

Cuaderno de Prácticas de FBD

Curso - 2019/20

Profesores de Prácticas:

Carlos de Mesa Mantilla, Nicolás Marín Ruiz

Juan Miguel Medina Rodríguez, Manuel Pegalajar Cuéllar

Olga Pons Capote, Antonio Gabriel López Herrera

Rocío Celeste Romero Zaliz y M. Amparo Vila Miranda

Pag. web de la asignatura: <http://prado.ugr.es>

Copyright © 2019 DECSAI

PUBLICADO POR DECSAI

DECSAI.UGR.ES

*Septiembre de 2019
Editado sobre una plantilla de latextemplates.com*

Índice general

0	Introducción	11
0.1	Introducción al SGBD Oracle	11
0.1.1	Nota histórica	11
0.1.2	Una visión de conjunto del sistema	12
0.2	Entorno de ejecución para las prácticas	12
0.3	Instalación del software de prácticas en un ordenador particular	12
0.4	Documentación, bibliografía y recursos	14
1	Definición del esquema de una base de datos	15
1.1	Conexión y acceso al SGBD	15
1.2	Cambiar la clave	16
1.3	Salir de SQL*Plus	16
1.4	Algunas facilidades	16
1.5	Ejecutar comandos de SQL	17
1.5.1	Descripción de una tabla	18
1.6	Sublenguaje de definición de datos en Oracle	18
1.6.1	Creación de tablas	18
1.6.2	Eliminación de tablas	20
1.6.3	Modificación del esquema de una tabla	20
1.7	Creación del esquema de la Base de Datos de prácticas	21
1.7.1	Creación de la tabla de Proveedores	21
1.7.2	Creación de la tabla de Piezas	22
1.7.3	Creación de la tabla de Proyectos	22
1.7.4	Creación de la tabla de Ventas	23
1.7.5	Modificación del esquema de la tabla de Ventas	23
1.8	Ejercicios adicionales	24
1.8.1	Creación del esquema de una Base de Datos sobre baloncesto	24
2	Mantenimiento de una base de datos	25
2.1	Lenguaje de manipulación de datos en Oracle	25
2.1.1	Inserción de tuplas en las tablas	25
2.1.2	Mostrar el contenido de una tabla	26
2.1.3	Modificar el contenido de una tabla	27

2.1.4	Borrado de tuplas	27
2.1.5	Particularidades del tipo de dato DATE	28
2.2	Inserción de tuplas en nuestra base de datos de ejemplo	29
2.3	Antes de salir...	31
2.4	Ejercicios adicionales	32
2.4.1	Inserción de tuplas en la Base de Datos sobre baloncesto	32
3	Realización de consultas a una base de datos	33
3.1	La sentencia de consulta SELECT	33
3.2	La consulta en SQL y su relación con los operadores del AR	34
3.2.1	La proyección AR en SQL	34
3.2.2	La selección AR en SQL	35
3.2.3	Consultas sobre el catálogo	36
3.2.4	Operadores AR sobre conjuntos en SQL	36
3.2.5	El producto cartesiano AR en SQL	37
3.2.6	El renombramiento o alias en SQL	38
3.2.7	La equi-reunión y la reunión natural AR en SQL	39
3.3	Ordenación de resultados	40
3.4	Subconsultas en SQL	40
3.4.1	IN, el operador de pertenencia	40
3.4.2	EXISTS, el operador de comprobación de existencia	41
3.4.3	Otros operadores, los comparadores sobre conjuntos	41
3.5	La división AR en SQL	42
3.5.1	Aproximación usando expresión equivalente en AR	42
3.5.2	Aproximación basada en el Cálculo Relacional	43
3.5.3	Aproximación mixta usando NOT EXISTS y la diferencia relacional	43
3.6	Funciones de agregación	43
3.6.1	Formando grupos	44
3.6.2	Seleccionando grupos	45
3.6.3	Subconsultas en la cláusula HAVING	46
3.7	Consultas adicionales	46
3.7.1	Consultas con el tipo DATE	46
3.7.2	Otras consultas sobre el catálogo	47
3.7.3	Ejercicios adicionales	48
3.8	Ejercicios adicionales	49
3.8.1	Realización de consultas sin operadores de agregación	49
3.8.2	Realización de consultas con operadores de agregación	50
4	Definición del nivel externo de un DBMS	53
4.1	Creación y manipulación de vistas	53
4.1.1	Consulta de vistas	54
4.1.2	Actualización de vistas	54
4.1.3	Eliminación de vistas	55
4.1.4	Ejercicios de vistas	55

5	Introducción a la administración: el catálogo y gestión de privilegios	57
5.1	Información acerca de la base de datos: las vistas del catálogo	57
5.1.1	Algunas vistas relevantes del catálogo de la base de datos	57
5.2	Gestión de privilegios	57
5.2.1	Privilegios del sistema	58
5.2.2	Privilegios sobre los objetos	59
5.2.3	Ejercicios de gestión de privilegios	60
6	Nivel interno: Índices, clusters y hashing	61
6.1	Creación y manipulación de índices	61
6.1.1	Selección de índices	61
6.1.2	Creación de índices	62
6.1.3	Índices compuestos	62
6.1.4	Estructura de los índices	63
6.1.5	Eliminación de índices	63
6.1.6	Creación y uso de otros tipos de índices	63
6.2	“Clusters”	65
6.2.1	Selección de tablas y clave del “cluster”	65
6.2.2	Creación de un “cluster” indizado	66
6.2.3	Creación de las tablas del “cluster” indizado	67
6.2.4	Creación del índice del “cluster”	67
6.2.5	Creación de un “cluster hash”	67
6.2.6	Creación de un “cluster hash” de una sola tabla	68
6.2.7	Eliminación de “clusters”	69
6.2.8	Eliminación de tablas del “cluster”	69
6.2.9	Eliminación del índice del “cluster”	69
A	Uso de SQL Developer	71
A.1	Instalación de SQL Developer	71
A.2	Creación de una conexión de Base de Datos	72
A.3	Ejecución de sentencias	72
A.4	Creación y edición visual de tablas	73
A.5	Inserción y actualización visual de tuplas	74
A.6	Exportación e importación de objetos y datos	75
A.6.1	Importación de datos	78
B	Uso de la herramienta WinRDBI	83
B.1	Introducción	83
B.2	Preparación de los datos	84
B.3	Notación BNF para describir los lenguajes	86
B.4	Álgebra Relacional (AR)	86
B.4.1	Correspondencia de la notación de teoría con la de WinRDBI	86

B.4.2	Esquemas y ejercicios adicionales	87
B.5	Cálculo Relacional Orientado a Tuplas (COT)	88
B.6	Cálculo Relacional Orientado a Dominios (COD)	88
C	Referencias	91

Índice de figuras

3.1	Instancia de la tabla ventas y el resultado de la consulta del ejemplo 3.20.	45
6.1	“Cluster” compuesto de las tablas proveedor y ventas, denominado cluster_codpro	66
B.1	Tipos de ficheros que soporta WinRDBI.	83
B.2	Ventana de WinRDB mostrando una instancia de la base de datos de las prácticas. .	85

Índice de cuadros

1.1	Tipos de datos básicos de SQL	20
1.2	Operadores básicos de SQL.	20
2.1	Elementos de un formato de fecha	29
5.1	Algunas vistas relevantes del catálogo	58
5.2	Ejemplos de privilegios sobre distintos objetos.	59
B.1	Correspondencia entre la notación usada en teoría y la proporcionada por WinRDBI	87

Capítulo 0

Introducción

Los objetivos generales que se persiguen con la elaboración del cuaderno de prácticas son:

- Familiarizar al alumno con los principales elementos de un sistema de gestión de bases de datos (SGDB) comercial, Oracle®.
- Conocer la configuración del sistema y del entorno de trabajo en el que se realizarán las prácticas de la asignatura.
- Conocer a nivel de usuario avanzado un lenguaje estándar para los sistemas de gestión de bases de datos relacionales, SQL. Concretamente, tras el desarrollo de las prácticas de la asignatura el alumno ha de adquirir conocimientos relativos a:
 - Creación y gestión de una base de datos sencilla
 - Realización de consultas a una base de datos
 - Gestión del nivel externo de un SGBD
 - Gestión del nivel interno de un SGBD

Algunos sistemas que usan SQL son: Oracle®, Sybase®, Microsoft® SQL Server, Access®, Ingres®, etc. Aunque la mayoría de estos sistemas tienen ciertas adaptaciones propias del lenguaje, en las prácticas se verán comandos estándares para la creación y manipulación de un esquema de base de datos.

Algunas consideraciones sobre este cuaderno de prácticas:

- El cuaderno está organizado en varias prácticas. Una práctica no se corresponde con una única sesión real en el laboratorio (2 horas), sino que contiene todo el material necesario para cubrir los objetivos que se proponen al inicio de la misma.
- Con objeto de conseguir un mejor aprovechamiento de los conocimientos que se exponen aquí, conviene leer detenidamente la información desarrollada en cada práctica y acudir a las sesiones en el laboratorio con los ejercicios planteados previamente.
- La realización de los ejercicios del cuaderno es voluntaria aunque muy recomendable ya que complementan y dan soporte a los conocimientos que se exponen en las clases teóricas y los seminarios.

0.1 Introducción al SGBD Oracle

0.1.1 Nota histórica

Oracle® es uno de los primeros sistemas de gestión de bases de datos que soportan completamente el modelo de datos relacional. Las sentencias del sublenguaje de definición de datos (DDL) incorporan los principales elementos de dicho modelo, incluyendo las reglas de integridad de entidad y referencial. Así mismo, su lenguaje de consulta tiene todas las características de un lenguaje de consulta relationalmente completo.

Oracle® fue desarrollado inicialmente por IBM® para el SYSTEM/R, pero en 1977 esta

compañía vendió Oracle® a Relational Software Inc., quién, en 1979, produjo la versión 2 del sistema. Esta versión fue el primer SGBD relacional disponible comercialmente y el primero que utilizó SQL como lenguaje de consulta y manipulación.

Desde esta fecha Oracle® ha ido desarrollando su sistema de gestión, creando al mismo tiempo productos adicionales y lanzando al mercado distintas versiones tanto del sistema básico como de los productos asociados. La versión de Oracle® disponible en el servidor, es la 12.1, sobre la que se va a trabajar en las aulas de prácticas.

0.1.2 Una visión de conjunto del sistema

Como hemos comentado antes, Oracle® es un sistema modular que se compone de distintos productos. Su elemento básico es el sistema de gestión de bases de datos (SGBD) Oracle®. Dispone de un catálogo extenso de productos para desarrollo y explotación, todos ellos basados en su servidor de BD y perfectamente acoplados con la filosofía de desarrollo Cliente/Servidor, Internet e Intranet. En las aulas de prácticas disponemos de dos tipos de clientes para interactuar con el SGBD: SQL*Plus y SQL Developer (Apéndice A).

Los productos más comunes que se encuentran en casi todas las instalaciones son:

- Oracle® RDBMS que, como ya hemos dicho antes, es el verdadero sistema de gestión.
- Utilidades de Oracle® que son las utilidades de administración de la base de datos.
- SQL*Plus es la herramienta básica de programación que permite a los usuarios la manipulación directa de la base de datos mediante SQL.

0.2 Entorno de ejecución para las prácticas

Para la realización de las prácticas sólo se precisa un cliente SQL. En nuestro caso vamos a utilizar SQL*Plus, aunque también se propone el uso de SQL Developer (Apéndice A), el cual es una herramienta más sofisticada que, además de permitir ejecutar sentencias SQL, dispone de muchas y útiles funcionalidades adicionales. Intentaremos proponer sesiones de trabajo que, supuesto un buen conocimiento de SQL, proporcionen un conocimiento básico de estas herramientas.

Este cuaderno también proporciona información básica acerca de otra herramienta de apoyo a la teoría, WinRDBI: WinRDBI es una herramienta desarrollada en Java que permite ejercitarse la ejecución de sentencias en los siguientes lenguajes: SQL, Álgebra Relacional, Cálculo Relacional orientado a Dominios y Cálculo Relacional orientado a Tuplas. Con lo cual resultará de gran ayuda para comprobar la corrección de las sentencias formuladas en esos lenguajes.

Todas estas herramientas están instaladas y adecuadamente configuradas en una “imagen” Windows XP, que se denomina “oraxp3”, que el alumno debe seleccionar al iniciar el ordenador de su puesto de prácticas para acceder a todo lo necesario para las sesiones programadas.

0.3 Instalación del software de prácticas en un ordenador particular

Existen versiones de uso gratuito u “Open Source” para todas las herramientas utilizadas en el programa de prácticas. En este apartado vamos a proporcionar la información necesaria para que, el alumno que lo desee, pueda configurar su ordenador particular con todo lo necesario para el seguimiento del programa de prácticas.

