazandolo a través de esta.

El editor vi sigue un esquema como el mostrado en la figura 4.1. La invocación para el editor vi es la siguiente:

% vi archivo

si se omite el nombre de archivo se entra a editar un archivo nuevo.

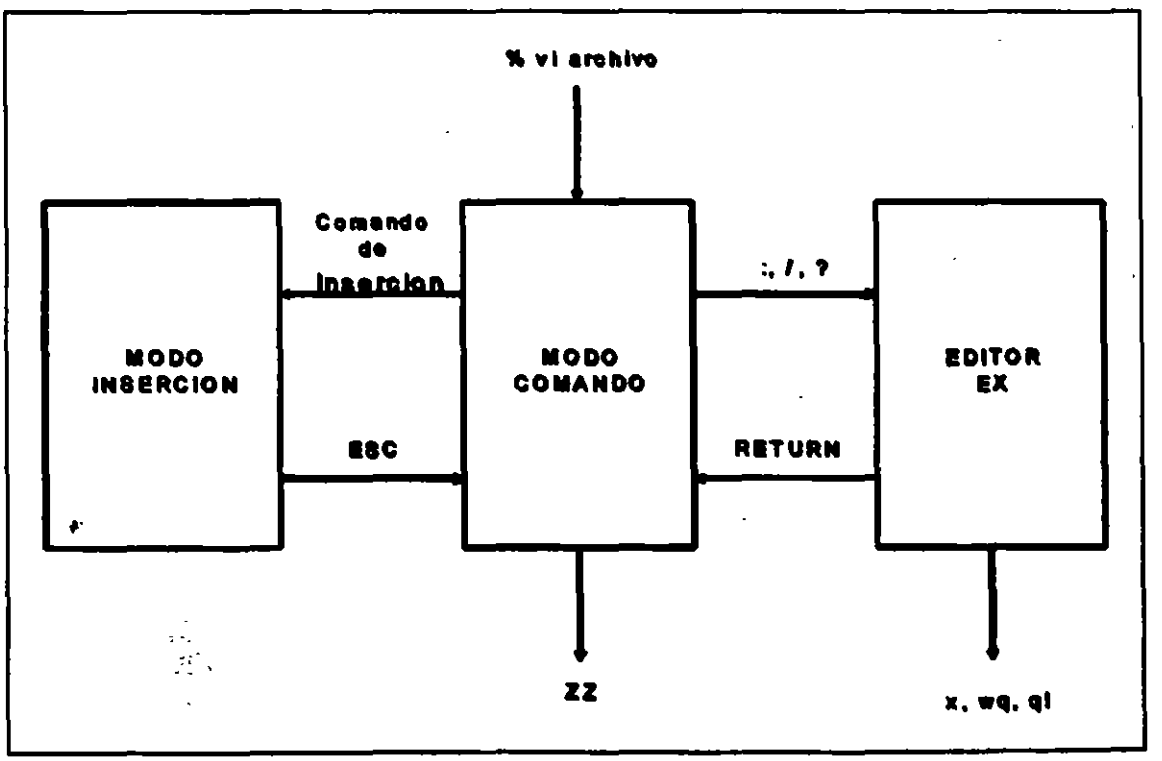


Fig. 4.1. Esquema del editor vi.

Cuando se comienza a editar con vi se entra en el modo comando. Este modo permite el movimiento del cursor a través del archivo que se va desplazando por la pantalla. La tabla 4.1 muestra algunos de los comandos para movimiento del cursor. La mayor parte de ellos son movimientos sencillos, no es necesario dar ejemplos de ellos, solamente se ejemplificarán aquellos para los cuales se crea necesario.

COMANDO	DESCRIPCION
k, -	Mueve el cursor una línea arriba tratando de conservar la posición dentro de la columna en la que se encuentra.
j, RETURN. +	Mueve el curso una línea abajo.
h, Ctrl-h	Mueve el cursor hacia la izquierda.
l	Mueve el cursor hacia la derecha.
0	Mueve el cursor al principio de la línea.
$	Mueve el cursor al final de la línea.
w, W	Mueve el cursor a la siguiente palabra.
b, B	Mueve el cursor a la palabra anterior.
e, E	Mueve el cursor al final de la palabra en la que se encuentra el cursor o de la siguiente.
)	Mueve el cursor a la próxima frase.
(	Mueve el cursor a la frase anterior.
}	Mueve el cursor al próximo párrafo.
{	Mueve el cursor al párrafo anterior.
nG	Posiciona el cursor en la línea n del texto.
G	Mueve el cursor al final del texto.
Ctrl-f	Posiciona el cursor en la próxima página de texto. Se entiende por página de texto las líneas que aparecen en pantalla.
Ctrl-b	Mueve el cursor a la página anterior.

Tabla 4.1. Comandos del editor vi para movimiento del cursor.

Para comenzar a editar el archivo se debe pasar a modo de inserción a través de un comando, una vez que se ejecuta este comando, todo el texto que se teclea se introduce en el texto. La mayoría de los sistemas UNIX no permiten moverse a través de la pantalla una vez que se entra en este modo. Para volver a modo comando se

debe utilizar la tecla <ESC>. La tabla 4.2 muestra los comando más comunes de inserción y reemplazo.

COMANDO	DESCRIPCION
a	Comienza la inserción de texto a la derecha de donde se encuentra el cursor.
i	Comienza la inserción en la posición del cursor.
I	Inserción al comienzo de la línea.
A	Inserción al final de la línea.
O	Abre una línea antes de aquella en la que se encuentra el cursor y activa modo de inserción.
o	Abre una línea después de aquella en la que se encuentra el cursor y activa el modo de inserción.
R	Activa el modo de inserción a partir de la posición del cursor sobrescribiendo el texto actual.
nr	Reemplaza n caracteres a partir de la posición del cursor.
ncw	Reemplazo n palabras, por el texto que se introduce, hasta abandonar el modo de inserción.
ns	Reemplaza n caracteres.
S	Reemplaza la línea sobre la que se encuentra el cursor.

Tabla 4.2. Comandos de inserción y reemplazo.

En modo comando se pueden utilizar, también comandos de borrado, algunos de ellos se muestran en la tabla 4.3.

COMANDO	DESCRIPCION
nx	Borra n caracteres.
ndw	Borra n palabras.
ndd	Borra n líneas.
nX	Borra n caracteres a la izquierda del cursor.
D	Borra toda la línea y la deja en blanco.

Tabla 4.3. Comandos de borrado.

Cuando se borran caracteres, líneas o palabras con alguno de los comando de borrado vistos anteriormente, vi guarda en una memoria temporal el último conjunto de caracteres borrados, los cuales constituyen un bloque y se pueden colocar sobre otra parte del texto por medio de algún comando de manejo de bloques. Los comandos de manejo de bloque se muestran en la tabla 4.4.

COMANDO	DESCRIPCION
nyy	Guarda en la memoria temporal las n líneas que se encuentran a partir de la posición del cursor.
p	Escribe el texto que se encuentra en la memoria temporal en la posición siguiente del cursor si se trata de caracteres o palabras o en la línea abajo del cursor en caso de líneas.
P	Escribe el texto que se encuentra en la memoria temporal en la posición del cursor si se trata de caracteres o palabras o en la línea arriba del cursor en caso de líneas.

Tabla 4.4. Comandos de manejo de bloques.

El editor incluye un conjunto de comandos para manipulación de archivos y búsquedas, los cuales son mostrados en las tablas 4.5 y 4.6 respectivamente. Cuando se ejecutan estos comandos la posición del cursor se cambia a la última línea.

COMANDO	DESCRIPCION
:w	Archiva el texto actual.
:w nombre	Graba el texto en el archivo que se especifica.
:wq	Archivar el texto actual y salir de la sesión de vi.
:q	Salir de sesión de vi (antes se debieron archivar los cambios hechos al texto).
:q!	Salir de vi sin archivar los cambios.
:e nombre	Edita el archivo que se especifica.
:e +nombre	Edita el archivo que se especifica y coloca el cursor al final del archivo.

COMANDO	DESCRIPCION
:r nombre	Lee el archivo especificado y lo coloca a partir de la posición del cursor.
:!comando	Ejecuta un comando del shell y regresa a vi.

Tabla 4.5. Comandos de manipulación de archivos.

COMANDO	DESCRIPCION
/patrón	Busca el patrón a través del texto a partir de la posición del cursor hacia abajo y en forma circular, es decir, si no lo encuentra y llega al final de texto comienza en la primera línea hasta llegar a la posición del cursor.
?patrón	Igual que el comando anterior, solamente que la búsqueda es en sentido inverso.
n	Búsqueda de la siguiente ocurrencia del patrón.
N	Búsqueda de la siguiente ocurrencia del patrón en sentido inverso.

