UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

Guía de ejercicios SQL para Sistemas de Bases de Datos I - UCAB

Trabajo de Ascenso presentado como requisito para ascender a la categoría de profesor Asistente

Ing. Ana Karina Fernandes Agrela

Caracas, Mayo 2020

Guía de Ejercicios

SQL

Introducción

Esta guía tiene como objetivo fundamental convertirse en un texto de apoyo para los estudiantes de la asignatura Sistemas de Bases de Datos I impartida en la Escuela de Ingeniería Informática de la Facultad de Ingeniería de la Universidad Católica Andrés Bello.

Con la guía el estudiante tendrá un resumen de los conceptos básicos sobre bases de datos relacionales y un compendio de ejercicios resueltos utilizando el Lenguaje de Consulta Estructurado o conocido por su definición en inglés Structured Query Language (SQL), en ésta se presentan casos prácticos, sin embargo, éste trabajo no pretende sustituir textos académicos, por el contrario pretende apoyar al estudiante en el proceso de aprendizaje del lenguaje utilizado para interactuar con sistemas manejadores de bases de datos.

Debido a la diversidad de lenguajes de programación y bases de datos existentes en el mercado, definir un vocabulario común puede resultar complejo, sin embargo, las bases de datos relacionales, en su mayoría, soportan SQL, un estándar internacional ANSI/ISO de comunicación que permite la definición, manipulación y control de forma sencilla ya que es un lenguaje declarativo y muy parecido al lenguaje natural.

Los ejercicios propuestos se presentan con la solución para que el estudiante pueda analizar y comparar dicha solución con la suya. El contenido de la guía se agrupa en cinco capítulos, a continuación se presenta una breve descripción de cada uno de los capítulos:

- Capítulo 1 realiza una presentación general de los conceptos fundamentales de las bases de datos relacionales.
- Capítulo 2 reseña la historia de SQL y los tipos de instrucciones que lo conforman.
- Capítulo 3 inicia al lector en el lenguaje de definición de datos (DDL).
- Capítulo 4 ofrece un resumen sobre el lenguaje de manipulación de datos (DML).
- Capítulo 5 que presenta un compendio de ejercicios resueltos en SQL que abarca la definición y manipulación de datos.

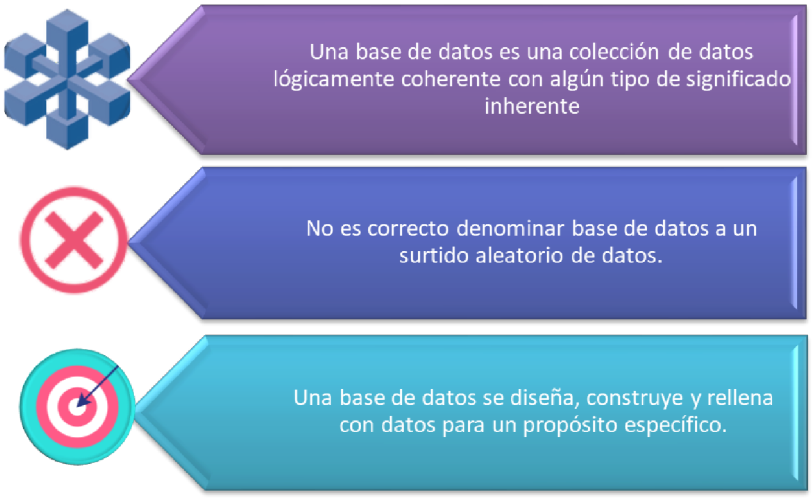
Capítulo

Bases de Datos Relacionales

"Una base de datos típica representa algún aspecto del mundo real y es utilizada por uno o más grupos de usuarios con fines específicos" [1]

El término **base de datos** ha sido utilizado para referirse a cualquier cosa, desde una colección de nombres y direcciones, hasta un complejo sistema de recuperación y almacenamiento de datos que se basa en interfaces de usuarios, una red de computadoras y servidores. Hay tantas definiciones para base de datos como libros sobre éstas. Por otra parte, los distintos proveedores de DBMS (Data Base Management System) han desarrollado diferentes arquitecturas, por lo que no todas las bases de datos están diseñadas de la misma manera. A pesar de la falta de una definición absoluta, la mayoría de las fuentes coinciden en que una base de datos, por lo menos, es una colección de datos organizada en un formato estructurado que es definido como metadatos que describe esa estructura. Puede pensar en los metadatos como información sobre los datos almacenados, que define cómo se almacenan éstos en una base de datos [2].

Por tanto una base de datos se puede definir como una colección de datos relacionados. La palabra datos se refiere a los hechos conocidos que se pueden grabar y que tienen un significado implícito, también representa algún aspecto del mundo real, lo que en ocasiones se denomina minimundo o universo de discurso.