Conforme a lo visto en esta unidad, las herramientas que vamos a utilizar son: El SGBD Oracle®, los clientes SQL: SQL*Plus y SQL Developer, y WinRDBI. Veremos que se presentan varias alternativas a la hora de configurar una instalación doméstica de este software.

En primer lugar, el alumno debe optar por utilizar el SGBD Oracle® de la Escuela o por instalar uno en su ordenador.

Veamos qué debe hacerse para implantar la primera alternativa. En este caso, los datos estarán en la cuenta de alumno en el Servidor de Oracle® de la Escuela y éste interactuará con ellos mediante un cliente SQL.

Como la dirección IP de la máquina donde está instalado el servidor de Oracle® de la Escuela es privada, sólo se puede acceder al mismo desde la subred de la UGR. Esto significa que, para accesos externos, debe configurarse una VPN y conectarse a la misma, con lo que, a partir de ese momento, se tendrá acceso a dicho servidor. Si el alumno tiene configurada una conexión inalámbrica eduroam y se conecta a Internet a través de ella, estará en al subred de la UGR y, por tanto, también podrá acceder al Servidor Oracle® de la Escuela.

En [16] pueden consultarse las instrucciones para la configuración y el uso de la VPN de la UGR (Recomendamos la configuración de la VPN SSL). En la referencia [17] se proporcionan las instrucciones para configurar dicho acceso inalámbrico a la UGR, configuración que además permite disponer de acceso inalámbrico en multitud de Universidades e instituciones educativas de ámbito nacional e internacional a través de nuestra cuenta inalámbrica de la UGR.

El siguiente paso para acceder al servidor de la Escuela es instalar y configurar un cliente SQL. Eso lo describiremos más adelante, ahora mostraremos las alternativas en el caso de que el alumno opte por instalar el SGBD Oracle® en su ordenador particular.

Para esta opción, el alumno dispone de dos alternativas: instalar la versión completa del servidor Oracle® 12.1 EE, para la que dispone licencia para uso docente mientras sea alumno de esta Escuela, o descargar la versión gratuita de dicho servidor, Oracle® XE 11.2 (opción recomendada). Ambas versiones incluyen un cliente SQL*Plus y cuentan con todas las funcionalidades que precisamos para el desarrollo del programa de prácticas, además de disponer de versiones para Windows y para Linux.

Los tutoriales “InstalacionXE.pdf” e “InstalacionXE_linux.pdf” proporcionan las instrucciones para descargar e instalar Oracle® XE en entornos Windows y Linux, respectivamente. Ambos están disponibles en el apartado de “Material de Prácticas” del portal docente [5]. También están disponibles en dicho apartado una serie de videotutoriales para la instalación y configuración de Java, Oracle® XE 11.2 y SQLDeveloper para Windows y para Linux, podéis acceder también desde este enlace [1].

Si hemos optado por la primera alternativa, es decir, trabajar con nuestra cuenta en el servidor Oracle® de la Escuela, habremos de instalar un cliente SQL. En el caso de que queramos usar SQL*Plus, debemos seguir el tutorial “instalacionClienteXE.pdf”, que se encuentra en el mencionado portal docente. Para instalar y configurar SQL Developer, debemos seguir las instrucciones que se proporcionan en el Apéndice A.1.

Para la descarga e instalación de WinRDBI es preciso acudir a la web de la herramienta [2] y, previo registro gratuito, descargarla y seguir las instrucciones de instalación descritas en [3]. El proceso de instalación también instala la documentación de la herramienta. La instalación de la versión de WinRDBI para Windows no reviste ninguna complejidad siguiendo las instrucciones de [3]. Sin embargo, la instalación de la versión para Linux es algo más complicada, por lo que vamos detallar aquí los pasos a seguir para la instalación de WinRDBI en Linux 64 bits (probado, al menos, con Fedora 16 64 bits):

- Bajarse de [2] el fichero de WinRDBI para Linux.
- Ejecutar la instalación con: `java -jar WinRDBI-4.3.160_06-linux.jar`.
- Descargarse desde aquí [10] una máquina virtual de Java JRE para 32 bits (la arquitectura debe ser x86), por ejemplo la `jre-7u3-linux-i586.tar.gz`.
- Descomprimir el fichero `.tar.gz` con la orden: `tar zxvf <fichero>`.
Por ejemplo, `tar zxvf jre-7u3-linux-i586.tar.gz`.
- Conectado como root, mover la carpeta descomprimida a la carpeta `/usr/java` (si no existe el directorio `/usr/java`, creadlo con la orden `mkdir /usr/java`) con la orden `mv`.

Por ejemplo: `mv jre1.7.0_03 /usr/java`.

- Crear un script llamado `~/WinRDBI/WinRDBI` (el directorio `~/WinRDBI` lo crea la instalación si no se le cambia) con el siguiente código:

```
#!/bin/bash
cd ~/WinRDBI
JAVA_HOME=/usr/java/jre1.7.0_03
PATH=$JAVA_HOME/bin:$PATH
LD_LIBRARY_PATH=~/WinRDBI:$JAVA_HOME/lib:$LD_LIBRARY_PATH
java -cp WinRDBI.jar:xerces.jar:amzi.jar WinRDBI
```

- Cambiar los permisos del script para ejecución con la orden: `chmod u+x /WinRDBI/WinRDBI`.
- Instalar la biblioteca de compatibilidad de C++ `compat-libstdc++-33` para x86.

Para ello empleamos la orden: `yum install compat-libstdc++-33.i686`.

Con esto, terminaría la instalación y preparación para ejecución del programa. Para invocar al programa, hay que ejecutar la orden `~/WinRDBI/WinRDBI`

Más información acerca de la herramienta puede encontrarse en el Apéndice B.

0.4 Documentación, bibliografía y recursos

En esta prácticas se van a ejercitara elementos de SQL, de SQL*Plus y de SQL Developer. Para acceder a los manuales de referencia de dichas herramientas, a su descarga y a multitud de recursos adicionales, es conveniente que el alumno se registre de forma gratuita en la “web” técnica de Oracle® [15]. Una vez registrado, puede descargar las últimas versiones de las herramientas utilizadas a través de [13] y acceder a la documentación de dichas herramientas, que puede consultar en línea, o descargar directamente, a través de [14].

Además, el alumno puede acceder a una numerosa biblioteca de tutoriales acerca del uso de las mencionadas herramientas y de otras muchas otras a través de [9].

Téngase en cuenta que alguno de los enlaces mencionados arriba pueden cambiar en función de la propia organización de la “Web de Oracle®” y que, aunque tratemos de actualizar el cuaderno con los nuevos enlaces, eso no siempre podrá realizarse de forma inmediata, por lo que el alumno sólo habrá de utilizar un buscador para encontrar la información que se menciona, que se suministra a título orientativo.

A parte de la información mencionada, el alumno puede recurrir a la bibliografía propuesta para la asignatura, así como, a completar con la ingente cantidad de recursos relativos disponibles en Internet.

Capítulo 1

Definición del esquema de una base de datos

Los objetivos de esta práctica son:

1. Familiarizar al alumno con el entorno de trabajo en el que se van a desarrollar las prácticas en el SGDB Oracle®.
2. Ver tipos de datos y algunas de las operaciones disponibles sobre éstos.
3. Creación y modificación de esquemas relacionales.
4. Inserción, actualización y eliminación de tuplas.

Al final esta práctica dispondremos de las tablas de la base de proveedores, piezas, proyectos y ventas, que nos servirán como ejemplo para ilustrar diferentes aspectos del uso de un DBMS.

Esta unidad va utilizar SQL*Plus como herramienta cliente para la realización de los ejercicios propuestos, sin embargo, se anima al alumno a que pruebe a realizar algunos de dichos ejercicios usando SQL Developer A.

La documentación de referencia para esta unidad y siguientes es [8] y [12]. La documentación relativa al uso de SQL Developer puede encontrarse en [11]. El Apéndice A proporciona una introducción al uso de SQL Developer.

1.1 Conexión y acceso al SGBD

El entorno de trabajo que se va a utilizar será Windows XP. El procedimiento para trabajar con Oracle® consta de los siguientes pasos:

1. Iniciar el ordenador.
2. Tenemos que identificarnos a través de la ventana de diálogo que aparece . Los tres campos se deben llenar con nuestros datos de usuario de la ETSII, **login**, **clave** y el **código** a introducir es *oraxp3*. (En ocasiones, dependiendo de la configuración del ordenador, tamaño pequeño de disco, es necesario reiniciar el ordenador con Control+ALT+Del y proceder como en el paso 2).
3. Se produce la descarga e inicialización de una instalación de Windows XP habilitada para acceder al *servidor Oracle®* desde SQL*Plus.
4. Seleccionamos el programa SQL Plus desde el escritorio.
5. Introducimos **login** de usuario: **x**+los dígitos del dni sin incluir el primero, esto es, la **x** sustituye el primer dígito del dni o pasaporte; como **clave** la misma cadena del login¹ anterior.

¹Esta es la clave asignada por defecto, cuyo valor veremos como se cambia más adelante.

6. Si todo va bien, estaremos conectados al servidor Oracle® y aparecerá el “prompt” de SQL*Plus.

SQL>

7. Cambiar el directorio de trabajo, por ejemplo, u :\FBD:

SQL>cd u :\FBD

Ya está todo dispuesto para introducir cualquier sentencia SQL o comando SQL*Plus. Indicar aquí, que ninguno es sensible a mayúsculas o minúsculas.

1.2 Cambiar la clave

La primera vez que nos conectamos se recomienda cambiar nuestra clave en Oracle® que, como hemos indicado inicialmente, coincide con la cadena del login. El cambio se realiza mediante la sentencia SQL, ALTER USER, en la forma que se indica:

SQL> ALTER USER x-login IDENTIFIED BY password;

donde, x-login es el identificador de vuestra cuenta en el sistema, y está compuesto de x seguido de los restantes dígitos del dni o pasaporte p.e. x2345678 y password es la nueva “clave” que se quiere establecer para acceder a Oracle®, (el alumno debe recordar la nueva clave para accesos sucesivos). Como hemos indicado, las palabras reservadas como ALTER ... se pueden introducir en mayúsculas o no.

Una vez ejecutada esta sentencia, necesitaréis introducir ese “login” y esa “clave” cada vez que entréis a Oracle® desde el cliente SQL*Plus o desde SQL Developer. Consultar A.2 para ver como configurar una conexión en esta última herramienta.

1.3 Salir de SQL*Plus

Para finalizar cualquier sesión de trabajo, se recomienda ejecutar desde la línea de comandos la instrucción COMMIT; que guarda los cambios realizados en la sesión de trabajo. Y finalmente, para abandonar el entorno con el comando EXIT;

1.4 Algunas facilidades

Paginar la información

En ocasiones los resultados de consultas pueden ser muy extensos para ser mostrados en pantalla de forma aceptable, por ello es conveniente activar los comandos de paginado con:

set pause on;

y luego

SQL>set pause '.....Pulsa una Tecla para continuar.....'

lo que hace que el barrido de pantalla se detenga cada vez que se llena ésta y nos permita observar los resultados de nuestras operaciones con mayor comodidad.

Guardar una copia de lo realizado

Este comando sirve para dirigir la salida de la pantalla de SQL*Plus a un cierto fichero. De esta forma se guardan tanto los comandos como las salidas de éstos. Su forma general es:

```
SPO[OL] [file_name [.ext ] | OFF | OUT ]
```

- file_name representa el fichero de texto donde se quiere enviar la salida. Como puede verse la extensión es opcional. En el caso de que no la lleve se pone por defecto la extensión *lst*.
- OFF termina la salida al fichero.
- OUT termina la salida al fichero y envía este a la impresora.

Una vez finalizada la creación del fichero de salida mediante el comando SPOOL OFF podrá editarse cómodamente la salida abriendo el fichero generado mediante un editor de texto. El apéndice A.3 ilustra las posibilidades que presenta SQL Developer para ejecutar, visualizar y guardar los resultados de la ejecución de sentencias.

1.5 Ejecutar comandos de SQL

Desde la línea de comandos

Por ejemplo, para la creación de una tabla denominada *prueba1*.

```
SQL> CREATE TABLE prueba1 (
      cad char(3),
      n int,
      x float);
```

Si todo ha ido bien, devolverá el mensaje TABLA CREADA; en caso contrario dará un mensaje de error adecuado (leedlo atentamente), en ocasiones se indica con un subrayado la palabra “responsable” del error.

Desde el buffer

En realidad SQL*Plus siempre trabaja con un fichero de comandos que, por defecto, se llama **afidiet.buf**. Este fichero contiene siempre último comando o bloque de sentencias que se ha tecleado en “línea”. El contenido de este fichero se puede editar tecleando simplemente *edit* y se vuelve a lanzar mediante el comando *run*. La introducción de “/” seguida de “Intro” también desencadena la ejecución del último bloque almacenado en el “buffer”.