Tabla 4.6. Comandos de búsqueda.

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

I N F O R M I X, S Q L.

MATERIAL DIDACTICO

COMPLEMENTO

SEPTIEMBRE 1995.

Palacio de Mineria Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

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 : Y
?	sustituye un caracter	CHAR	? X,caracteres
*	sustituyecero o más caracteres	CHAR	* X, caracteres

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 no 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* e 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.

La 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 necesariamente 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 deacuerdo 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 *control 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. Aqui 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 y/ó decisiones lógicas 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 determina cuando ellos toman efecto

- 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, atributo;

- 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 usado 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 deben 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 utilizarse para prevenir que el valor que se esta introduciendo a la tabla activa no exista 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 previene la entrada de datos durante la creación de un nuevo renglón, utilizando 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:

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 cción de editar durante una operción **Add**.
EDITUPDATE	es una operación reser da refiriendose a la acción de editar durante una oper ión **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.

LET

El uso de la acción LET asigna un valor a un field tag para desplegarlo en la forma.

Sintaxis:

LET field tag = expresión

donde:

LET — es una palabra reservada

field tag — es un field tag de solo despliegue (display only field) de una columna de la tabla/lista_columnas.

expresión — expreseión que puede ser:

- Un field tag
- Una constante
- Una función agregada: COUNT, TOTAL, AVERAGE, MAX, MIN; estas funciones de ser seguidas de la palabra OF field tag, donde el field tag es el nombre de la columna y no es del tipo display only field.
- TODAY que regresa la fecha de hoy
- Una combinación de los operadores aritméticos: +, -, * y /.

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
```

NEXTFIELD

Cuando se utilizan las opciones EDITADD ó EDITUPDATE, la acción NEXTFIELD mueve directamente el cursor. Por default NEXTFIELD sigue la secuencia de los campos como se definieron en la sección ATTRIBUTES.

Sintaxis:

NEXTFIELD = field tag | EXITNOW

donde:

NEXTFIELD palabra reservada

field tag es el field tag correspondiente a la columna de la tabla activa.

EXITNOW es una palabra reservada para que NEXTFIELD termine la edición actual.

Ejemplo de la forma **sample**:

```
before editadd editupdate of orders
        nextfield = o20
```

COMMENTS

Esta acción despliega un mensaje en la linea de Status de la pantalla. Esta contrasta con el atributo COMMENTS de la sección ATTRIBUTE, la cual despliega un mensaje en la línea de Comentarios.

Sintaxis:

COMMENTS [BELL] [REBERSE] "mensaje"

donde:

COMMENTS	es una palabra reservada que escribe un mensaje en la linea de Status.
BELL	er una palabra reservada que manda una campana cuando se despliega el mensaje.
REVERSE	es una palabra reservada que pone el mensaje en video inverso.
mensaje	es el mensaje que se va a enviar en la línea se Status.

IF-THEN-ELSE

Esta acción realizará una serie de acciones cuando la condición sea evaluada.

Sintaxis:

IF expresión_booleana THEN acción [ELSE acción]

d: de:

IF	es una palabra reservada
expresión_booleana	es uan expresión booleana que tomará el valor de falso ó veradadero.
THEN	es una palabra reservada
acción	es la acción o lsa acciones que se realizarán si la expresión booleana toma el valor de falso.

Modificación de un pantalla de la forma

La modificación de una forma se puede realizar desde ISQL o bien desde el sistema operativo. Desde ISQL se elige el menu de **Form**, después la opción de **Modify**, en donde estará esperando se seleccione la forma que se desea modificar.

```
database ejemplo
        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

        tables
        cliente

        attributes
        c1= cliente.no_clien
        c2 = cliente.nombre
        c3 = cliente.apellidos
        c4 = cliente.direccion
        c5 = cliente.colonia
        c6 = cliente.delegacion
        c7 = cliente.cp
        c8 = cliente.telefono
        end
```

A continuación se muestra la forma **SAMPLE**:

```
database stores

screen
{

==========================================================================
==========================================================================
CUSTOMER INFORMATION:

Customer Number: [c1         ]

      Company: [c4                        ]
   First Name: [c2          ]                  Last Name: [c3                ]

     Address: [c5                  ]
              [c6                  ]

         City: [c7                ]     State: [ c8 ]        Zip: [c9   ]
    Telephone: [c10                ]

==========================================================================
==========================================================================
}

screen

{
==========================================================================
CUSTOMER NUMBER: [c1       ]              COMPANY: [c4                        ]

ORDER INFORMATION:
         Order Number: [o11       ]          Order Date: [o12             ]

         Stock Number: [i13       ]          Manufacturer: [ i16 ]
           Description: [s14          ]                    [m17          ]
                  Unit: [s16          ]
                                        Quantity: [i18       ]
                                        Unitprice: [s15          ]
                                        Total Price: [i19          ]
```

```
SHIPPING INFORMATION:
 Customer P.O.: [o20      ]                    Ship Charge: [d1          ]

        Backlog: [ a ]                    Total Order Amount: [d2       ]
      Ship Date: [o21          ]
      Date Paid: [o22          ]
   Instructions: [o23                                      ]
}
end

tables
customer items stock
orders manufact

attributes
c1 = *customer.customer_num
  = orders.customer_num;
c2 = fname,
     comments = "Please enter initial if available";
c3 = lname;
c4 = company, reverse;
c5 = address1;
c6 = address2;
c7 = city;
c8 = state, upshift, autonext,
    include = ("CA","OR","NV","WA"),
    default = "CA" ;
c9 =  zipcode, autonext;
c10 = phone, picture = "###-###-####x####";
o11 = *orders.order_num = items.order_num;
o12 = order_date, default = today, format = "mm/dd/yyyy";
i13 = items.stock_num;
   = *stock.stock_num, noentry, noupdate, queryclear;
i16 = items.manu_code, lookup m17 = manufact.manu_name
          joining *manufact.manu_code, upshift, autonext;
   = *stock.manu_code, noentry, noupdate,
        upshift, autonext, queryclear;
s14 = stock.description, noentry, noupdate;
s15 = stock.unit_price, noentry, noupdate;
s16 = stock.unit_descr, noentry, noupdate;
```

```
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    oe money;
d2 =  displayonly    oe 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
```

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

I N F O R M I X SQL

M A T E R I A L D I D A C T I C O

SEPTIEMBRE 1995

Palacio de Mineria Calle de Tacuba 5 Primer piso Deleg. Cuauhtemoc 06000 México, D.F. APDO. Postal M-2285
Telefonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

Chapter 12.

ACE Report Format

Chapter Objectives

At the end of this lesson, you will be able to:

- create a default report using the Informix-SQL menu environment
- use the five options of the Informix-SQL report menu to create and run a report
- identify the six sections of an ACE report and determine which three are required for all reports
- change the appropriate four sections of an ACE report for
  - defining variables
  - altering margins
  - accepting user input
  - redirecting output

The ACE Menu Structure

You can create, compile, and run a report specification through the INFORMIX-SQL menus.

From the Main Menu, select r for Report.

You will be presented with the following screen:

```
REPORT: Run Modify Generate New Compile Drop Exit
Run a Report.
---------- -------------------------Press CTRL-W for Help------
```

Run

Modify

Generate

New

Compile

Drop

CHAPTER EXERCISE

Generating a Default Report

Press **g** to Generate a default report.

You are prompted for input along the way.
Use the table listed below to determine appropriate responses.

Menu prompt	your response
SELECT DATABASE >>	stores# (if no current database)
GENERATE REPORT >>	cust_rpt
CHOOSE TABLE >>	customer

Press **r** to Run the report.

Compiling a report produces two files.

filename.ace	This is the ASCII (readable) report specification file.
filename.arc	This is the compiled version of the report specification file. This is the one that is runable.

Output for the Default Report Specification for the Customer Table

```
cnum        101
fname       Donald
lname       Quinn
company     Quinn's  Sports
address1    587 Alvarado
address2
city        Redwood City
state       OR
zip         84063
phone       915-544-8729

cnum        102
fname       Charles
lname       Ream
company     Athletic Supplies
address1    41 Jordan Avenue
address2
city        Reno
state       NV
zip         24086
phone       415-356-9876

      . . .

cnum        118
fname       Bill
lname       Blakely
company     Blue Ribbon Sports
address1    5427 College
address2
city        Oakland
state       CA
zip         94609
phone       415-655-0011
```

A Default Report Specification

From the Report Menu type **m** for Modify.

Specify the default form **cust_rpt**.

The following report specification will be displayed.