Modelo Relacional

Fue introducido en el año 1970 por E. F. Codd y define una estructura de datos relacional que protege los datos y permite que sean manipulados de manera que es previsible y resistente al error. El modelo relacional, el cual se basa principalmente en los principios matemáticos de la teoría de conjuntos y lógica de predicados, apoya la recuperación de datos sencilla, aplica la integración de datos (la precisión y coherencia de los datos), y proporciona una estructura de base de datos independiente de las aplicaciones al acceder a los datos almacenados. El núcleo del modelo relacional es la relación. [3]

La base del modelo relacional es un concepto matemático conocido como relación, evitando la complejidad de la matemática se puede considerar una relación como una matriz compuesta de filas y columnas que se intersecan. Cada fila en una relación se llama tupla y cada columna representa un atributo.

El modelo relacional de datos se implementa por medio de un complejo sistema de administración de base de datos relacional.

Entidad

Una entidad es una «cosa» u «objeto» en el mundo real que es distinguible de todos los demás objetos. Una entidad tiene un conjunto de propiedades, y los valores para algún conjunto de propiedades pueden identificar una entidad de forma unívoca. Por ejemplo,

la C.I. 6.789.901 identifica unívocamente una persona particular. Análogamente, se puede pensar en los préstamos bancarios como entidades, y un número de préstamo P-15 en la sucursal de la Castellana identifica unívocamente una entidad de préstamo. Una entidad puede ser concreta, como una persona o un libro, o puede ser abstracta, como un préstamo, unas vacaciones o un concepto [4].

Cada entidad tiene atributos (propiedades particulares que la describen).

Atributos

Una entidad se representa mediante un conjunto de atributos. Los atributos describen propiedades que posee cada miembro de un conjunto de entidades. La designación de un atributo para un conjunto de entidades expresa que la base de datos almacena información similar concerniente a cada entidad del conjunto de entidades; sin embargo, cada entidad puede tener su propio valor para cada atributo. Cada entidad tiene un valor para cada uno de sus atributos. Para cada atributo hay un conjunto de valores permitidos, llamados el dominio, o el conjunto de valores, de ese atributo. El dominio del atributo nombre-cliente podría ser el conjunto de todas las cadenas de texto de una cierta longitud. Análogamente, el dominio del atributo número-préstamo podría ser el conjunto de todas las cadenas de la forma «P-n», donde n es un entero positivo. [4].

Un atributo, como se usa en el modelo E-R, se puede caracterizar por los siguientes tipos de atributo:

- **Atributos compuestos y atributos simples (atómicos)**. Los atributos compuestos se pueden dividir en sub partes más pequeñas, que representan atributos más básicos con significados independientes. Por ejemplo, el atributo Dirección de la entidad EMPLEADO, se puede subdividir en DirCalle, Ciudad, Municipio y Código Postal. Los atributos que no son divisibles se denominan atributos simples o atómicos. Los atributos compuestos pueden formar una jerarquía. Por ejemplo, DirCalle se puede subdividir en tres atributos simples: Número, Calle y NumApto.

El valor de un atributo compuesto es la concatenación de los valores de sus atributos simples. [1]

- **Atributos monovalor y multivalor**. La mayoría de los atributos tienen un solo valor para una entidad en particular; dichos atributos reciben el nombre de monovalor o de un solo valor. Por ejemplo, Edad es un atributo monovalor de una persona. En algunos casos, un atributo puede tener un conjunto de valores para la misma entidad (por ejemplo, un atributo Colores para un vehículo, o un atributo Licenciaturas para una persona). Los vehículos con un solo color tiene un solo valor, mientras que los vehículos de dos tonos tienen dos valores de color. De forma parecida, puede que una persona no tenga ninguna licenciatura, otra puede que tenga una, y una tercera persona puede que tenga dos o más; por consiguiente, diferentes personas pueden tener una cantidad de valores diferente para el atributo Licenciaturas. Dichos atributos se denominan multivalor. Un atributo multivalor puede tener límites superior e inferior para restringir el número de valores permitidos para cada entidad individual.[1]
- **Atributos almacenados y derivados**. En algunos casos, dos (o más) valores de atributo están relacionados (por ejemplo, los atributos Edad y FechaNac de una persona). Para una entidad de persona en particular, el valor de Edad puede determinarse a partir de la fecha actual (el día de hoy) y el valor de FechaNac de esa persona. El atributo Edad se denomina entonces atributo derivado y se dice que se ha derivado del atributo FechaNac, que es el denominado atributo almacenado. Algunos valores de atributo se pueden derivar de entidades relacionadas; por ejemplo, un atributo NumEmpleados de una entidad DEPARTAMENTO puede derivarse contando el número de empleados relacionados con (o que trabajan para) ese departamento. [1]

Relación

Una relación es una asociación entre diferentes entidades. Por ejemplo, se puede definir una relación que asocie al cliente López con el préstamo P-15. Esta relación específica

que López es un cliente con el préstamo número P-15. Un conjunto de relaciones es un conjunto de relaciones del mismo tipo. [4]

Razón de cardinalidad y participación

La correspondencia de cardinalidades, o razón de cardinalidad, expresa el número de entidades a las que otra entidad puede estar asociada vía un conjunto de relaciones. [4]

La razón de cardinalidad de una relación especifica el número máximo de instancias de relación en las que una entidad puede participar.