Desde un fichero

Otra alternativa, recomendada, para trabajar es editar y lanzar ficheros de comandos que se pueden guardar y volver a relanzar desde SQL*Plus. Estos ficheros de comandos SQL se pueden crear con cualquier editor (conviene dejarlos en un directorio de Bases de Datos preparado a tal efecto), que deberían tener extensión *.sql*. Para ello hay varias posibilidades:

1. Tecleando *edit nombre-de-fichero*, siendo éste un fichero no creado previamente y escribiendo en él antes de abandonar el editor.
2. Tecleando *save nombre-de-fichero*. De esta forma se copia el contenido del buffer al fichero en cuestión.
3. Utilizar un editor de texto externo como “wordpad” de Windows, para crear y almacenar el archivo de sentencias.

En las situaciones 1 y 2, el fichero se crea con la extensión adecuada y se almacena en el directorio de usuario correspondiente. En el último caso es necesario renombrar el fichero para establecerle la extensión (sql), ya que el editor le asigna otra por defecto.

Para ejecutar un fichero desde SQL*Plus, basta teclear `start camino\nombre-de-fichero` o bien `@camino\nombre-de-fichero`. Donde `camino` es el camino absoluto o relativo desde el que se ejecuta SQL*Plus (por defecto está en el directorio de instalación `Home\Oracle\bin`). Por simplicidad se recomienda establecer el directorio de trabajo, en el entorno, al directorio donde están nuestros archivos sql. Para ello basta con seleccionar en la ventana de SQL*Plus la opción OPEN del menu FILE y navegar hasta la carpeta donde están nuestros archivos para después pulsar el botón CANCEL.²

Ejercicio 1.1 Editar el fichero **pp.sql** en el directorio de trabajo, debe contener las instrucciones:

```
CREATE TABLE prueba2(
    cad1 char(8),
    num int);
```

a continuación ejecutar el fichero. Mediante esta sentencia hemos creado una tabla vacía llamada **prueba2** con dos atributos, **cad1** y **num**. Para comprobarlo, es necesario consultar el catálogo de la base de datos o ejecutar un comando `describe`, como veremos a continuación.

1.5.1 Descripción de una tabla

Una vez creada una tabla, podemos consultar su esquema y conocer algunas restricciones básicas asociadas a cada atributo. Para ello, es necesario utilizar la siguiente sentencia:

```
DESCRIBE nombre-tabla;
```

Hay que tener en cuenta que este es un comando SQL*Plus, no una sentencia SQL estándar. Para más detalles sobre las restricciones de tablas, será necesario consultar el catálogo, que veremos más adelante.

El uso de la funcionalidades descritas en esta sección desde SQL Developer puede consultarse en A.3.

Ejercicio 1.2 Ver la descripción de las tablas *prueba1*, *prueba2*.

1.6 Sublenguaje de definición de datos en Oracle

El sublenguaje de definición de datos (DDL) de SQL proporciona las sentencias para la creación y modificación de objetos de la base de datos, como pueden ser: tablas, vistas, índices, “clusters”, etc. En este apartado vamos a ejercitarnos las sentencias DDL relativas a la creación y mantenimiento de tablas usando SQL*Plus, para la realización de dichas operaciones desde SQL Developer, consultese A.4.

1.6.1 Creación de tablas

Como ya hemos visto en ejercicios anteriores, la creación de una tabla se realiza mediante la sentencia CREATE TABLE. La versión básica de dicha sentencia incluye la definición de los

² Es un amago de apertura de un archivo, abrir un archivo no lo ejecuta, sólo lo carga en el “buffer” de SQL*Plus, para ejecutarlo habría que introducir “/” seguida de “Intro”.

atributos y sus tipos de datos correspondientes, el valor por defecto que toma un atributo cuando no se especifica su valor al insertar una nueva tupla (cláusula DEFAULT), así como las claves primaria, candidatas y externas. A continuación se introduce la forma básica de su sintaxis:

```
CREATE TABLE nombre-tabla(
    nombre-atributo1 tipo-attributo1 [NOT NULL] [DEFAULT expr],
    nombre-atributo2 tipo-attributo2 [NOT NULL] [DEFAULT expr],...
    [PRIMARY KEY(nombre-atributo1, nombre-atributo2...),]
    [UNIQUE (nombre-atributo1, nombre-atributo2...),]
    [FOREIGN KEY(nombre-atributo1, nombre-atributo2...)
        REFERENCES nombre-tabla(nombre-atributo, ...),]
    [CHECK(condicion)]
);
```

Además, cabe destacar que cuando la clave primaria, la clave candidata o la clave externa está formada por un solo atributo, las palabras reservadas PRIMARY KEY, UNIQUE y REFERENCES, respectivamente, se podrán incluir a continuación de la definición del atributo correspondiente, tal y como se muestra a continuación:

```
CREATE TABLE nombre-tabla1(
    nombre-atributo1 tipo-attributo1 PRIMARY KEY,
    nombre-atributo2 tipo-attributo2 UNIQUE,
    nombre-atributo3 [tipo-attributo3]
    REFERENCES nombre-tabla2(nombre-atributo3));
```

La cláusula tipo-attributo3 puede omitirse, en cuyo caso el atributo será del mismo tipo que el atributo al que hace referencia.

■ **Ejemplo 1.1** Como ejemplo, vamos a considerar la tabla *plantilla*, donde vamos a almacenar el dni, nombre y fecha de alta de los trabajadores de una empresa, considerando dni como clave primaria. Algunos de los tipos de datos que ofrece SQL se pueden consultar en la Tabla 1.1.

```
CREATE TABLE plantilla(
    dni varchar2(9),
    nombre varchar2(15),
    estadocivil varchar2(10)
    CHECK (estadocivil IN ('soltero', 'casado', 'divorciado', 'viudo')),
    fechaalta date,
    PRIMARY KEY (dni));
```

Obsérvese que estamos delimitando el rango de valores para el atributo *estadocivil* mediante la sentencia CHECK. Los operadores que se pueden utilizar en estas expresiones quedan recogidos en la Tabla 1.2. ■

Ejercicio 1.3 Buscar la lista completa de los tipos de datos que ofrece Oracle® (Data types). Para ello debe consultarse el apartado correspondiente del manual de referencia de SQL de Oracle® [8]. ■

Además de definir reglas de integridad específicas sobre determinados campos, se pueden definir reglas de integridad genéricas tales como la regla de integridad de entidad y la regla de integridad referencial. Por ejemplo, si queremos almacenar la relación entre jefes y subordinados a partir de la tabla *plantilla*, podremos crear otra tabla con la siguiente estructura:

Tipo de dato	Descripción
INT ó INTEGER ó NUMERIC	Enteros con signo (su rango depende del sistema).
REAL ó FLOAT	Datos numéricos en coma flotante.
CHAR(n)	Cadena de longitud fija k .
VARCHAR(n)	Cadena de longitud variable de hasta n caracteres.
VARCHAR2(n)	Mínimo 1 carácter y máximo 4000. (Esta es una implementación de cadena más eficiente propia de Oracle®)
NUMBER(p,s)	Número con precisión p y escala s , donde precisión indica el número de dígitos, y escala el número de cifras decimales.
LONG	Cadena de caracteres de longitud variable de hasta 2 gigabytes (específico de Oracle®).
LONG RAW(size)	Cadena de datos binarios de longitud variable de hasta 2 gigabytes (específico de Oracle®).
DATE ó TIME ó TIMESTAMP	Fechas.

Cuadro 1.1: Tipos de datos básicos de SQL

Operador	Descripción	Ejemplo de uso
=, !=, <, >, <=, >=	Operadores relacionales	atributo1 <= atributo2
+, -, *, /	Operadores aritméticos	atributo1 = atributo2 + 8
	Concatenación de cadenas	atributo1 = 'Valor:' atributo2.
NOT, AND, OR	Operaciones lógicas	atributo1 >= 5 AND atributo1 <= 10
BETWEEN	Pertenencia a intervalo	atributo1 BETWEEN 5 AND 10
IN	Pertenencia a conjunto de escalares	IN ('soltero', 'casado')

Cuadro 1.2: Operadores básicos de SQL.

```
SQL> CREATE TABLE serjefe(
      dnijefe REFERENCES plantilla(dni),
      dnitrabajador REFERENCES plantilla(dni),
      PRIMARY KEY (dnitrabajador)
    );
```

Ejercicio 1.4 Deduce el diagrama E/R que recoge la semántica de las tablas *plantilla* y *serjefe*. ■

1.6.2 Eliminación de tablas

Se puede eliminar una tabla con todas las tuplas que contiene, liberando el espacio con la sentencia:

```
DROP TABLE nombre-tabla;
```

Ejercicio 1.5 Borrar la tabla *prueba1* y comprobar las tablas que quedan. ■

1.6.3 Modificación del esquema de una tabla

Para una tabla existente podemos utilizar la sentencia ALTER TABLE para modificar su estructura, por ejemplo añadiéndole una nueva columna, modificando la definición o las restricciones de alguno de sus atributos, o bien, eliminando algún atributo o restricción. Para

obtener más información acerca del uso de esta sentencia puede consultarse el manual de referencia de SQL de Oracle® [8]. La sintaxis básica es:

```
ALTER TABLE nombre_tabla modificador;
```

El tipo de alteración de la tabla dependerá del modificador que incluyamos. Por ejemplo, para añadir un atributo nuevo a una tabla se utiliza el modificador ADD del siguiente modo:

```
ADD (atributo [tipo] [DEFAULT expresion] [restriccion_atributo]);
```

Para añadir una restricción a una tabla existente el modificador sería:

```
ADD CONSTRAINT nombre_restriccion [[UNIQUE | PRIMARY KEY] (lista_columnas) |  
FOREING KEY (lista_columnas) REFERENCES tabla(lista_col) |  
CHECK (condicion)];
```

Para eliminar una restricción de una tabla el modificador a emplear sería:

```
DROP CONSTRAINT nombre_restriccion [CASCADE];
```

Ejercicio 1.6 Modifica el esquema de la tabla plantilla añadiendo un nuevo atributo llamado fechabaja de tipo date.

1.7 Creación del esquema de la Base de Datos de prácticas

En esta ocasión, vamos a crear las tablas que vamos a usar en próximas sesiones de prácticas. En la definición de estas tablas vamos a especificar claves primarias, claves externas, restricciones sobre campos, etc. Con objeto de poder reproducir estas operaciones, vamos crear los siguientes archivos de creación de las tablas: *proveedor*, *pieza*, *proyecto* y *ventas*.

1.7.1 Creación de la tabla de Proveedores

Se pide crear el archivo *proveedor.sql* con la información que se muestra a continuación. Destacar la palabra reservada *constraint* que permite dar nombre a restricciones impuestas a los atributos.

```
SQL> CREATE TABLE proveedor(  
codpro VARCHAR2(3) CONSTRAINT codpro_no_nulo NOT NULL  
CONSTRAINT codpro_clave_primaria PRIMARY KEY,  
nompro VARCHAR2(30) CONSTRAINT nompro_no_nulo NOT NULL,  
status NUMBER CONSTRAINT status_entre_1_y_10  
CHECK (status>=1 and status<=10),  
ciudad VARCHAR2(15));
```

Como resultado de la ejecución de este archivo, se crea la tabla de proveedores definiendo “codpro” como clave primaria y con una restricción de integridad sobre el valor de “status”.