```
database stores end

select
     cnum,
     fname,
     lname,
     company,
     address1,
     address2,
     city,
     state,
     zip,
     phone
from customer end

format every row end
```

Report Specification Statements

ACE report specifications have three required and three optional sections.

*Database	Identifies the database on which the report is based.
Define	(optional) Defines any local variables used in the report.
Input	(optional) Generates prompts for user-entered information.
Output	(optional) Specifies (1) the page margins and (2) the place where the output is to be sent.
*Select	Identifies the data that will be used in the report. Multiple SELECT statements may be used in this section.
***Format**	**Specifies how the output should be displayed.**

* *This is a required section.*

The DATABASE Section

Every ACE report must begin with the DATABASE keyword.

This MUST be the first line of the report.

Syntax:

DATABASE database-name END

DATABASE is a required section.

The keyword END is required.

The DEFINE Section

The DEFINE section is used to declare variables used in the report and parameters the report can accept from the command line.

```
Syntax:

define
     param[integer]     variable-name     data-type
     variable           variable-name     data-type
end
```

DEFINE is an optional section. If used it must come after the database section.

Using variables

Variables needed for the SELECT or FORMAT section of an ACE report are defined here.

Variable names

must begin with a letter
can contain letters, numbers, and underscores
have a maximum length of 18 characters
can be any database data type except serial

Using parameters

"Parameters" refers to user-entered parameters which are listed when calling an ACE report from the operating system command line.

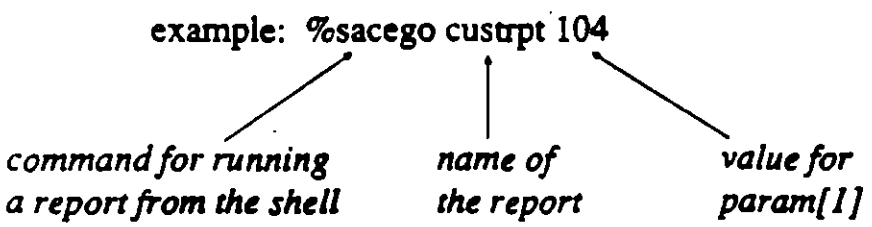
If a report has parameter variables defined, that report cannot be run from within the INFORMIX-SQL menu environment. It must be run from the command line, passing the report one value for each parameter defined.

Use the same data types for parameters that you use for variables in ACE.

Example:

```
define
     param[1]  cust_num        integer
     variable  begin_date      date
     variable  end_date        date
     variable  mfc             char(3)
end
```

Note
Parameters must be passed to an ACE report at run time. In the example above, this report would be run by using the sacego command.



CHAPTER EXERCISE

Sam has decided that he wants a report on his customers. He wants to specify a range of customer numbers, so that he can give his first ten customers special deals and implement other such promotional ideas.

Right now, you don't know how to use this information in your select statement, so just set up the variables and we'll complete Sam's request in the next exercise.

Modify our default report **cust_rpt** to include the definition of two integer variables

start_num end_num

Start by pressing **m** for modify from the Report menu.

Highlight the report **cust_rpt** and press RETURN.

Use **vi** to insert a define section in your report.

Don't forget DEFINE must come AFTER the DATABASE section.

The INPUT section

The input section allows you to produce an interactive ACE report by prompting for and accepting input while ACE is running a report.

The input section consists of the keywords INPUT and END with one or more PROMPT FOR statements in between.

```
Syntax:

input
     prompt for variable-name using "message string"
end
```

The INPUT section is optional.

If used it MUST follow the DEFINE section.

Variables being prompted for must be in the define section.

```
Example:

input
     prompt for begin_date using
           "Please enter the starting date: mm/dd/yy "
     prompt for end_date using
          "Please enter the ending date: mm/dd/yy "
end
```

When this ACE report is run, the user will be prompted for the starting date. Once the data is entered, press RETURN. The user will then be prompted for the ending date.

NOTE: You CANNOT prompt for parameters.

The OUTPUT Section

The output section controls the width of the margins and the length of the page. The output section also allows you to direct the report to a file or system printer. On UNIX systems, you can direct output to a pipe.

```
Syntax: including default values

OUTPUT                                          Defaults
     LEFT MARGIN              integer           6
     RIGHT MARGIN             integer           132
     TOP MARGIN               integer           3
     BOTTOM MARGIN            integer           3
     PAGE LENGTH              integer           66
     REPORT TO                "filename"        screen
     REPORT TO PIPE           "program"
     REPORT TO PRINTER
END
```

The OUTPUT section is optional.

It MUST come after the INPUT section.

If no OUTPUT section is included, OUTPUT retains the default values.

Examples

The following example directs the output to the file "labels.out":

```
output
     page  length  9
report  to  "labels.out"
      left  margin  0
end
```

CHAPTER EXERCISE

Modify your report **cust_rpt** to include the following:

An INPUT section:
PROMPT the user for a starting customer number and an ending customer number.

An OUTPUT section:
Change the page length to 22
Send the report to a pipe "more"
Change the left margin to 0

Run your report.

The SELECT Section

A SELECT statement in an ACE report can be any select statement we have used so far. The only difference is the keyword END.

```
Syntax:

SELECT select-list
     FROM table-name
     [WHERE condition]
     [GROUP BY column-list]
     [HAVING condition]
     [ORDER BY column-name]
     [INTO TEMP table-name]
END
```

The SELECT section is required in an ACE report. It must come after the DATABASE, DEFINE, INPUT, and OUTPUT sections.

You may use variables or parameters that have been set up in the DEFINE section as conditions in a SELECT statement. If you do this you must precede the variable name with a dollar sign.

Example: A simple select statement

```
select *
     from  invoice
end
```

Example: Using variables in a select statement.

```
select invno, invdate, cnum, custpo, shipdate, shipcharge
     from   invoice
     where shipdate between $begin_date and $end_date
     order by invno
end
```

Example: Joining in a select statement.

```
select  customer.cnum, fname, lname,
     invno, invdate, custpo, shipdate,shipcharge
     from customer, invoice
     where customer.cnum = invoice.cnum and
     shipdate between $begin_date and $end_date
end
```

Example: Multiple table joining in a select statement.

```
select customer.cnum, fname, lname,
       invoice.invno, invdate, custpo, shipdate,
       shicharge, itemno, stockno, mfcode, qty, itemcost
     from customer, invoice, item
     where customer.cnum = invoice.cnum and
           invoice.invno = item.invno and
           shipdate between $begin_date and $end_date
end
```

CHAPTER EXERCISE

Modify your report **cust_rpt** to include:

The use of your variables in the select statement. You should select customers with

cnum between start_num and end_num

Order the customer list by customer number (cnum).

Run your report.

Remember Sam's customer numbers start with 100.

HINT: Don't forget to proceed your variables with a $ in the SELECT statement.

Don't forget the keyword END.

The syntax for an ACE report specification is summarized below.

```
DATABASE database-name END

DEFINE
     VARIABLE variable-name datatype
     PARAM[integer] variable-name datatype
END

INPUT
     PROMPT FOR variable-name USING "message string"
END

OUTPUT                                          Defaults
     LEFT MARGIN              integer           6
     RIGHT MARGIN             integer           132
     TOP MARGIN               integer           3
     BOTTOM MARGIN            integer           3
     PAGE LENGTH              integer           66
     REPORT TO                "filename"        screen
     REPORT TO PIPE           "program"
     REPORT TO PRINTER
END

SELECT statement;
 SELECT statement
END

FORMAT EVERY ROW END
```

Note: Unlike PERFORM, each section in an ACE report specification must end with the keyword END.

CHAPTER EXERCISE

Sam wants you to create a report for the stock table. He wants to be able to choose a manufacturers code, and then see a stock list for that manufacturer.

ASSIGNMENT

1) Create a default report for the stock table.

2) Add the following prompt:

Enter a 3-letter manufacturer's code (in UPPER case):

Remember: You will need to define a 3 character variable to hold this information.

3) Set the margins to zero and the page length to 22 lines.

4) Use the information entered at the prompt to restrict the SELECT statement to only those stock items from that manufacturer.

5) Order the report by mfcode and stockno.

Chapter Objectives

At the end of this lesson, you will be able to produce reports which:

- formats the selected rows
- produce aggregate calculations
- group similar rows under subheadings
- create a file containing SQL statements that may then be executed

The FORMAT Section

Once you have set up your ACE report you are ready to format the data to appear as you desire. Below is the syntax for the FORMAT section of an ACE report. It consists of the keyword FORMAT and several CONTROL BLOCKS.