Para ejecutar este fichero teclear:

```
SQL> @proveedor
```

Para ver el esquema de la tabla Proveedor se debe de utilizar el comando DESCRIBE recogido en la sección 1.5.1. Cuyo ejecución muestra el siguiente resultado:

Name	Null?	Type
CODPRO	NOT NULL	CHAR(3)
NOMPRO	NOT NULL	VARCHAR2(30)
STATUS		NUMBER
CIUDAD		VARCHAR2(15)

1.7.2 Creación de la tabla de Piezas

Puesto que el proceso es el mismo nos limitamos a mostrar el contenido del fichero pieza.sql:

```
SQL> CREATE TABLE pieza (
    codpie VARCHAR2(3) CONSTRAINT codpie_clave_primaria PRIMARY KEY,
    nompie VARCHAR2(10) CONSTRAINT nompie_no_nulo NOT NULL,
    color VARCHAR2(10),
    peso NUMBER(5,2)
    CONSTRAINT peso_entre_0_y_100 CHECK (peso>0 and peso<=100),
    ciudad VARCHAR2(15));
```

almacenar, salir del editor y ejecutar

```
SQL>start pieza;
```

La información acerca de la tabla es:

Name	Null?	Type
CODPIE	NOT NULL	CHAR(3)
NOMPIE	NOT NULL	VARCHAR2(10)
COLOR		VARCHAR2(10)
PESO		NUMBER(5,2)
CIUDAD		VARCHAR2(15)

1.7.3 Creación de la tabla de Proyectos

Editamos el fichero proyecto.sql, con la siguiente información:

```
SQL> CREATE TABLE proyecto(
    codpj VARCHAR2(3) CONSTRAINT codpj_clave_primaria PRIMARY KEY,
    nompj VARCHAR2(20) CONSTRAINT nompj_no_nulo NOT NULL,
    ciudad VARCHAR2(15));
```

almacenamos, salimos del editor y ejecutamos:

```
SQL>start proyecto;
```

La información acerca de la tabla es:

Name	Null?	Type
CODPJ	NOT NULL	CHAR(3)
NOMPJ	NOT NULL	VARCHAR2(20)
CIUDAD		VARCHAR2(15)

1.7.4 Creación de la tabla de Ventas

Editamos ventas.sql con el contenido:

```
SQL> CREATE TABLE ventas (
    codpro CONSTRAINT codpro_clave_externa_proveedor
    REFERENCES proveedor(codpro),
    codpie CONSTRAINT codpie_clave_externa_pieza
    REFERENCES pieza(codpie),
    codpj CONSTRAINT codpj_clave_externa_proyecto
    REFERENCES proyecto(codpj),
    cantidad NUMBER(4),
    CONSTRAINT clave_primaria PRIMARY KEY (codpro,codpie,codpj));
```

almacenamos, salimos del editor y ejecutamos.

Como puede verse hemos definido ventas con tres llaves externas a proveedor, pieza y proyecto y con llave primaria incluyendo tres de sus atributos.

```
SQL> start ventas;
SQL> describe ventas
Name           Null?    Type
-----          -----
CODPRO          NOT NULL CHAR(3)
CODPIE          NOT NULL CHAR(3)
CODPJ          NOT NULL CHAR(3)
CANTIDAD        NUMBER(4)
```

Comprueba las tablas que tenemos creadas hasta el momento En nuestro caso aparecerá

TABLE_NAME
PRUEBA2
PLANTILLA
SERJEFE
PIEZA
PROYECTO
PROVEEDOR
VENTAS

1.7.5 Modificación del esquema de la tabla de Ventas

Utilizando la sentencia ALTER TABLE, descrita anteriormente, vamos a modificar el esquema de la tabla Ventas añadiendo un nuevo atributo llamado *fecha* de tipo *date*.

Ejercicio 1.7 Comprobar que se ha cambiado correctamente el esquema de la tabla Ventas.
La descripción de la tabla debe contener los siguientes campos:

Name	Null?	Type
CODPRO	NOT NULL	CHAR(3)
CODPIE	NOT NULL	CHAR(3)
CODPJ	NOT NULL	CHAR(3)
CANTIDAD		NUMBER(4)

FECHA

DATE

Todos estos ejercicios se pueden realizar desde la “Hoja de Trabajo” de SQL Developer o, de forma visual, como se describe en el Apéndice A.4. Como ejercicios adicionales se deja al alumno trasladar al sistema el esquema de la base de datos de la Gestión docente universitaria utilizada en clase de teoría.

1.8 Ejercicios adicionales

1.8.1 Creación del esquema de una Base de Datos sobre baloncesto

Ejercicio 1.8 Dado el esquema siguiente de la base de datos de una liga de baloncesto:

Equipos (codE, nombreE, localidad, entrenador, fecha_crea)

Jugadores (codJ, codE, nombreJ)

Encuentros(ELocal, EVisitante, fecha, PLocal,PVisitante)

Faltas (codJ, ELocal, EVisitante, num)

Se pide que se cree dicho esquema con las siguientes restricciones de diseño:

Equipos: No se debe permitir que ninguno de los atributos tome valor nulo, además, el nombre del equipo ha de ser único.

Jugadores: No se debe permitir valor nulo ni en nombreJ, ni en el equipo al que pertenece.

Encuentros: Los encuentros se realizan entre equipos de la liga, cada uno de los atributos ELocal y EVisitante es clave externa a la tabla Equipos. Los resultados PLocal y PVisitante (tantos marcados por ELocal y por EVisitante, respectivamente) han de ser positivos y tomar 0 como valor por defecto.

Faltas: Representan la cantidad de faltas personales cometidas por un jugador en el encuentro indicado. El conjunto de atributos formado por ELocal y EVisitante son clave externa a la tabla Encuentros y el atributo codJ es clave externa a la tabla Jugadores. El número de faltas num estará comprendido entre 0 y 5, ambos incluidos y debe tomar 0 como valor por defecto.

Capítulo 2

Mantenimiento de una base de datos

Los objetivos de esta práctica son:

1. Introducir el uso de las sentencias SQL para la inserción, actualización y eliminación de tuplas.
2. Ilustrar algunas particularidades de uso del tipo de dato *date*.

Al final esta práctica habremos introducido tuplas en las tablas de la base de proveedores, piezas, proyectos y ventas, que nos servirán como ejemplo para ilustrar diferentes aspectos del uso de un DBMS.

Esta unidad va utilizar SQL*Plus como herramienta cliente para la realización de los ejercicios propuestos, sin embargo, se anima al alumno a que pruebe a realizar algunos de dichos ejercicios usando SQL Developer A.

La documentación de referencia para esta unidad y siguientes es [8] y [12]. La documentación relativa al uso de SQL Developer puede encontrarse en [11]. El Apéndice A proporciona una introducción al uso de SQL Developer.

2.1 Lenguaje de manipulación de datos en Oracle

En este sub-apartado vamos a ejercitarnos con las versiones básicas de las sentencias de inserción, actualización y borrado de SQL, pertenecientes a su sub-lenguaje de manipulación de datos (DML). Dejamos para la próxima práctica el ejercicio de la sentencia de consulta SELECT, también perteneciente a dicho sub-lenguaje.

Los aspectos que se ejercitan en este sub-apartado y el siguiente también se pueden llevar a cabo de forma visual desde SQL Developer conforme a como se describe en el Apéndice A.5.

2.1.1 Inserción de tuplas en las tablas

Una vez creadas las tablas, es preciso introducir tuplas en ellas. Para ello se hará un uso intensivo de la sentencia *insert* de SQL.

La forma general de la sentencia *insert* es la siguiente:

```
INSERT INTO nombre_tabla [(column1, column2,...)]  
VALUES(valor1, valor2,...);
```

También podemos insertar tuplas en una tabla a partir de otra tabla de la base de datos. En este caso, la forma general de la sentencia sería:

```
INSERT INTO nombre_tabla [(column1, column2,...)]  
(SELECT column1, column2,...  
FROM nombre_tabla2);
```

- **Ejemplo 2.1** Utiliza la sentencia INSERT para introducir valores en la tabla PRUEBA2. Para hacerlo, vamos a editar el fichero `introducir_datos.sql` cuyo contenido debe ser:

```
INSERT INTO prueba2 VALUES ('aa',1);
INSERT INTO prueba2 VALUES('Aa',2);
INSERT INTO prueba2 VALUES ('aa',1);
```

■

almacenar y salir del editor. Ejecutar

```
SQL> start introducir_datos;
```

Si la tupla es correcta (número y tipos de datos acordes, etc.) devolverá: 3 filas creadas. En caso contrario dará el correspondiente mensaje de error por cada tupla introducida errónea y se desestimará la inserción.

Observa que en el ejemplo 2.1 se pueden introducir tuplas repetidas, al no haber definido clave primaria en la tabla *prueba2*.

- **Ejemplo 2.2** Utiliza la sentencia INSERT para introducir valores en las tablas *plantilla* y *serjefe* del siguiente modo:

```
INSERT INTO plantilla (dni,nombre,estadocivil,fechaalta)
    VALUES ('12345678','Pepe','soltero', SYSDATE);
INSERT INTO plantilla (dni,nombre,estadocivil,fechaalta)
    VALUES ('87654321','Juan', 'casado', SYSDATE);
INSERT INTO serjefe VALUES ('87654321','12345678');
INSERT INTO plantilla (dni, estadocivil) VALUES ('11223344','soltero');
```

■

donde SYSDATE indica la fecha y hora del sistema.

2.1.2 Mostrar el contenido de una tabla

Una vez que hemos introducido los datos en las tablas de nuestro ejemplo, podemos ver el contenido de las mismas ejecutando la sentencia de consulta:

```
SQL> SELECT * FROM nombre-tabla;
```

La lista de atributos entre la cláusula SELECT y la cláusula FROM equivale en SQL a la operación de proyección de Álgebra Relacional. En este caso particular, el * equivale a proyectar sobre todos los atributos de las tablas relacionadas en al cláusula FROM. Para proyectar campos individuales, se debe ejecutar la siguiente sentencia:

```
SQL> SELECT campo1, campo2, .... FROM nombre-tabla;
```

Para conocer qué tablas tenemos creadas hasta este momento, podemos consultar una vista del catálogo del SGBD (5) denominada `user_tables`, en la forma que sigue:

```
SQL> SELECT table_name FROM user_tables;
```

En la próxima práctica ejercitaremos múltiples posibilidades de la sentencia SELECT.

Ejercicio 2.1 Ejecuta la sentencia SELECT para mostrar el contenido de las tablas PRUEBA2 y PLANTILLA. Intenta mostrar sólo algunos campos de las mismas.

2.1.3 Modificar el contenido de una tabla

Para modificar los datos de una tabla introducidos con anterioridad, hemos de utilizar la sentencia UPDATE, cuya forma general es la siguiente:

```
UPDATE nombre_tabla  
SET nombre_atributo = 'nuevovalor'  
[, nombre_atributo2 = 'nuevovalor2'...]  
[WHERE <condicion>];
```

Esta sentencia modifica la/s tupla/s que se ajustan al criterio especificado en la cláusula WHERE. Hay que destacar que [] indica opcionalidad. Así se puede modificar un atributo o más de un atributo simultáneamente. La sintaxis de la cláusula WHERE se basa en la expresión recogida en <condicion>. Dicha expresión es lógica y se construye a partir de los operadores de la Tabla 1.2. La cláusula WHERE puede aparecer en otras sentencias tales como DELETE, que veremos en la sección 2.1.4 y SELECT, que se estudiará con más detalle más adelante en este cuaderno de prácticas.

■ **Ejemplo 2.3** Ejecuta la sentencia UPDATE sobre la tabla *plantilla* y cambia el estado civil de Juan a *divorciado*.

```
SQL> UPDATE plantilla  
      SET estadocivil = 'divorciado'  
      WHERE nombre='Juan';
```

Ejercicio 2.2 Ejecuta la sentencia UPDATE sobre la tabla *plantilla* y cambia el nombre del trabajador con dni '12345678' a 'Luis'.

2.1.4 Borrado de tuplas

La instrucción DELETE se utiliza para eliminar tuplas de una tabla. Las tuplas que se eliminan son aquellas que hacen cierta la expresión <condicion>. Su sintaxis es la siguiente:

```
DELETE [FROM] nombre_tabla [WHERE <condicion>];
```

Donde [] indica opcionalidad, esto es la cláusula WHERE con su sintaxis habitual es opcional. condicion es cualquier expresión lógica similar utilizando los operadores de la Tabla 1.2.

■ **Ejemplo 2.4** Borra todas las tuplas de la tabla prueba2.

```
SQL> DELETE FROM prueba2;
```

La instrucción de borrado, sin cláusula WHERE, borra todas las tuplas de la tabla.

Ejercicio 2.3 Borra todas las tuplas de la tabla *plantilla*.

```
SQL> DELETE FROM plantilla;
```

En este caso da un mensaje de error (¿por qué?). Aunque sí podríamos borrar las tuplas

de la tabla *serjefe*.

```
SQL> DELETE FROM serjefe;
```

■

2.1.5 Particularidades del tipo de dato DATE

El tipo DATE sirve para almacenar información relativa a fechas. Está expresado en Juliano y su rango va del 1 de Enero de 4712 “Antes de Cristo” al 31 de Diciembre de 9999. Un determinado valor de este tipo almacena los segundos transcurridos desde el 1 de Enero de 4712 “Antes de Cristo”. Este formato de fecha permite, por tanto, disponer de un referencial continuo para el almacenamiento y la manipulación de fechas.

Oracle® permite sumar y restar valores constantes y otras fechas a los datos de tipo fecha. Para ello, la fecha se representa internamente como un único número (número de días); así, por ejemplo, SYSDATE + 1 es mañana , SYSDATE - 7 es *hace una semana* y SYSDATE + (10/1440) es *dentro de diez minutos*.

■ **Ejemplo 2.5** Ejecuta la sentencia UPDATE sobre la tabla *plantilla* y cambia la fecha de alta de Juan al día siguiente.

```
SQL> UPDATE plantilla
      SET fechaalta = fechaalta+1
    WHERE nombre='Juan';
```

■

Aunque los datos de fecha podrían representarse mediante los tipos VARCHAR y NUMBER, el tipo DATE ofrece, además, funciones específicas para su manejo que tienen en cuenta su semántica.

Introducción de fechas mediante la función TO_DATE

Con esta función se genera un valor de tipo date a partir del valor suministrado por la primera cadena pasada a la función usando como formato la segunda cadena proporcionada.

Un ejemplo de uso de la función TO_DATE es el siguiente:

```
SQL> INSERT INTO plantilla
      VALUES ('11223355','Miguel','casado',
      TO_DATE('22/10/2005','dd/mm/yyyy'),null);
```

En este ejemplo, puesto que el formato proporcionado es ‘dd/mm/yyyy’, conforme a la Tabla 2.1, los dos primeros dígitos (22) se interpretan como el día del mes, los dos dígitos siguientes (10) como el ordinal del mes (Octubre) y los últimos cuatro dígitos (2005) como el año expresado por cuatro dígitos, por tanto, se genera un dato de tipo date correspondiente al 22 de Octubre del año 2005.

Mostrar fechas mediante la función TO_CHAR

Para la recuperación de datos de tipo fecha en *un formato concreto*¹, la función que debe utilizarse es **TO_CHAR**, que transforma un valor de fecha (en su formato interno) a una cadena de caracteres imprimible según el formato fecha especificado. Los formatos de fecha disponibles se recogen en la Tabla 2.1.

¹En el caso de que se desee tener un formato distinto del que hay establecido por defecto.

-/..;"texto"	Separadores permitidos.
HH o HH12	Hora del día (1-12).
AM ó PM	Indicador para formato de 12 horas (p.e. 'HH:AM').
HH24	Hora del día (0-23).
MI	Minuto (0-59).
SS	Segundo (0-59).
D	Día de la semana entre 1 y 7.
DAY/day	Nombre del día (lunes, martes,...)
DD	Día del mes entre 1 y 31.
DDD	Día del año entre 1 y 365.
J	Número de día según calendario Juliano.
MM	Dos dígitos para el mes.
MON	Tres primeros caracteres del mes.
MONTH/month	Nombre completo del mes.
YYYY	Cuatro dígitos para el año.
Y,YYY	Cuatro dígitos con separador.
YEAR/year	Nombre del año.
YY	Ultimos dos dígitos del año.

Cuadro 2.1: Elementos de un formato de fecha

■ **Ejemplo 2.6** SQL> SELECT TO_CHAR(fechaalta,'dd-mon-yyyy') FROM plantilla;

Si se omite la función TO_CHAR en la sentencia SELECT, el formato aplicado será el que haya por defecto.

■ **Ejemplo 2.7** Ejecuta la sentencia SELECT sobre la tabla *plantilla* mostrando la fecha sin utilizar la función TO_CHAR y observa el resultado.

SQL> SELECT fechaalta FROM plantilla;

2.2 Inserción de tuplas en nuestra base de datos de ejemplo

Una vez creadas las tablas *proveedor*, *pieza*, *proyecto* y *ventas* vamos a introducir datos en ellas haciendo uso de la sentencia *INSERT* explicada en la sección 2.1.1. Se recomienda usar las tuplas que aparecen listadas a continuación para que las consultas posteriores tengan respuestas no vacías y se puedan cotejar los resultados.

Hay que tener en cuenta que, en SQL, se distinguen mayúsculas y minúsculas para las cadenas de caracteres, por lo que las cadenas 'PARIS', 'París' y 'Paris' se consideran diferentes.

A continuación se muestran los contenidos de las tablas proveedor, pieza y proyecto:

```
SQL> SELECT * FROM proveedor;
COD NOMPRO STATUS CIUDAD
----- -----
S1 Jose Fernandez 2 Madrid
S2 Manuel Vidal 1 Londres
S3 Luisa Gomez 3 Lisboa
S4 Pedro Sanchez 4 Paris
S5 Maria Reyes 5 Roma
```

SQL> SELECT * FROM pieza;

COD	NOMBRE	COLOR	PESO	CIUDAD
P1	Tuerca	Gris	2.5	Madrid
P2	Tornillo	Rojo	1.25	Paris
P3	Arandela	Blanco	3	Londres
P4	Clavo	Gris	5.5	Lisboa
P5	Alcayata	Blanco	10	Roma

SQL> SELECT * FROM proyecto;

COD	NOMBRE	CIUDAD
J1	Proyecto 1	Londres
J2	Proyecto 2	Londres
J3	Proyecto 3	Paris
J4	Proyecto 4	Roma

Para realizar la introducción de tuplas en la tabla ventas del usuario se provee de una tabla ya rellena en el servidor de Bd de la Escuela, se trata de la tabla **opc.ventas**, de la que se puede consultar su contenido con la sentencia: SQL> SELECT * FROM opc.ventas;.

A continuación se muestra su contenido.

COD	COD	COD	CANTIDAD	FECHA
S1	P1	J1	150	18/09/97
S1	P1	J2	100	06/05/96
S1	P1	J3	500	06/05/96
S1	P2	J1	200	22/07/95
S2	P2	J2	15	23/11/04
S4	P2	J3	1700	28/11/00
S1	P3	J1	800	22/07/95
S5	P3	J2	30	21/01/04
S1	P4	J1	10	22/07/95
S1	P4	J3	250	09/03/94
S2	P5	J2	300	23/11/04
S2	P2	J1	4500	15/08/04
S3	P1	J1	90	09/06/04
S3	P2	J1	190	12/04/02
S3	P5	J3	20	28/11/00
S4	P5	J1	15	12/04/02
S4	P3	J1	100	12/04/02
S4	P1	J3	1500	26/01/03
S1	P4	J4	290	09/03/94
S1	P2	J4	175	09/03/94
S5	P1	J4	400	21/01/04
S5	P3	J3	400	21/01/04

22 filas seleccionadas.

Obsérvese que esta tabla se llama *ventas*, al igual que la que hemos creado, pero pertenece a otro usuario, el usuario *opc*, que en esta ocasión ha dado permiso de consulta a cualquier usuario. Podemos llenar fácilmente nuestra tabla *ventas* a partir de la anterior atendiendo a la sintaxis dada en la sección 2.1.1. Si los esquemas de las tablas no son iguales, es decir, si no coinciden en número, orden y tipo de los atributos, se producirá un error. Por lo tanto, antes de copiar los datos de una a otra tabla, debemos de comprobar antes sus esquemas. Una vez comprobado, si coinciden, debemos ejecutar:

```
SQL> INSERT INTO ventas SELECT * FROM opc.ventas;
```

Tras esta última operación, es el momento de ejecutar el comando COMMIT para guardar de forma permanente todas las operaciones realizadas hasta el momento.

Ejercicio 2.4 A continuación vamos a tratar de insertar algunas tuplas nuevas en *ventas*. Comprueba que se introducen correctamente y, en caso contrario, razona por qué da error.

```
INSERT INTO ventas VALUES ('S3', 'P1', 'J1', 150, '24/12/05');  
INSERT INTO ventas (codpro, codpj) VALUES ('S4', 'J2');  
INSERT INTO ventas VALUES('S5','P3','J6',400,TO_DATE('25/12/00'));
```

Ejercicio 2.5 Actualizar la fecha del proveedor S5 al año 2005'

```
SQL> UPDATE ventas  
      SET fecha = TO_DATE(2005,'YYYY')  
    WHERE codpro='S5';
```

Ejercicio 2.6 Para mostrar la columna FECHA con un formato específico e imprimirla, utilizar la siguiente sentencia:

```
SQL> SELECT codpro,codpie,  
      TO_CHAR(fecha,'"Dia" day,dd/mm/yy') FROM ventas;
```

donde el texto que se quiere incluir como parte de la fecha debe ir entre comillas dobles. ■

2.3 Antes de salir...

Por último, antes de terminar la sesión, se pide dejar únicamente las tablas necesarias y las tuplas definitivas para las próximas sesiones. Para ello:

1. Comprobar las tablas de usuario
2. Borrar el resto de tablas excepto las tablas *proveedor*, *pieza*, *proyecto* y *ventas*
3. Si has ejecutado alguno de los ejercicios 2.4 o 2.5 ejecuta el comando ROLLBACK que restituye la base de datos al estado anterior. Esto es, deshace los últimos cambios realizados desde el último commit.
4. Ejecutar COMMIT antes de abandonar la sesión.

Rellenar todas las tablas de la base de datos de baloncesto con algunas tuplas.

2.4 Ejercicios adicionales

2.4.1 Inserción de tuplas en la Base de Datos sobre baloncesto

Ejercicio 2.7 Preparar un archivo para la introducción de datos en las tablas Equipos, Jugadores, Encuentros y Faltas de la base de datos sobre baloncestos creada en la sección 1.8, para que nos permitan realizar consultas con resultados significativos, conforme a los siguientes criterios:

- Que se inserten 4 equipos con 5 jugadores en cada uno.
- Que se inserten 10 encuentros (no se ha terminado la liga).
- Que se inserten los resultados esos encuentros dejando un único equipo invicto.

Capítulo 3

Realización de consultas a una base de datos

Este capítulo del guión se presenta un breve recorrido por los elementos más comunes del uso de la sentencia SELECT, junto a serie de *ejemplos* y *ejercicios sencillos* que pueden servir al alumno como referencia en una primera *etapa de aprendizaje*. Este contenido se completa con el *estudio más amplio* de dicha sentencia desarrollado en las explicaciones teórico/prácticas de los profesores y en la bibliografía de la asignatura.

Los ejercicios de esta unidad se pueden realizar mediante SQL*Plus o mediante la “Hoja de Trabajo” de SQL Developer (ver Apéndice A.3). Esta herramienta también dispone de un editor visual básico para la edición de consultas (la pestaña “Generador de consultas”), aunque no se recomienda su uso.

Con el objeto de proporcionar una perspectiva conjunta de SQL, del Álgebra Relacional y del Cálculo Relacional Orientado a Tuplas, para aquellos tipos de consultas susceptibles de expresarse mediante expresiones del Álgebra y/o del Cálculo, se proporciona la expresión en dichos lenguajes y, además, para poder ejercitarse con dichas consultas, se adjunta la misma expresión en términos de la sintaxis empleada por la herramienta WinRDBI, cuyo uso se describe en el Apéndice B.

Esta práctica está confeccionada para procurar una guía para el aprendizaje de esta compleja sentencia, y de su relación con el Álgebra y Cálculo Relacionales, sin embargo es preciso aclarar lo siguiente:

- El seguimiento exhaustivo de esta guía no garantiza la superación de esta parte de las prácticas de la asignatura.
- Tras asimilar los conceptos de SQL, de Álgebra y Cálculo Relacionales que aparecen en esta práctica, se recomienda al alumno/a la realización de los ejercicios adicionales que se hallan en la parte final de esta unidad del cuaderno de prácticas, tanto en SQL como, en aquellos ejercicios que lo permitan, en AR, COT y WinRDBI.

3.1 La sentencia de consulta SELECT

La sentencia SELECT permite consultar las tablas seleccionando datos en tuplas y columnas de una o varias tablas. La sintaxis general de la sentencia con sus múltiples cláusulas se detalla a continuación:

```

SELECT [ DISTINCT | ALL ]
    expresion [alias_columna_expresion]
    {,expresion [alias_columna_expresion]}
FROM [esquema.]tabla|vista [alias_tabla_vista]
[WHERE <condicion>]
[GROUP BY expresion {,expresion}]
[HAVING <condicion>]
[ {UNION | UNION ALL | INTERSECT | MINUS} <SELECT instruccion>]
[ORDER BY {expresion} [ASC | DESC]]

```

Seguidamente iremos viendo con detalle algunas de sus principales cláusulas.

3.2 La consulta en SQL y su relación con los operadores del AR

En este apartado ejercitaremos los siguientes componentes de la sentencia SELECT:

```

SELECT [ DISTINCT | ALL ]
    <expresion> [alias_columna_expresion]
    {,<expresion> [alias_columna_expresion]}
FROM [esquema.]tabla|vista [alias_tabla_vista]
[ WHERE <condicion>]

```

3.2.1 La proyección AR en SQL

La proyección del Álgebra Relacional se expresa en la sentencia SELECT mediante la lista de campos, denominados “select list” que se relacionan entre la cláusula SELECT y la cláusula FROM. Se utiliza el * para determinar que se proyecte sobre todos los campos de las tablas listadas en la cláusula FROM.

■ **Ejemplo 3.1** Muestra las ciudades donde hay un proyecto. ■

AR: $\pi_{ciudad}(Proyecto)$

WinRDBI (AR): qProjection := project ciudad (proyecto);

WinRDBI (CRT): qQuery := { J.ciudad | proyecto(J) };

SQL> SELECT ciudad FROM proyecto;

Ejercicio 3.1 Comprueba el resultado de la proyección. ¿Es éste conforme a lo que se obtiene en el AR? ■

Solución: Uso de la cláusula DISTINCT.