```
FORMAT

    FIRST PAGE HEADER

    PAGE HEADER

    BEFORE GROUP OF

    ON EVERY ROW

    AFTER GROUP OF

    PAGE TRAILER

    ON LAST ROW

END
```

The Default Report Specification

```
database stores end

select
     cnum,
     fname,
     lname,
     company,
     address1,
     address2,
     city,
     state,
     zip,
     phone
from customer end

format every row end
```

The default report has the "format every row end" syntax.

The default format is adequate for an add-hoc query but seldom good enough for an ACE report.

We will first look at how to format a row in an ACE report.

Control Block for ON EVERY ROW

The first control block we will examine is the ON EVERY ROW block. This block tells ACE what to do with each row returned by a select statement. ON EVERY ROW is a major contributor to the appearance of an ACE report.

ON EVERY ROW is followed by a **procedural statement** or statments. There are many procedural statements that can be used with the ON EVERY ROW block.

```
on  every row
    print cnum using "#####",
          column  8, fname clipped,
          column 20, lname clipped,
          column 40, city clipped, ",  , state,
          2 spaces, zip
```

Procedural Statements

You specify when you want an action performed with the control blocks. A procedural statement specifies what action is to be performed.

PRINT [expr-list][;]

Prints the contents of the expr-list.
The semi-colon suppresses the line feed.

Expr-list can contain literal text, column names,variables, and some special keywords:

COLUMN #	Used to move the print head to particular column on the page.
# SPACE(S)	Moves print head the indicated number of spaces in the current line.
column USING expr	Display column in the format designated by expr. Expr refers to a set of characters with special formatting characteristics: < left justification # right justification $ display the dollar sign & zero padding USING applies to numeric and date data only.
column CLIPPED	Used to remove trailing spaces from character data. CLIPPED is for character data only.
PAGENO	A keyword that evaluates to the current page number.
TODAY	A keyword that evaluates to today's date. Printed as mm/dd/yyyy.

Report Specification (cust_rpt.ace)

```
database stores end

define
      variable start_num integer
      variable end_num integer
end

input
      prompt for start_num using
      "Enter starting customer number: "
      prompt for end_num using
      "Enter ending customer number: "
end

output
      page length 22
      left margin 0
      report to pipe "more"
end

select  cnum, fname, lname, company, address1,
          address2, city, state, zip, phone
      from customer
      where cnum between $start_num and $end_num
      order by cnum
end

format
      on every row
            print    cnum using "#####",
            column  8, fname clipped,
            column 20, lname clipped,
            column 40, city clipped, ", ",
                       state, 2 spaces, zip
end
```

Chapter Exercise

Sam would like you to modify the stock report that you created in the last chapter. He wants the mfcode to be in the left margin, stockno to be in column 10, description to be in column 20, unitprice to be in column 40, on_hand to be in column 55, and ro_point to be in column 65.

He would also like you to remove the prompt for a manufacturer code. He wants this list to incorporate all manufacturers (you might just comment out the input statement; you never know when Sam will want that piece back in the report).

The output for the report should look something like the output below:

Columns

```
ANZ     5     tennis racquet    $19.80     1980   400
ANZ     6     tennis balls       $4.80     5000   2000
ANZ     8     volleyball        $84.00     500    200
ANZ     9     volleyball net    $20.00     700    350
HRO     1     baseball glove    $35.00     450    500
HRO     2     baseball          $12.60     1300   400
HRO     4     football          $48.00     720    200
HRO     7     basketball        $35.00     175    40
HSK     1     baseball glove    $80.00     200    50
HSK     3     baseball bat      $24.00     900    100
HSK     4     football          $96.00     400    500
NRG     5     tennis racquet    $28.00     2000   400
SMT     1     baseball glove    $45.00     700    100
SMT     5     tennis racquet    $25.00     2000   400
SMT     6     tennis balls       $3.60     6000   3000
```

Control Blocks for Headers & Trailers

Most reports have some kind of title page and usually some column headers. We will look at how these features are accomplished in an ACE report through the use of page headers and page trailers.

FIRST PAGE HEADER

PAGE HEADER

PAGE TRAILER

Any of the above control blocks can use the print command to place custom titles or column headers onto a page.

Header & Trailer Example

```
format

        first page header
              print column 30, "CUSTOMER LIST"
              print column 30, "As of ", today
              print
              print "Cust #",
                    column 8, "First,",
                    column 20, "Last",
                    column 40, "Location"

        page header
              print column 30, "CUSTOMER LIST CONT..."
              print
              print "Cust #",
                    column 8, "First,",
                    column 20, "Last",
                    column 40, "Location"

         every row
              print       cnum using "#####",
                    column  8, fname clipped,
                    column 20, lname clipped,
                    column 40, city clipped, ",  ", state, 2 spaces, zip

        page  trailer
              print "-", pageno using "&<", "-"

end
```

Procedural Statements

The following procedural statements do not have to be placed in a header or a trailer, but they are commonly used there.

PAUSE [string]

A statement that causes scrolling to stop when output goes to the screen. The "string" is for an optional comment. Pressing RETURN begins scrolling again. This command is ignored if output DOES NOT go to the screen.

SKIP integer LINES

Skips the indicated number of lines in the report.

SKIP TO TOP OF PAGE

Causes subsequent printing to begin at the top of the next page.

Procedural Statement Example

```
format

        first page header
              print column  30, "CUSTOMER LIST"
              print column 30, "As of ", today
              skip 1 line
              print "Cust #",
                    column 8, "First,",
                    column 20, "Last",
                    column 40, "Location"
              skip1 line

        page header
              print column 30, "CUSTOMER LIST CONT..."
              print
              print "Cust  #",
                    column 8, "First,",
                    column  20, "Last",
                    column 40, "Location"
              skip 1 line

        on every row
              print  cnum using "#####",
                    column 8, fname clipped,
                    column 20, lname clipped,
                    column 40, city clipped, ",  ", state, 2 spaces, zip

        page trailer
              PAUSE "Press RETURN to continue ...."

end
```

CHAPTER EXERCISE

Sam would like the stock report that you have been working on to look like the outline that he has given below.

STOCK LIST
As of 04/18/1989

MF Code	Stock #	Description	Unit Price	On Hand	ROP
ANZ	5	tennis racquet	$19.80	1980	400
ANZ	6	tennis balls	$4.80	5000	2000
ANZ	8	volleyball	$84.00	500	200
ANZ	9	volleyball net	$20.00	790	350
HRO	1	baseball glove	$35.00	450	500
HRO	2	baseball	$12.60	1300	400
HRO	4	fc tball	$48.00	720	200
HRO	7	basketball	$35.00	175	40
HSK	1	baseball glove	$80.00	200	50
HSK	3	baseball bat	$24.00	900	100
HSK	4	football	$96.00	400	500
NRG	5	tennis racquet	$28.00	2000	400
SMT	1	baseball glove	$45.00	700	100
SMT	5	tennis racquet	$25.00	2000	400
SMT	6	tennis balls	$3.60	6000	3000

Created by: John Doe -1-
PRESS RETURN TO CONTINUE

Specifications:

The title and date should only be on the first page.

The column headers should be on every page.

Your name and the page number should be on the bottom of every page.

Control Block for Last Row

After all the rows have been processed you may want to incorporate summary information into your report. One of the ways to do this is through the use of the ON LAST ROW control block.

ON LAST ROW is executed once for each run of an ACE report. It can be used to display values returned from an aggregate function or anything else that you want to do once all of the rows have been processed.

Aggregates in an ACE Report

An aggregate function returns information about rows. This information is summarized below:

Aggregate	Function
COUNT	returns an integer that is the total number of rows returned by the select statement.
PERCENT	evaluates COUNT as a percent of the total number of rows returned by the select statement (used when grouping data).
TOTAL	evaluates as the total of a column for all the rows that satisfied the select statement.
AVERAGE/AVG	evaluates as the average of a column for all the rows that satisfied the select statement.
MIN	evaluates as the minimum value of a column for all the rows that satisfied the select statement.
MAX	evaluates as the maximum value of a column for all the rows that satisfied the select statement.

Using Aggregate Functions in ACE

```
format
        first page header
              print column 30, "CUSTOMER LIST"
              print column 30, "As of ", today
              skip 1 line
              print  "Cust #",
                     column 8, "First,",
                     column 20, "Last",
                     column 40, "Location"
              skip 1 line

         page header
              print column 30, "CUSTOMER LIST CONT..."
               print
              print  "Cust #",
                     column 8, "First,"
                     column 20, "Last",
                     column 40, "Location"
              skip 1 line

        on every row
              print    cnum using "#####",
                     column  8, fname clipped,
                     column 20, lname clipped,
                     column 40, city clipped, ", ",
                                state, 2 spaces, zip

         page trailer
              PAUSE "Press RETURN to continue ...."