SQL> SELECT DISTINCT ciudad FROM proyecto;

■ **Ejemplo 3.2** Muestra la información disponible acerca de los proveedores. ■

SQL> SELECT * FROM proveedor; * muestra el esquema completo, o bien proyectando uno a uno los atributos

SQL> SELECT codpro, nompro, status, ciudad FROM proveedor;

Ejercicio 3.2 Muestra los suministros realizados (tan solo los códigos de los componentes de una venta). ¿Es necesario utilizar DISTINCT? ■

3.2.2 La selección AR en SQL

Para realizar la selección Algebráica σ en SQL se emplea la cláusula WHERE seguida de <condicion>, aunque siempre será necesario especificar la cláusula SELECT de la instrucción de consulta. <condicion> es una expresión booleana que implica cualquiera de los atributos de la tabla que figura en la cláusula FROM de la instrucción. Los operadores que pueden intervenir en esta expresión son cualesquiera de los ya presentados en la Tabla 1.2 y algunos adicionales que mostramos en esta sección.

■ **Ejemplo 3.3** Muestra los códigos de los proveedores que suministran al proyecto 'J1'. ■

AR : $\pi_{codpro}(\sigma_{codpj='J1'}(Ventas))$

WinRDBI(AR) : mQuery := project codpro (select codpj='J1' (ventas));
 WinRDBI(CRT) : qQuery := { V.codpro | ventas(V) and V.codpj='J1' };

SQL> SELECT codpro FROM ventas WHERE codpj='J1';

Ejercicio 3.3 Muestra las piezas de Madrid que son grises o rojas. ■

Ejercicio 3.4 Encontrar todos los suministros cuya cantidad está entre 200 y 300, ambos inclusive. ■

Operadores adicionales específicos de SQL

■ **El operador like y los caracteres comodín _ y %** El operador like se emplea para comparar cadenas de caracteres mediante el uso de patrones. Cuando se emplea el carácter comodín %, éste se sustituye por cualquier cadena de 0 ó más caracteres:

■ **Ejemplo 3.4** Mostrar los proveedores cuyo nombre de ciudad empieza por 'L'. ■

SQL> SELECT nompro, ciudad FROM proveedor WHERE ciudad LIKE 'L%';
 El carácter comodín _ sustituye un sólo carácter.

Ejercicio 3.5 Mostrar las piezas que contengan la palabra tornillo con la t en mayúscula o en minúscula. ■

■ Uso de operadores aritméticos y funciones numéricas.

■ **Ejemplo 3.5** Describe la cantidad de cada venta expresada en docenas, docenas redondeadas y truncadas al tercer decimal y aproximadas por el entero inferior y por el entero superior, sólo de las ventas cuyo número de piezas es mayor de diez docenas. ■

SQL> SELECT cantidad/12, round(cantidad/12,3), trunc(cantidad/12,3),
 floor(cantidad/12), ceil(cantidad/12)
 FROM ventas WHERE (cantidad/12)>10;

■ **Comparación con el valor nulo.** El operador IS [NOT] NULL

■ **Ejemplo 3.6** Encontrar los proveedores que tienen su status registrado en la base de datos. ■

SQL> SELECT codpro, nompro FROM proveedor WHERE status IS NOT NULL;

3.2.3 Consultas sobre el catálogo

Ya podemos consultar con más detalle algunas de las vistas del catálogo.

```
SQL> SELECT table_name
   FROM USER_TABLES;
```

■ **Ejemplo 3.7** Mostrar la información de todas las tablas denominadas *ventas* a las que tienes acceso. ■

```
SQL> SELECT table_name
   FROM ALL_TABLES
  WHERE TABLE_NAME like '%ventas';
```

Ejercicio 3.6 Comprueba que no devuelve ninguna. Pero SI que hay!!!. Prueba a usar la función `upper()` comparando con 'VENTAS' o la función `lower()` comparando con 'ventas'

3.2.4 Operadores AR sobre conjuntos en SQL

```
<SELECT instrucion>
    UNION | UNION ALL | INTERSECT | MINUS
<SELECT instrucion>
```

Estos operadores tienen una restricción similar a sus correspondientes del AR, para poder llevarse a cabo: los esquemas de las tablas resultantes de cada sentencia SELECT han de ser iguales en tipo, esto es, los atributos no tienen porqué llamarse igual, aunque sí han de coincidir en número, posición en el “select list” y tipo. Tras la operación, el esquema del resultado coincide con el esquema del primer operando.

■ **Ejemplo 3.8** Ciudades donde viven proveedores con status mayor de 2 en las que no se fabrica la pieza 'P1'. ■

$$\text{AR} : \pi_{ciudad}(\sigma_{status>2}(proveedor)) - \pi_{ciudad}(\sigma_{codpie='P1'}(pieza))$$

```
WinRDBI(AR): mQuery := (project ciudad (select status>2 (proveedor)))
                      difference
                      (project ciudad (select codpie='P1' (pieza)));
```

```
WinRDBI(CRT): sta_2 := {P.ciudad | proveedor(P) and P.status >2};
               pieP1 := {Pi.ciudad | pieza(Pi) and Pi.codpie='P1' };
               qresult := { T | sta_2(T) and not pieP1(T) };
```

```
% Alternativa en una sola sentencia
qresult := {P.ciudad | proveedor(P) and P.status >2
            and not (exists T)
            (pieza(T) and T.ciudad=P.ciudad and T.codpie ='P1' )};
```

```
SQL> (SELECT DISTINCT ciudad FROM proveedor WHERE status>2)
      MINUS
      (SELECT DISTINCT ciudad FROM pieza WHERE codpie='P1');
```

Nótese que los operadores UNION, MINUS e INTERSECT implementan en SQL las operaciones \cup , $-$, \cap del AR, respectivamente y que, por tanto, consideran los argumentos como relaciones (sin tuplas repetidas) y devuelven el resultado como una relación (sin tuplas repetidas). Por consiguiente, la sentencia SQL que resuelve el ejercicio anterior podría prescindir de las cláusulas DISTINCT. Sin embargo el operador UNION ALL devuelve todas las tuplas incluidas en las tablas argumento, sin eliminar tuplas duplicadas.

Ejercicio 3.7 Resolver la consulta del ejemplo 3.8 utilizando el operador \cap .

Ejercicio 3.8 Encontrar los códigos de aquellos proyectos a los que sólo abastece 'S1'.

Ejercicio 3.9 Mostrar todas las ciudades de la base de datos. Utilizar UNION

Ejercicio 3.10 Mostrar todas las ciudades de la base de datos. Utilizar UNION ALL

3.2.5 El producto cartesiano AR en SQL

En la cláusula FROM de una sentencia de consulta puede aparecer una lista de tablas (o de subconsultas) en lugar de una sola. En este caso, el sistema realiza el producto cartesiano de todas las tablas (o subconsultas) incluidas en dicha lista para, posteriormente, seleccionar aquellas tuplas que hacen verdad la condición de la cláusula WHERE (en el caso de que se haya establecido) mostrándolas como resultado de ese producto cartesiano.

Ejercicio 3.11 Comprueba cuántas tuplas resultan del producto cartesiano aplicado a ventas y proveedor

■ **Ejemplo 3.9** Muestra las posibles ternas (codpro,codpie,codpj) tal que, todos los implicados sean de la misma ciudad.

AR : $\pi_{codpro, codpie, codpj}(\sigma_{Proveedor.ciudad=Proyecto.ciudad \wedge Proyecto.ciudad=Pieza.ciudad}((Proveedor \times Proyecto) \times Pieza))$

WinRDBI(AR): pQuery := project codpro, codpie, codpj
(proveedor njoin proyecto) njoin pieza;

WinRDBI(CRT): qresult := { P.codpro, Pi.codpie, Pj.codpj |
proveedor (P) and pieza(Pi) and proyecto(Pj)
and P.ciudad=Pi.ciudad and Pi.ciudad=Pj.ciudad };

SQL> SELECT codpro, codpie, codpj
FROM proveedor, proyecto, pieza
WHERE Proveedor.ciudad=Proyecto.ciudad
AND Proyecto.ciudad=Pieza.ciudad;

■ **Ejemplo 3.10** Mostrar las ternas (codpro,codpie,codpj) tal que todos los implicados son de Londres.

AR : $\pi_{codpro, codpie, codpj}((\sigma_{Proveedor.ciudad='Londres'}(Proveedor) \times \sigma_{Proyecto.ciudad='Londres'}(Proyecto)) \times \sigma_{Pieza.ciudad='Londres'}(Pieza))$

```

WinRDBI(AR): pQuery :=
(
  (project codpro (select ciudad='Londres' (proveedor))) product
  (project codpj (select ciudad='Londres' (proyecto)))
) product
  (project codpie (select ciudad='Londres' (pieza)));

```



```

WinRDBI(CRT): qresult := { P.codpro, Pi.codpie, Pj.codpj |
  proveedor (P) and pieza(Pi) and proyecto(Pj)
  and P.ciudad='Londres' and Pi.ciudad='Londres'
  and Pj.ciudad='Londres'};
```



```

SQL> SELECT codpro,codpie,codpj
  FROM proveedor, proyecto, pieza
 WHERE Proveedor.ciudad='Londres' AND Proyecto.ciudad='Londres'
   AND Pieza.ciudad='Londres';
```

Ejercicio 3.12 Mostrar las ternas que son de la misma ciudad pero que hayan realizado alguna venta.

3.2.6 El renombramiento o alias en SQL

El empleo de alias puede ser útil para abreviar texto cuando es necesario prefijar atributos para eliminar ambigüedades, sin embargo, es estrictamente necesario cuando se hace un producto cartesiano de una tabla consigo misma o cuando hay que hacer referencia a los campos de una consulta incluida en la cláusula FROM. Los alias se definen asociándolos a aquellas tablas o consultas presentes en la cláusula FROM que se deseen redefinir.

■ **Ejemplo 3.11** Muestra las posibles ternas (codpro,codpie,codpj) tal que todos los implicados sean de la misma ciudad. Revisitando la consulta del ejemplo 3.9.

$$AR : \pi_{codpro,codpie,codpj}(\sigma_{S.ciudad=Y.ciudad \wedge Y.ciudad=P.ciudad}(\\ \rho_S(Proveedor) \times \rho_Y(Proyecto) \times \rho_P(Pieza)))$$

```

WinRDBI(AR):
% Creación de tres relaciones intermedias con atributo ciudad renombrado
cproveedor(codpro,sciudad) := project codpro,ciudad (proveedor);
cproyecto(codpj,yciudad) := project codpj,ciudad (proyecto);
cpieza(codpie,pciudad) := project codpie,ciudad (pieza);
% Consulta
cQuery := project codpro,codpj,codpie (
  select sciudad = yciudad and yciudad= pciudad
  ((cproveedor product cproyecto) product cpieza)
);
```

```

SQL> SELECT codpro, codpie, codpj
  FROM proveedor S, proyecto Y, pieza P
 WHERE S.ciudad=Y.ciudad and Y.ciudad=P.ciudad;
```

Ejercicio 3.13 Encontrar parejas de proveedores que no viven en la misma ciudad.

Ejercicio 3.14 Encuentra las piezas con máximo peso.

3.2.7 La equi-reunión y la reunión natural AR en SQL

Llegado este punto, disponemos de todos los elementos SQL para expresar el operador equi-reunión y la reunión natural (\bowtie). Para la reunión natural se usa la cláusula NATURAL JOIN dentro de la cláusula FROM entre las tablas o subconsultas participantes. EL SGBD aplica la reunión natural sobre aquellos campos que se llamen de igual forma en las tablas o subconsultas intervenientes, si no coincidieran en tipo, devolvería error.

■ **Ejemplo 3.12** Mostrar los nombres de proveedores y cantidad de aquellos que han realizado alguna venta en cantidad superior a 800 unidades.