        on last row
              print "There were ",count using "<<", " customers printed."
              print "The highest Customer number is ", max of cnum
              print "The lowest  Customer number is ", min of cnum

end
```

CHAPTER EXERCISE

Sam wants to know the total count of how many things he sells.

He also wants to know the highest and lowest priced items.

The output should look something like the following:

STOCK LIST
As of 04/18/1989

MF Code	Stock #	Description	Unit Price	On Hand	ROP
ANZ	5	tennis racquet	$19.80	1980	400
ANZ	6	tennis balls	$4.80	5000	2000
ANZ	8	volleyball	$84.00	500	200
ANZ	9	volleyball net	$20.00	790	350
HRO	1	baseball glove	$35.00	450	500
HRO	2	baseball	$12.60	1300	400
HRO	4	football	$48.00	720	200
HRO	7	basketball	$35.00	175	40
HSK	1	baseball glove	$80.00	200	50
HSK	3	baseball bat	$24.00	900	1[illegible]0
HSK	4	football	$96.00	400	5[illegible]0
NRG	5	tennis racquet	$28.00	2000	400
SMT	1	baseball glove	$45.00	700	100
SMT	5	tennis racquet	$25.00	2000	400
SMT	6	tennis balls	$3.60	6000	3000

There were a total of 15 stock items.
The highest priced one was $96.00 and the lowest was $3.60
Created by: John Doe -1-
PRESS RETURN TO CONTINUE

Grouping Information in a Report

There may be times when you wish to see information grouped by a particular column. Before discussing grouping information in an ACE report, let us review the data returned in the last exercise.

ANZ	5	tennis racquet	$19.80	1980	400
ANZ	6	tennis balls	$4.80	5000	2000
ANZ	8	volleyball	$84.00	500	200
ANZ	9	volleyball net	$20.00	790	350
HRO	1	baseball glove	$35.00	450	500
HRO	2	baseball	$12.60	1300	400
HRO	4	football	$48.00	720	200
HRO	7	basketball	$35.00	175	40
HSK	1	baseball glove	$80.00	200	50
HSK	3	baseball bat	$24.00	900	100
HSK	4	football	$96.00	400	500
NRG	5	tennis racquet	$28.00	2000	400
SMT	1	baseball glove	$45.00	700	100
SMT	5	tennis racquet	$25.00	2000	400
SMT	6	tennis balls	$3.60	6000	3000

The select statement that produces the above output used an "*order by mfcode*" clause, resulting in five groups of mfcode. Because an "*order by mfcode*" was used with the select statement, the control blocks for grouping can now do their work.

Control Blocks for Grouping

There are 2 control blocks for grouping information, BEFORE GROUP OF and AFTER GROUP OF.

The BEFORE GROUP OF control block specifies what action ACE is to take before it processes a group of rows.

The AFTER GROUP OF control block specifies what action ACE is to take after it processes a group of rows.

A Group is determined by the ORDER BY clause of the SELECT section.

Using Grouping in an ACE Report

```
select *
        from customer
        order by state
end

format
        first page header
              print column 30, "CUSTOMER LIST"
              print column 30, "As of ", today

        before group of state
              skip 1 line
              print "Customers in ", state
              print  "Cust #",
                    column 8, "First,",
                    column 20, "Last",
                    column 40, "Location"
              skip 1 line

        on every row
              print     cnum using "#####",
                    column  8, fname clipped,
                    column 20, lname clipped,
                    column 40, city clipped, ",  ", state, 2 spaces, zip

        after group of state
              skip 1 line
              print "There are ", group count using "<<<<<",
                    " customers in ", state, "."
              skip to top of page

        on last row
              print "There were ",count using "<<", " customers printed."
              print "The highest Customer number is ", max of cnum
              print "The lowest  Customer number is ", min of cnum

end
```

Output

```
                                CUSTOMER LIST
                                As of 04/18/1989

Customers in CA

Cust #      First Last              Location

102         Charles Ream            Palo Alto,  CA  94304
104         Raymond Vector          Los Altos,  CA  94022
106         Philip Currie           Palo Alto,  CA  94303
107         Carole Sadler           San Francisco,  CA  94117
108         Ludwig Pauli            Sunnyvale,  CA  94086
111         Frances Keyes           Sunnyvale,  CA  94085
112         Margaret Lawson         Los Altos Hills,  CA  94022

                                -01-

                                CUSTOMER LIST CONT...

113         Lana Beatty             Menlo Park,  CA  94025
115         Alfred Grant            Menlo Park,  CA  94025
118         Bill Blakely                  Oakland,  CA  94609

There are 10 customers in CA.

                                -02-
```

CUSTOMER LIST CONT...

Customers in NV

Cust #	First Last	Location
109	Jane Miller	Reno, NV 24086

There are 1 customers in NV.

-03-

CUSTOMER LIST CONT...

Customers in OR

Cust #	First Last	Location
101	Donald Quinn	Redwood City, OR 84063
105	Anthony Higgins	Redwood City, OR 84026
110	Roy Jaeger	Redwood City, OR 84026
114	Frank Albertson	Redwood City, OR 84026
117	Arnold Sipes	Redwood City, OR 88068

There are 5 customers in OR.

-04-

CUSTOMER LIST CONT...

Customers in WA

Cust #	First Last	Location
103	George Watson	Mountain View, WA 44063
116	Jean Parmelee	Mountain View, WA 44040

There are 2 customers in WA.

-05-

CUSTOMER LIST CONT...

There were 18 customers printed.
The highest Customer number is 118
The lowest Customer number is 101

-06-

CHAPTER EXERCISE

Sam would like the stock report to be grouped by manufacturer code.

Make the changes to your stock report so that:

- Each manufacturer is on their own page.
- There is a heading before each group of stock.
- All pages have the same page header.
- After each group tell how many items are carried by that manufacturer.

An example of the output can be found on the next two pages.

EXAMPLE OUTPUT FOR EXERCISE

STOCK LIST
As of 04/18/1989

Stock listings for manufacturer: ANZ

MF Code	Stock #	Description	Unit Price	On Hand	ROP
ANZ	5	tennis racquet	$19.80	1980	400
ANZ	6	tennis ball	$4.80	5000	2000
ANZ	8	volleyball	$84.00	500	200
ANZ	9	volleyball net	$20.00	790	350

Total of 4 things manufactured by ANZ

Created by: John Doe

STOCK LIST
As of 04/18/1989

Stock listings for manufacturer: HRO

MF Code	Stock #	Description	Unitprice	On Hand	ROP
HRO	1	baseball glove	$35.00	450	500
HRO	2	baseball	$12.60	1300	400
HRO	4	football	$48.00	720	200
HRO	7	basketball	$35.00	175	40

Total of 4 things manufactured by HRO

Created by: John Doe

. . . . . .

There were a total of 15 stock items.

The lowest priced one was $3.60 and the highest was $96.00

Created by: John Doe

Procedural Statements

If-then-else & Let

IF . . . THEN . . . ELSE

The IF statement evaluates an expression and then takes conditional actions. If the action involves more than one statement, all the action statements must be between the keywords BEGIN and END

LET var-name = expr

The LET statement assigns a value to a defined variable.

hint: Initialize variables in the first page header clause. This clause is executed first and is only executed once.

Example Report

```
database stores end

define
        variable need_num smallint
end

output
        left margin 0
        page length 24
        report to "reord.out"
end

select  *
        from stock
        order by stockno, mfcode
end

format

        first page header
              let need_num = 0
              print "Stock #",
                    column 10, "MF. Code",
                    column 20, "Description",
                     column 52, "On Hand",
                    column 65, "Re-order"
              skip 1 line
```

```
        on every row
              if on_hand < ro_point then
              begin
                    print stockno using "#######",
                          column 10, mfcode,
                          column 20, description clipped,
                          column 52, on_hand using "##,###",
                           column 65, ro_point using "##,###",
                          column 75, "****"
                    let need_num = need_num + 1
              end
              else
                    print stockno using "#######",
                          column 10, mfcode,
                          column 20, description clipped,
                          column 52, on_hand using "##,###",
                          column 65, ro_point using "##,###"

         page trailer
              print "*** = Needs to be re-ordered!"

         on last row
              skip 1 lines
              print "There were a total of ",need_num using "<<", " items",
              " that need to be re-ordered."
end
```

Output for "reorder.ace"

Stock #	MF. Code	Description	On Hand	Re-order	
1	HRO	baseball glove	450	500	***
1	HSK	baseball glove	200	50	
1	SMT	baseball glove	700	100	
2	HRO	baseball	1,300	400	
3	HSK	baseball bat	900	100	
4	HRO	football	720	200	
4	HSK	football	400	500	***
5	ANZ	tennis racquet	1,980	400	
5	NRG	tennis racquet	2,000	400	
5	SMT	tennis racquet	2,000	400	
6	ANZ	tennis balls	5,000	2,000	
6	SMT	tennis balls	6,000	3,000	
7	HRO	basketball	175	40	
8	ANZ	volleyball	500	200	
9	ANZ	volleyball net	790	350	

There were a total of 2 items that need to be re-ordered.

*** = Needs to be re-ordered!

Using ACE to Manipulate Data

It is possible to create an SQL script using ACE. This script can be executed through the Query Language Interpreter and used to manipulate the database.

Consider the following select statement and it's output:

```
select stockno, mfcode, sum(qty) grpqty
        from item, invoice
        where item.invno = invoice.invno and invdate = today
        group by stockno,mfcode
```

Output:

stockno	mfcode	grpqty
1	HRO	100
8	ANZ	50
7	HRO	100
1	HSK	75
9	ANZ	200
6	SMT	2500

What does this information tell us?

It tells us how many of a particular stock item was sold "today". This information should be subtracted from the on_hand column of the stock table and the reorder report run again. Let's review an ACE report that can write the update statements for us.

ACE Report "updstock.ace"

```
database stores end

output
        report to "updstock.sql"
end

select stockno, mfcode, sum(qty) grpqty
        from item, invoice
        where item.ordno = invoice.ordno and
              orddate = today
        group by stockno, mfcode
end

format

         on every row
              print "update stock set stock.on_hand = stock.on_hand - ",
                      grpqty using "<<<"
              print " where stockno = ", stockno using "<<",
                     "and mfcode = '", mfcode, "';"
              skip 1 line
end
```

The output for this report would look like the following:

```
update stock set stock.on_hand = stock.on_hand - 100
where stockno = 1 and mfcode = 'HRO';


update stock set stock.on_hand = stock.on_hand - 50
where stockno = 8 and mfcode = 'ANZ';


update stock set stock.on_hand = stock.on_hand - 100
where stockno = 7 and mfcode = 'HRO';


update stock set stock.on_hand = stock.on_hand - 75
where stockno = 1 and mfcode = 'HSK';


update stock set stock.on_hand = stock.on_hand - 200
where stockno = 9 and mfcode = 'ANZ';


update stock set stock.on_hand = stock.on_hand - 2500
where stockno = 6 and mfcode = 'SMT';
```

Putting It All Together

Let's look at one last ACE report to see what else can be done when we put all of the pieces together. This ACE report will print invoices for our customers. It is assumed this report will be run on a nightly basis and will go generate an invoice for all orders placed today. A sample of the output is listed below:

Sams Sporting Goods
101 Main Street
Palo Alto, CA 91111
(415) 555-1212

04/04/1989

INVOICE

Raymond Vector
Los Altos Sports
1899 La Loma Drive Los Altos, CA 94022
Your customer number is: 104

****

Order #: 1002 Purchase Order #: B77890 Shipped Via
Ordered on 04/04/1988 Shipped on 04/28/1988 S & L Trucking

****

Item #	Stock #	MFC	Description	Qty	Unit Price	Tot al Cost
1	3	HSK	baseball bat	7@	21.60	$ 151.20
2	4	HSK	football	15@	86.40	$ 1,296.00
3	3	HSK	baseball bat	4@	21.60	$ 86.40

Sub Total	1,533.60
Tax	107.35
Shipping	10.00
Please Pay	1,650.95

We thank you for your order and look forward
to doing business with you in the future.

Code for "all_in_one.ace"

```
database stores end

define
        variable lineitem money
        variable grtotal money
end

output
        left margin 0
        report to "invoice.out"
end

select *, customer.cnum custnum, invoice.ordno ordnum
        from customer, invoice, item, stock
         where customer.cnum = invoice.cnum and
        invoice.ordno = item.ordno and
        item.stockno = stock.stockno and
        item.mfcode = stock.mfcode and
        orddate = today
        order by custnum, ordnum
 end
```

format

```
        page header
              print column 30,"Sams Sporting Goods"
              print column 31,"101 Main Street"
              print column 30,"Palo Alto, CA 91111"
              print column 32,"(415) 555-1212"
              skip 1 line
              print column 34,today
              skip 1 line
              print column 33,"I N V O I C E"
              skip 2 lines
```

Code for "all_in_one.ace" continued

```
before group of custnum
      print fname clipped, 1 space, lname
      print company
      print address1
      if address2 is not null then
            print address2
      print city clipped, ", ", state, 2 spaces, zip
      print "Your customer number is: ",custnum using "<<"
      skip 1 line

before group of ordnum
      let grtotal = 0
      print "****************************************",
            "****************************************"
      print "Order #: ", ordnum using "<<<<",
            column 25,"Purchase Order #: ",
            column 43,custpo,
            column 61,"Shipped Via"
      print "Ordered on ", orddate,
            column 25,"Shipped on ", shipdate,
            column 58,ship_via
      print "****************************************",
            "****************************************"
      skip 1 line
      print "Item #",
            column 10, "Stock #",
             column 20, "MFC",
            column 25, "Description",
            column 45, "Qty",
            column 52, "Unit Price",
            column 65, "Total Cost"
```

Code for "all_in_one.ace" continued

```
        on every row
              let lineitem = qty * itemcost
              let grtotal = grtotal + lineitem
              print itemno using "####",
                    column 10, stockno using "####",
                    column 20, mfcode,
                    column 25, description clipped,
                    column 45, qty using "#####",
                    column 50, "@",
                    column 52, itemcost using "####&.&&",
                    column 65, lineitem using "$###,##&.&&"