AR : $\pi_{nompro,cantidad}(proveedor \bowtie \sigma_{cantidad>800}(ventas))$

```
WinRDBI(AR): ventas800 := project nompro,cantidad
              (proveedor njoin (select cantidad>800 (ventas)));
```

```
WinRDBI(CRT): qresult := {P.nompro, V1.cantidad | proveedor (P) and ventas(V1)
                           and (exists V)(ventas(V) and V.codpro=P.codpro
                           and V1.cantidad=V.cantidad and V.cantidad > 800) };
```

```
SQL> SELECT nompro, cantidad
      FROM proveedor NATURAL JOIN (SELECT * FROM ventas WHERE cantidad>800);
```

```
-- expresión equivalente sin usar el operador natural join. Nótese que es
-- preciso el uso de alias sobre la tabla y la subconsulta involucradas en
-- el producto cartesiano
```

```
SQL> SELECT nompro, cantidad
      FROM proveedor s, (SELECT * FROM ventas WHERE cantidad>800) v
      WHERE s.codpro= v.codpro;
```

Observe el resultado que se obtiene de la reunión natural cuando se proyecta sobre todos los atributos. Si se quiere reunir en base a campos que no tienen el mismo nombre, se pueden usar dos alternativas: producto cartesiano junto a condición de reunión en la cláusula WHERE o la equi-reunión expresada mediante cláusula JOIN ... ON en la forma que se indica a continuación:

```
SQL> SELECT nompro, cantidad
      FROM proveedor s JOIN (SELECT * FROM ventas WHERE cantidad>800) v
      ON (s.codpro=v.codpro);
```

Ejercicio 3.15 Mostrar las piezas vendidas por los proveedores de Madrid.

Ejercicio 3.16 Encuentra la ciudad y los códigos de las piezas suministradas a cualquier proyecto por un proveedor que está en la misma ciudad donde está el proyecto.

3.3 Ordenación de resultados

Ya sabemos que en el modelo relacional no existe orden entre las tuplas ni entre los atributos, aunque sí es posible indicar al SGBD que ordene los resultados según algún criterio, mediante la cláusula ORDER BY. Caso de emplearse ésta, el orden por defecto es creciente (ASC).

```
SELECT [DISTINCT | ALL] expresion [alias_columna_expresion]
      {,expresion [alias_columna_expresion]}
  FROM [esquema.]tabla|vista [alias_tabla_vista]
  [WHERE <condicion>]
  ORDER BY expresion [ASC | DESC]{,expresion [ASC | DESC]}
```

- **Ejemplo 3.13** Encontrar los nombres de proveedores ordenados alfabéticamente.

```
SQL> SELECT nompro
      FROM proveedor
      ORDER BY nompro;
```

Ejercicio 3.17 Comprobar la salida de la consulta anterior sin la cláusula ORDER BY.

Ejercicio 3.18 Listar las ventas ordenadas por cantidad, si algunas ventas coinciden en la cantidad se ordenan en función de la fecha de manera descendente.

3.4 Subconsultas en SQL

Existen en SQL distintos operadores que permiten operar sobre el resultado de una consulta, esto se hace incorporando una subconsulta en la cláusula WHERE de la consulta principal. La razón de proceder de esta forma es que se fragmenta la consulta original en varias consultas más sencillas, evitando en muchas ocasiones numerosas reuniones.

```
SELECT <expresion>
  FROM tabla
 WHERE <expresion> OPERADOR <SELECT instrucion>
```

Dónde OPERADOR es cualquiera de los que se presentan en esta sección y, la cláusula SELECT a la derecha de OPERADOR puede contener a su vez otra subconsulta, que puede, a su vez, anidar un determinado numero de subconsultas. El máximo número de anidamientos permitido depende de cada sistema. Para su resolución el sistema procede resolviendo la subconsulta anidada a un mayor nivel de profundidad y sigue resolviendo todas las instrucciones SELECT en orden inverso de anidamiento.

3.4.1 IN, el operador de pertenencia

Un uso muy frecuente del operador de pertenencia a un conjunto IN consiste en obtener mediante una subconsulta los elementos de dicho conjunto.

- **Ejemplo 3.14** Encontrar las piezas suministradas por proveedores de Londres. (Sin usar el operador de reunión.)

```
SQL> SELECT codpie
      FROM ventas
     WHERE codpro IN
          (SELECT codpro FROM proveedor WHERE ciudad = 'Londres');
```

Ejercicio 3.19 Mostrar las piezas vendidas por los proveedores de Madrid. (Fragmentando la consulta con ayuda del operador IN.) Compara la solución con la del ejercicio 3.15. ■

Ejercicio 3.20 Encuentra los proyectos que están en una ciudad donde se fabrica alguna pieza. ■

Ejercicio 3.21 Encuentra los códigos de aquellos proyectos que no utilizan ninguna pieza roja que esté suministrada por un proveedor de Londres. ■

3.4.2 EXISTS, el operador de comprobación de existencia

Este operador devuelve verdadero cuando existe alguna tupla en la relación sobre la que se aplica. El operador EXISTS puede interpretarse también como de comprobación de conjunto no vacío.

■ **Ejemplo 3.15** Encontrar los proveedores que suministran la pieza 'P1'. ■

```
SQL> SELECT codpro
      FROM proveedor
     WHERE EXISTS (SELECT * FROM ventas
                   WHERE ventas.codpro = proveedor.codpro
                     AND ventas.codpie='P1');
```

3.4.3 Otros operadores, los comparadores sobre conjuntos

Cualquiera de los operadores relacionales $<$ | \leq | $>$ | \geq | \neq junto con alguno de los cuantificadores [ANY|ALL] pueden servir para conectar una subconsulta con la consulta principal.

■ **Ejemplo 3.16** Muestra el código de los proveedores cuyo estatus sea igual al del proveedor 'S3'. ■

```
SQL> SELECT codpro
      FROM proveedor
     WHERE status = (SELECT status FROM proveedor WHERE codpro='S3');
```

■ **Ejemplo 3.17** Muestra el código de las piezas cuyo peso es mayor que el peso de alguna pieza 'tornillo'. ■

```
SQL> SELECT codpie
      FROM pieza
     WHERE peso > ANY
           (SELECT peso FROM pieza WHERE nompie LIKE 'Tornillo%');
```

Ejercicio 3.22 Muestra el código de las piezas cuyo peso es mayor que el peso de cualquier 'tornillo'. ■

Ejercicio 3.23 Encuentra las piezas con peso máximo. Compara esta solución con la obtenida en el ejercicio 3.14 ■

3.5 La división AR en SQL

Ya estamos en condiciones de trasladar el operador \div del AR a SQL. Para ello procederemos a hacerlo de tres formas diferentes: utilizando una aproximación basada en el Álgebra Relacional, otra basada en el Cálculo Relacional y otra usando un enfoque mixto.

■ **Ejemplo 3.18** Mostrar el código de los proveedores que suministran todas las piezas. ■

$$\text{AR} : \pi_{\text{codpro}, \text{codpie}}(\text{ventas}) \div \pi_{\text{codpie}}(\text{pieza})$$

3.5.1 Aproximación usando expresión equivalente en AR

La división relacional no es un operador primitivo del Álgebra Relacional ya que se puede expresar en combinación de los otros operadores primitivos de la siguiente forma:

$$\text{Relacion1} \div \text{Relacion2} = \pi_A(\text{Relacion1}) - \pi_A((\pi_A(\text{Relacion1}) \times \text{Relacion2}) - \text{Relacion1}) \\ \text{siendo } A = \{\text{AtributosRelacion1}\} - \{\text{AtributosRelacion2}\}$$

Particularizando a la consulta propuesta la expresión algebraica quedaría así:

$$\pi_{\text{codpro}}(\text{ventas}) - \pi_{\text{codpro}}((\pi_{\text{codpro}}(\text{ventas}) \times \pi_{\text{codpie}}(\text{pieza})) - \pi_{\text{codpro}, \text{codpie}}(\text{ventas}))$$

La herramienta WinRDBI no proporciona, de forma intencionada, el operador división, por lo que hay que acudir a la anterior expresión para elaborar la consulta solicitada, que queda de la siguiente forma:

WinRDBI(AR) :

```
dividendo := project codpro, codpie (ventas);
divisor := project codpie (pieza);
codproventas := project codpro (ventas);
pDivision := codproventas
            difference (
                project codpro
                ((codproventas product divisor) difference dividendo)
            );
```

Usando la expresión anterior podríamos expresar la consulta en SQL de la siguiente forma:

```
SQL> (SELECT codpro FROM ventas)
      MINUS
      (SELECT codpro
       FROM (
           (SELECT v.codpro, p.codpie FROM
            (SELECT DISTINCT codpro FROM ventas) v,
            (SELECT codpie FROM pieza) p
           )
           MINUS
           (SELECT codpro, codpie FROM ventas)
       )
    );
```

3.5.2 Aproximación basada en el Cálculo Relacional

De forma intuitiva, la manera de proceder sería:

```
Seleccionar proveedores tal que
no exista (una pieza (para la que no exista un suministro de ese proveedor))
WinRDBI(CRT): qresult := { P.nompro | proveedor (P) and not (exists Pi) (
    pieza(Pi) and not (exists V) (
        ventas(V) and V.codpro=P.codpro and V.codpie=Pi.codpie
    )
) };
```

```
SQL> SELECT codpro
  FROM proveedor
 WHERE NOT EXISTS (
    (SELECT * FROM pieza
     WHERE NOT EXISTS (SELECT * FROM ventas
                        WHERE pieza.codpie= ventas.codpie
                          AND proveedor.codpro=ventas.codpro)));
```

3.5.3 Aproximación mixta usando NOT EXISTS y la diferencia relacional

De forma intuitiva, la manera de proceder sería:

```
Seleccionar proveedores tal que
  (el conjunto de todas las piezas)
    menos
  (el conjunto de las piezas suministradas por ese proveedor)
    sea vacío.
```

Es decir, no existe ninguna tupla en la diferencia de esos conjuntos.

```
SQL> SELECT codpro
  FROM proveedor
 WHERE NOT EXISTS (
    (SELECT DISTINCT codpie FROM pieza)
      MINUS
    (SELECT DISTINCT codpie FROM ventas WHERE proveedor.codpro=ventas.codpro)
 );
```

Ejercicio 3.24 Encontrar los códigos de las piezas suministradas a todos los proyectos localizados en Londres.

Ejercicio 3.25 Encontrar aquellos proveedores que envían piezas procedentes de todas las ciudades donde hay un proyecto.

3.6 Funciones de agregación

En ocasiones puede ser interesante resumir la información relativa a un determinado conjunto de tuplas. SQL incorpora una serie de funciones estándares. A continuación se presentan algunas de ellas:

`SUM()`, `MIN()`, `MAX()`, `AVG()`, `COUNT()`, `STDDEV()` . . . , que respectivamente calculan: la suma, el mínimo, el máximo, la media, el cardinal y la desviación típica sobre el conjunto de valores pasados como argumento de la función. Cuando se usa la cláusula `DISTINCT` como argumento de la función, sólo tendrá en cuenta los valores distintos para realizar la correspondiente agregación.

Vamos a ver ejemplos de utilización de estas funciones en la cláusula `SELECT`.

■ **Ejemplo 3.19** Mostrar el máximo, el mínimo y el total de unidades vendidas.

```
SQL> SELECT MAX(cantidad), MIN(cantidad), SUM(cantidad) FROM ventas;
```

Comparar y justificar el resultado con el obtenido de:

```
SQL> SELECT MAX(DISTINCT cantidad), MIN(DISTINCT cantidad),
      SUM(DISTINCT cantidad) FROM ventas;
```

■ **Ejercicio 3.26** Encontrar el número de envíos con más de 1000 unidades.

■ **Ejercicio 3.27** Mostrar el máximo peso.

■ **Ejercicio 3.28** Mostrar el código de la pieza de máximo peso. Compara esta solución con las correspondientes de los ejercicios 3.14 y 3.23.

■ **Ejercicio 3.29** Comprueba si la siguiente sentencia resuelve el ejercicio anterior.

```
SQL> SELECT codpie, MAX(peso)
      FROM pieza;
```

■ **Ejercicio 3.30** Muestra los códigos de proveedores que han hecho más de 3 envíos diferentes.

3.6.1 Formando grupos

Hasta aquí, las funciones de agregación se han aplicado sobre todas las tuplas que devuelve una consulta. Sin embargo es posible realizar un particionado sobre el conjunto de las tuplas usando la cláusula `GROUP BY`. Mediante esta cláusula se indica el atributo o conjunto de atributos por cuyos valores se quiere agrupar las tuplas y proceder así a aplicar las funciones de agregación a cada uno de los grupos.

```
SELECT [ DISTINCT | ALL ]
      expresion [alias_columna_expcion]
      {,expresion [alias_columna_expcion]}
  FROM [esquema.]tabla|vista [alias_tabla_vista]
  [WHERE <condicion>]
  GROUP BY expresion {,expresion}
```

■ **Ejemplo 3.20** Para cada proveedor, mostrar la cantidad de ventas realizadas y el máximo de unidades suministrado en una venta.

```
SQL> SELECT codpro, COUNT(*), MAX(cantidad)
      FROM ventas
      GROUP BY (codpro);
```

codpro	codp	codpj	cantidad
S1	P1	J1	150
S1	P1	J2	100
S1	P1	J3	500
S1	P2	...	
S1	
S2	P2	J2	15
S2	P5	J2	300
S2	
S3	P1	J1	90
S3	P2	J1	190
S3	
S4	P2	J3	1700
S4	P5	J1	10
S4	
S5	P3	J2	30
S5	P1	J4	400
S5	

codpro	count	max
S1	9	800
S2	3	4500
S3	3	190
S4	4	1700
S5	3	400

Figura 3.1: Instancia de la tabla ventas y el resultado de la consulta del ejemplo 3.20.