        after group of ordnum
              skip 1 line
              print "---------------------------------------",
                    "---------------------------------------"
              print column 52,"Sub Total",
                    column 65,grtotal using "####,##&.&&"
              print column 52,"Tax",
                    column 65,(grtotal * .07) using "####,##&.&&"
              print column 52,"Shipping",
                    column 65,shipcharge using "####,##&.&&"
              print column 52,"---------------------------"
              print column 52,"Please Pay",
                    column 64, ((grtotal) +
                               (grtotal * .07) +
                               shipcharge) using "#,###,##&.&&"
              skip 3 lines
              print file "thanks.txt"
              skip to top of page
end
```

Chapter 13.

The User-Menu System

Chapter Objectives

At the end of this lesson, you will be able to:

- incorporate your forms and reports into a menu environment that is easily understood by end users

Running a User Menu

The User Menu is a customized menu associated with a single INFORMIX-SQL database.

The menu created for the stores database is illustrated on the next page

When you run the User Menu, you will see the menu titled

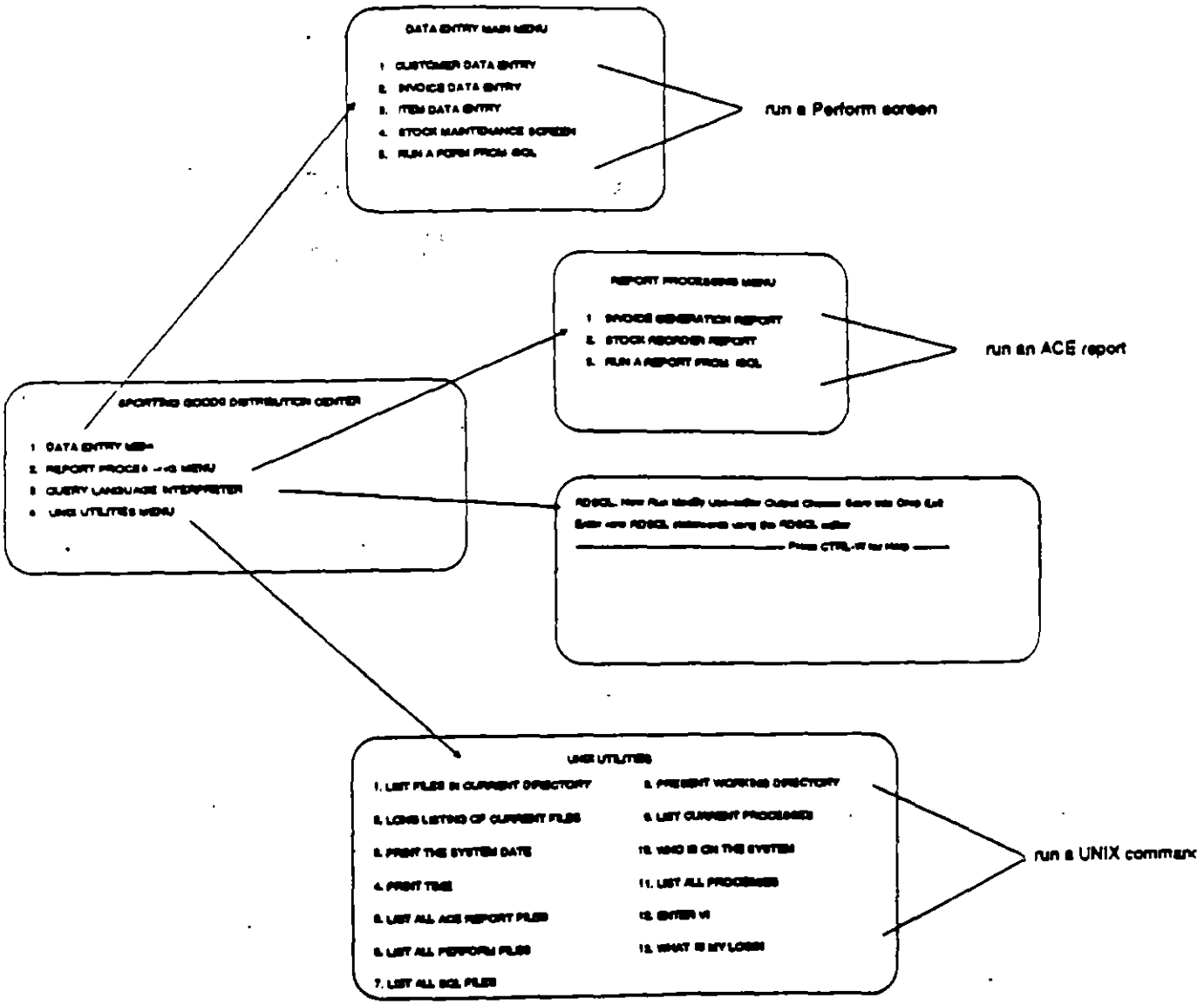
```
"SPORTING GOODS DISTRIBUTION CENTER".
```

When you select a menu item one of two things may occur:

1. A sub menu will appear.

2. An application will run.

Running The Stores User Menu



CHAPTER EXERCISE

From the main menu of Informix choose the **User-menu** option.

Choose **Run**.

You are now running the User Menu program which is interpreting the entries made in the sysmenus and sysmenuitems tables.

The menu has four options, three of the options call other menus and one option takes you into the Query Language Interpreter.

Creating A User Menu

User Menu is a user interface to the applications we have written throughout the course. It allows users access to your application without necessarily giving them access to the INFORMIX-SQL product.

User Menu is a program that takes information from two special tables and translates that information into the User Menu program. The two tables involved are listed below:

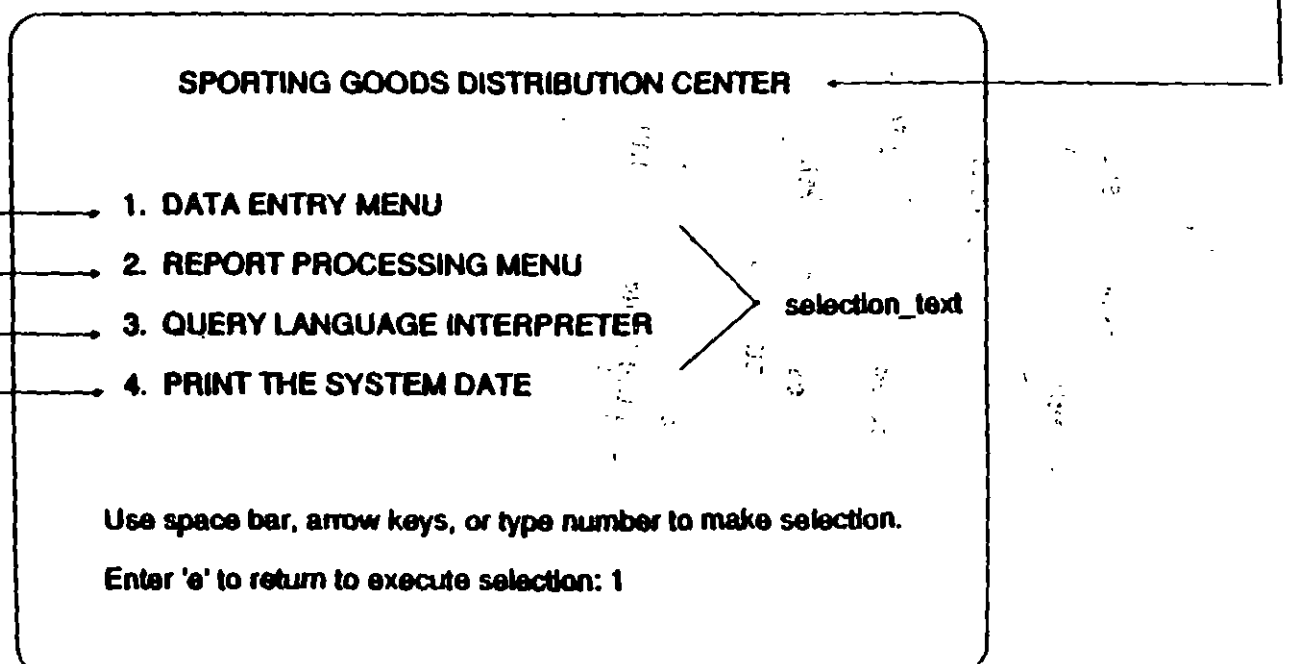
sysmenus	This table contains two pieces of information, the name of each menu for internal reference and the title of each menu for screen display.
sysmenuitems	This table contains information about each menu option, i.e. which table that option belongs to; what is displayed on the screen as the option description; what should happen when a user chooses that option.

sysmenuitems (detail)

menu_name	selection_number	selection_text	selection_action
main	1	DATA ENTRY MENU	forms
main	2	REPORT PROCESSING	reports
main	3	QUERY LANGUAGE	
main	4	PRINT THE DATE	date

sysmenus (master)

menu_name	menu_title
main	SPORTING GOODS DISTRIBUTION CENTER
forms	DATA ENTRY MAIN MENU
reports	REPORT PROCESSING MENU



Entering User-Menu Data

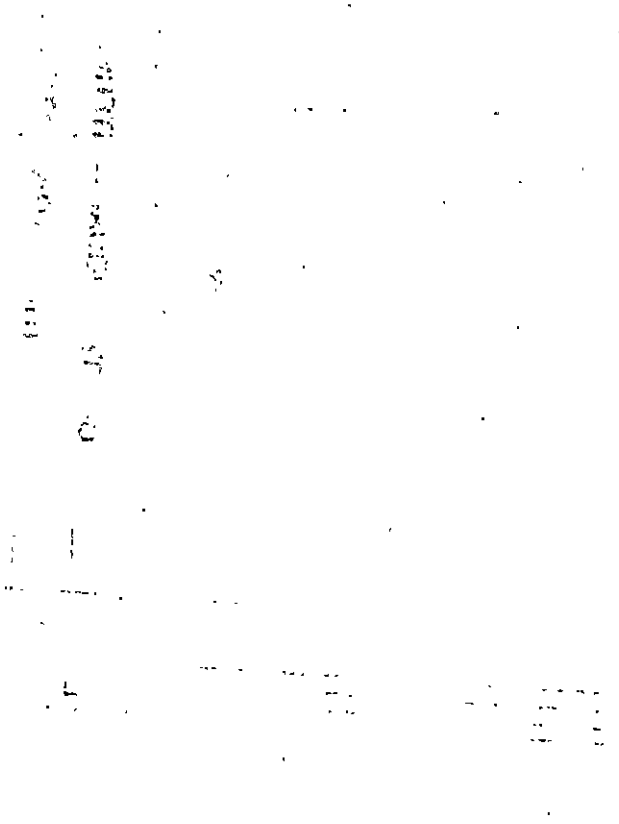
Rows are inserted into the sysmenus and sysmenuitems tables through a PERFORM screen form. This form has the following characteristics:

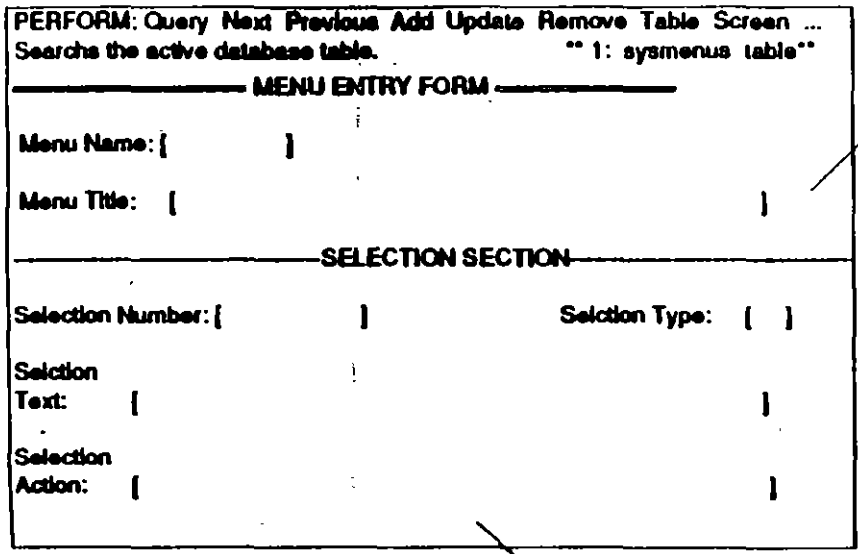
Displays a single screen

Accesses two tables: sysmenus and sysmenuitems

Defines a Master/Detail relationship with sysmenus MASTER OF sysmenuitems

The following diagram illustrates the relationship between the PERFORM screen and the tables sysmenus and sysmenuitems.





sysmenus (master)

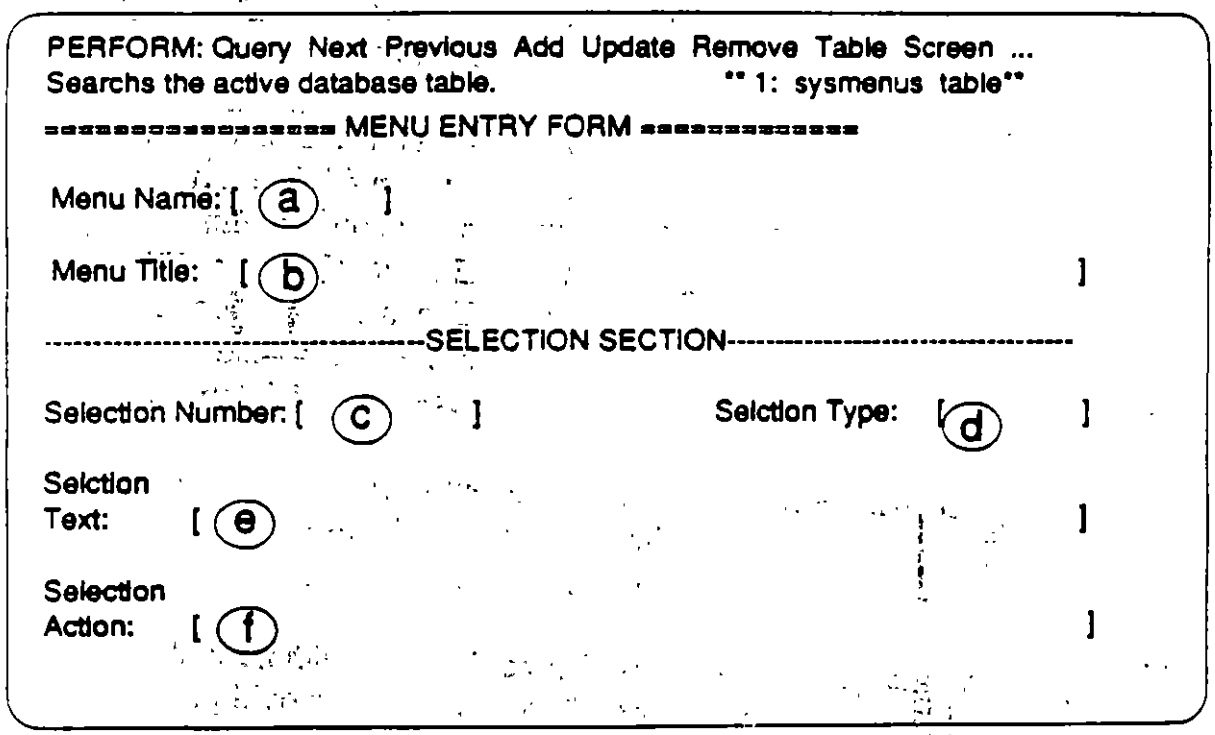
menu_name	menu_title
main	SPORTING GOODS DISTRIBUTION CENTER
forms	DATA ENTRY MAIN MENU
reports	REPORT PROCESSING MENU

sysmenuitems (detail)

imenu name	itemnum	mtype	mtext	progname
main	1	m	DATA ENTRY MENU	forms
main	2	m	REPORT PROCESSING MENU	reports
main	3	q	QUERY LANGUAGE INTERPRETER	
main	4	m	UNIX UTILITIES MENU	date

Entering Data

The PERFORM screen used to enter data into the sysmenus and sysmenuitems is explained below:



a = sysmenus.menuname: This is an internal name for a menu.

b = sysmenus.title: This what the user sees across the top of a menu screen.

c = sysmenuitems.itemnum: This the number which goes next to the choices available on a menu.

d = sysmenuitems.mtype: This field tells User-Menu what is going to happen when a particular choice is made from a menu. It is used in conjunction with field f to determine what action to take.
The available options here are:

F	Run a PERFORM form
R	Run an ACE report
M	Call another menu
Q	Execute the statements in a .sql file
P	Execute a program or an operating system command
S	Execute a script menu

e = sysmenuitems.mtext: This is the text which is placed next to each itemnum, it tells the user what each choice on a menu will do.

f = sysmenuitems.progname: Used in conjunction with field d. If d has a value of "f" then progname would be the name of a compiled PERFORM screen. If d has a value of "r" then progname would be the name of a compiled ACE report.

Sysmenus & Sysmenuitems

Below is a listing of the data currently contained in the sysmenus and sysmenuitems tables.

sysmenus

menuname	title
main	SPORTING GOODS DISTRIBUTION CENTER
forms	DATA ENTRY MAIN MENU
reports	REPORT PROCESSING MENU
unix_menu	UNIX UTILITIES

join column

sysmenuitems

imenuname	itemnum	mtext	mtype	progname
main	1	DATA ENTRY MENU	M	forms
main	2	REPORT PROCESSING MENU	M	reports
main	3	QUERY LANGUAGE INTERPRETER	Q	
main	4	UNIX UTILITIES MENU	M	unix_menu
forms	1	CUSTOMER DATA ENTRY	F	state_disp
forms	2	INVOICE DATA ENTRY	F	invfrm
forms	3	ITEM DATA ENTRY	F	four_tbl
forms	4	STOCK MAINTENANCE SCREEN	F	stockvrfy
reports	1	INVOICE GENERATION REPORT	R	all_in_one
reports	2	STOCK REORDER REPORT	R	reorder
reports	3	RUN A REPORT FROM ISQL	R	
unix_menu	1	LIST FILES IN CURRENT DIRECTORY	P	ls
unix_menu	2	LONG LISTING OF CURRENT FILES	P	ls -l
unix_menu	3	PRINT THE SYSTEM DATE	P	date
unix_menu	4	PRINT TIME	P	time
unix_menu	5	LIST ALL ACE REPORT FILES	P	ls *.ace
unix_menu	6	LIST ALL PERFORM FILES	P	ls *.per
unix_menu	7	LIST ALL SQL FILES	P	ls *.sql
unix_menu	8	PRESENT WORKING DIRECTORY	P	pwd
unix_menu	9	LIST CURRENT PROCESSES	P	ps
unix_menu	10	WHO IS ON THE SYSTEM	P	who
unix_menu	11	LIST ALL PROCESSES	P	ps -e
unix_menu	12	ENTER VI	P	vi
unix_menu	13	WHAT IS MY LOGIN	P	who am i

CHAPTER EXERCISE

Add a new menu to the current User Menu system. You will need to:

Add one row to the sysmenus table
- menu_name = mymenu
- menu_title = MY PROGRAMS FROM CLASS

Add four rows to the sysmenuitems table
- use selections numbers 1-4
- fill in the appropriate selection_type
- fill in the appropriate selection_action
- selection_text may be anything that makes sense to you

An example of the output for this exercise is shown below:

```
MY PROGRAMS FROM CLASS

1. One of my Perform Screens

2. Another one of my Perform Screens

3. One of my Reports

4. Another One of my Reports

Use space bar, arrow keys, or type number to make selection.
Enter 'e' to return to previous menu or exit.
Enter carriage return to execute selection: 1
```