Ejercicio 3.31 Mostrar la media de las cantidades vendidas por cada código de pieza junto con su nombre. ■

Ejercicio 3.32 Encontrar la cantidad media de ventas de la pieza 'P1' realizadas por cada proveedor. ■

Ejercicio 3.33 Encontrar la cantidad total de cada pieza enviada a cada proyecto. ■

3.6.2 Seleccionando grupos

Hasta ahora cuando se definían los grupos, todos grupos formaban parte del resultado. Sin embargo, es posible establecer condiciones sobre los grupos mediante la cláusula HAVING junto con una <condicion>, de forma parecida a como se hace con la cláusula WHERE sobre tuplas. Esto es, la condición se aplica **sobre los grupos** y determina que grupos aparecen como resultado de la consulta. La <condicion> a satisfacer por los grupos se elabora utilizando alguna de las funciones de agregación vistas y componiendo una expresión booleana con los operadores ya conocidos.

```
SELECT [ DISTINCT | ALL ]
    expresion [alias_columna_expresion]
    {,expresion [alias_columna_expresion]}
FROM [esquema.]tabla|vista [alias_tabla_vista]
[WHERE <condicion>]
GROUP BY expresion {,expresion}
[HAVING <condicion>]
```

■ **Ejemplo 3.21** Hallar la cantidad media de ventas realizadas por cada proveedor, indicando solamente los códigos de proveedores que han hecho más de 3 ventas. ■

```
SQL> SELECT codpro, AVG(cantidad)
  FROM ventas
 GROUP BY (codpro)
 HAVING COUNT(*) >3;
```

Para integrar los nuevos conceptos de selección de grupos con los antiguos de selección de tuplas vamos a plantear una consulta donde se combinan las cláusulas WHERE y having.

■ **Ejemplo 3.22** Mostrar la media de unidades vendidas de la pieza 'P1' realizadas por cada proveedor, indicando solamente la información de aquellos proveedores que han hecho entre 2 y 10 ventas. ■

```
SQL> Select codpro, codpie, AVG(cantidad)
   FROM ventas
  WHERE codpie='P1'
 GROUP BY (codpro, codpie) HAVING COUNT(*) BETWEEN 2 AND 10
```

■ **Ejemplo 3.23** Encuentra los nombres de proyectos y cantidad media de piezas que recibe por proveedor. ■

```
SQL> SELECT v.codpro, v.codpj, j.nompj, AVG(v.cantidad)
   FROM ventas v, proyecto j
  WHERE v.codpj=j.codpj
 GROUP BY (v.codpj, j.nompj,v.codpro);
```

Ejercicio 3.34 Comprueba si es correcta la solución anterior. ■

Ejercicio 3.35 Mostrar los nombres de proveedores tales que el total de sus ventas superen la cantidad de 1000 unidades. ■

3.6.3 Subconsultas en la cláusula HAVING

Ya hemos visto cómo una consulta compleja se puede fragmentar en varias subconsultas anidadas introduciendo una subconsulta en la cláusula WHERE y combinando los resultados. No es éste el único lugar dónde se pueden utilizar subconsultas, también puede hacerse en la cláusula HAVING.

■ **Ejemplo 3.24** Mostrar el proveedor que más ha vendido en total. ■

```
SQL> SELECT codpro, sum(cantidad)
   FROM ventas
  GROUP BY codpro
 HAVING SUM(cantidad) = (SELECT MAX(SUM(V1.cantidad))
                           FROM ventas V1
                          GROUP BY V1.codpro);
```

Ejercicio 3.36 Mostrar la pieza que más se ha vendido en total. ■

3.7 Consultas adicionales

3.7.1 Consultas con el tipo DATE

Uso de fechas en la cláusula SELECT.

Utilizando la función de conversión `to_char()` para la conversión de una fecha a una cadena en un formato determinado por los parámetros que se detallaron en la tabla 2.1.

■ **Ejemplo 3.25** Lista las fechas de las ventas en un formato día, mes y año con 4 dígitos. ■

```
SQL> SELECT TO_CHAR(fecha, 'DD-MM-YYYY')  
      FROM ventas;
```

Uso de fechas en la cláusula WHERE.

Haciendo uso de la función de conversión `to_date()` para hacer comparaciones entre fechas en formato interno.

■ Ejemplo 3.26

Encontrar las ventas realizadas entre el 1 de enero de 2002 y el 31 de diciembre de 2004. ■

```
SQL> SELECT * FROM ventas  
      WHERE fecha BETWEEN TO_DATE('01/01/2002', 'DD/MM/YYYY') AND  
            TO_DATE('31/12/2004', 'DD/MM/YYYY');
```

Ejercicio 3.37 Comprueba que no funciona correctamente si las comparaciones de fechas se hacen con cadenas. ■

■ Ejemplo 3.27 Mostrar las piezas que nunca fueron suministradas después del año 2001. ■

```
SQL> (SELECT DISTINCT codpie FROM pieza)  
      MINUS  
      (SELECT DISTINCT codpie FROM ventas  
      WHERE TO_NUMBER(TO_CHAR(fecha, 'YYYY')) > 2001);
```

ó

```
SELECT p.codpie FROM pieza p WHERE NOT EXISTS  
(SELECT * FROM ventas v WHERE TO_NUMBER(TO_CHAR(v.fecha, 'YYYY')) > 2001  
AND v.codpie=p.codpie);
```

Uso de fechas en la cláusula GROUP BY.

■ Ejemplo 3.28 Agrupar los suministros de la tabla de ventas por años y sumar las cantidades totales anuales. ■

```
SQL> SELECT TO_CHAR(fecha, 'YYYY'), SUM(cantidad)  
      FROM ventas  
      GROUP BY TO_CHAR(fecha, 'YYYY');
```

Ejercicio 3.38 Encontrar la cantidad media de piezas suministradas cada mes. ■

3.7.2 Otras consultas sobre el catálogo

Ya podemos consultar con cierto detalle algunas de las vistas del catálogo.

■ Ejemplo 3.29 Mostrar la información de todos los usuarios del sistema; la vista que nos interesa es `ALL_USERS`. ■

```
SQL> SELECT *  
      FROM ALL_USERS;
```

Puede interesar primero ver el esquema de tal vista mediante `DESCRIBE ALL_USERS` para hacer una proyección más selectiva.

■ **Ejemplo 3.30** Queremos saber qué índices tenemos definidos sobre nuestras tablas, pero en esta ocasión vamos a consultar al propio catálogo para que nos muestre algunas de las vistas que contiene (así ya no necesitamos chuleta).

```
SQL> DESCRIBE DICTIONARY;
```

```
SQL> SELECT * FROM DICTIONARY  
      WHERE table_name LIKE '%INDEX%'
```

Ejercicio 3.39 ¿ Cuál es el nombre de la vista que tienes que consultar y qué campos te pueden interesar?

Ejercicio 3.40 Muestra las tablas *ventas* a las que tienes acceso de consulta junto con el nombre del propietario y su número de identificación en el sistema.

Ejercicio 3.41 Muestra todos tus objetos creados en el sistema. ¿Hay algo más que tablas?

3.7.3 Ejercicios adicionales

Ejercicio 3.42 Mostrar los códigos de aquellos proveedores que hayan superado las ventas totales realizadas por el proveedor 'S1'.

Ejercicio 3.43 Mostrar los mejores proveedores, entendiéndose como los que tienen mayores cantidades totales.

Ejercicio 3.44 Mostrar los proveedores que venden piezas a todas las ciudades de los proyectos a los que suministra 'S3', sin incluirlo.

Ejercicio 3.45 Encontrar aquellos proveedores que hayan hecho al menos diez pedidos.

Ejercicio 3.46 Encontrar aquellos proveedores que venden todas las piezas suministradas por S1.

Ejercicio 3.47 Encontrar la cantidad total de piezas que ha vendido cada proveedor que cumple la condición de vender todas las piezas suministradas por S1.

Ejercicio 3.48 Encontrar qué proyectos están suministrados por todos los proveedores que suministran la pieza P3.

Ejercicio 3.49 Encontrar la cantidad media de piezas suministrada a aquellos proveedores que venden la pieza P3.

Ejercicio 3.50 Queremos saber los nombres de tus índices y sobre qué tablas están montados, indica además su propietario.

Ejercicio 3.51 Implementar el comando DESCRIBE para tu tabla *ventas* a través de una consulta a las vistas del catálogo.

Ejercicio 3.52 Mostrar para cada proveedor la media de productos suministrados cada año.

Piensa con detenimiento el significado de la palabra todos/as en las siguientes tres consultas y resuélvelas convenientemente:

Ejercicio 3.53 Encontrar todos los proveedores que venden una pieza roja.

Ejercicio 3.54 Encontrar todos los proveedores que venden todas las piezas rojas.

Ejercicio 3.55 Encontrar todos los proveedores tales que todas las piezas que venden son rojas.

Ejercicio 3.56 Encontrar el nombre de aquellos proveedores que venden más de una pieza roja.

Ejercicio 3.57 Encontrar todos los proveedores que vendiendo todas las piezas rojas cumplen la condición de que todas sus ventas son de más de 10 unidades.

Ejercicio 3.58 Coloca el status igual a 1 a aquellos proveedores que sólo suministran la pieza P1.

Ejercicio 3.59 Encuentra, de entre las piezas que no se han vendido en septiembre de 2009, las ciudades de aquéllas que se han vendido en mayor cantidad durante Agosto de ese mismo año.

Como ejercicios adicionales se deja al alumno la posibilidad de realizar todas las consultas en SQL correspondientes a las ya resueltas en álgebra relacional sobre la base de datos de la Gestión Docente Universitaria.

3.8 Ejercicios adicionales

3.8.1 Realización de consultas sin operadores de agregación

Considerando el esquema de bases de datos la liga de baloncesto creado en el apartado 1.8:

Equipos (codE, nombreE, localidad, entrenador, fecha_crea)

Jugadores (codJ, codE, nombreJ)

Encuentros(ELocal, EVisitante, fecha, PLLocal,PVisitante)

Alineaciones (codJ, ELocal, EVisitante, Faltas)

a) codE en Jugadores clave externa a Equipos.

b) ELocal en Encuentros clave externa a Equipos y EVisitante en Encuentros clave externa a Equipos.

c) codJ en Alineaciones clave externa a Jugadores.

d) ELocal y EVisitante en Alineaciones clave externa a Encuentros.

Resolver mediante SQL, las siguientes consultas:

Ejercicio 3.60 Muestra la información disponible acerca de los encuentros de liga.

Ejercicio 3.61 Muestra los nombres de los equipos y de los jugadores jugadores ordenados alfabéticamente.

Ejercicio 3.62 Muestra los jugadores que no tienen ninguna falta.

Ejercicio 3.63 Muestra los compañeros de equipo del jugador que tiene por código x ($\text{codJ}='x'$) y donde x es uno elegido por ti.

Ejercicio 3.64 Muestra los jugadores y la localidad donde juegan (la de sus equipos).

■ **Ejemplo 3.31** Muestra todos los encuentros posibles de la liga.

Ejercicio 3.65 Muestra los equipos que han ganado algún encuentro jugando como local.

Ejercicio 3.66 Muestra los equipos que han ganado algún encuentro.

Ejercicio 3.67 Muestra los equipos que todos los encuentros han ganado lo han hecho como equipo local.

Ejercicio 3.68 Muestra los equipos que han ganado todos los encuentros jugando como equipo local.

Ejercicio 3.69 Muestra los encuentros que faltan para terminar la liga.

Ejercicio 3.70 Muestra los encuentros que tienen lugar en la misma localidad.

3.8.2 Realización de consultas con operadores de agregación

Resolver mediante SQL las siguientes consultas sobre la base de datos de la liga de baloncesto:

Ejercicio 3.71 Para cada equipo muestra el número de encuentros que ha disputado como local.

Ejercicio 3.72 Muestra los encuentros en los que se alcanzó mayor diferencia.

Ejercicio 3.73 Muestra los jugadores que no han superado 3 faltas acumuladas.

Ejercicio 3.74 Muestra los equipos con mayores puntuaciones en los partidos jugados fuera de casa.

Ejercicio 3.75 Muestra todas las victorias de cada equipo, jugando como local o como visitante.

Ejercicio 3.76 Muestra el equipo con mayor número de victorias.

Ejercicio 3.77 Muestra el promedio de puntos por equipo en los encuentros de ida.

Ejercicio 3.78 Muestra el equipo con mayor número de puntos en total de los encuentros jugados.