La restricción de participación especifica si la existencia de una entidad depende de si está relacionada con otra entidad a través de un tipo de relación. Hay dos tipos de restricciones de participación, total y parcial

Sistema de gestión de bases de datos (DBMS)

Es una colección de programas que permite a los usuarios crear y mantener una base de datos. El DBMS es un sistema de software de propósito general que facilita los procesos de definición, construcción, manipulación y compartición de bases de datos entre varios usuarios y aplicaciones. Definir una base de datos implica especificar los tipos de datos, estructuras y restricciones de los datos que se almacenarán en la base de datos. La definición o información descriptiva de una base de datos también se almacena en esta última en forma de catálogo o diccionario de la base de datos; es lo que se conoce como metadatos. La construcción de la base de datos es el proceso consistente en almacenar los datos en algún medio de almacenamiento controlado por el DBMS. La manipulación de una base de datos incluye funciones como la consulta de la base de datos para recuperar datos específicos, actualizar la base de datos para reflejar los cambios introducidos en el minimundo y generar informes a partir de los datos. Compartir una base de datos permite que varios usuarios y programas accedan a la base de datos de forma simultánea. Una aplicación accede a la base de datos enviando consultas o solicitudes de datos al DBMS. Una consulta normalmente provoca la recuperación de

algunos datos; una transacción puede provocar la lectura o la escritura de algunos datos en la base de datos.

Otras funciones importantes ofrecidas por el DBMS son la protección de la base de datos y su mantenimiento durante un largo periodo de tiempo. La protección incluye la protección del sistema contra el funcionamiento defectuoso del hardware o el software (caídas) y la protección de la seguridad contra el acceso no autorizado o malintencionado. Una gran base de datos típica puede tener un ciclo de vida de muchos años, por lo que el DBMS debe ser capaz de el sistema de bases de datos permitiendo que el sistema evolucione según cambian los requisitos con el tiempo.[1]

Capítulo

SQL: Lenguaje de Consulta Estructurado

"El lenguaje SQL se puede considerar como una de las principales razones del éxito comercial de las bases de datos relacionales." [1]

El lenguaje SQL se puede considerar como una de las principales razones del éxito comercial de las bases de datos relacionales. Como se convirtió en un estándar para estas últimas, los usuarios perdieron el miedo a migrar sus aplicaciones de base de datos desde otros tipos de sistemas de bases de datos (por ejemplo, sistemas de red o jerárquicos) a los sistemas relacionales, porque aunque estuvieran satisfechos con el producto DBMS relacional que estaban utilizando, no esperaban que la conversión a otro producto DBMS relacional fuera caro y consumiera mucho tiempo, ya que ambos sistemas seguían los mismos estándares en cuanto al lenguaje. En la práctica, por supuesto, hay muchas diferencias entre los distintos paquetes DBMS relacionales comerciales. Sin embargo, si el usuario sólo utiliza las funciones que forman parte del estándar, y si ambos sistemas relacionales soportan fielmente el estándar, la conversión entre los dos sistemas es mucho más sencilla. Otra ventaja de disponer de un estándar es que los usuarios pueden escribir sentencias en una aplicación de base de datos para acceder a los datos almacenados en dos o más DBMSs relacionales sin tener que cambiar el sublenguaje de base de datos (SQL), siempre y cuando esos DBMS soporten el SQL estándar. [1]

Historia de SQL

La historia de SQL empieza en 1974 con la definición, por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de IBM, de un lenguaje para la especificación de las características de las bases de datos que adopta el modelo relacional. Este lenguaje se llamaba SEQUEL (Structured English Query Language) y se implementó en un prototipo llamado SEQUEL-XRM entre 1974 y 1975. Las experimentaciones con ese prototipo condujeron, entre 1976 y 1977, a una revisión del lenguaje (SEQUEL/2), que a partir de ese momento cambió de nombre por motivos legales, convirtiéndose en SQL [5].

El prototipo (System R), basado en este lenguaje, se adoptó y utilizó internamente en IBM y lo adoptaron algunos de sus clientes elegidos. Gracias al éxito de este sistema, que no

estaba todavía comercializado, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL. A partir de 1981, IBM comenzó a entregar sus productos relacionales y en 1983 empezó a vender DB2. En el curso de los años ochenta, numerosas compañías (por ejemplo Oracle y Sybase, sólo por mencionar algunos) comercializaron productos basados en SQL, que se convierte en el estándar industrial que respecta a las bases de datos relacionales. En 1986, el ANSI (American National Standards Institute) adoptó SQL como estándar para los lenguajes relacionales y en 1987 se transformó en estándar ISO (International Standards Organization) [1].

En los años siguientes, SQL ha sufrido diversas revisiones que han conducido primero a la versión SQL/89, posteriormente la versión SQL/92, y la SQL/1999, y la actual SQL/2016. El hecho de tener un estándar definido por un lenguaje para bases de datos relacionales abre potencialmente el camino a la intercomunicabilidad entre todos los productos que se basan en él. En general cada productor adopta e implementa en la propia base de datos sólo el corazón del lenguaje SQL, extendiéndose de manera individual según la propia visión que cada cual tenga del mundo de las bases de datos.

SQL

El nombre SQL significa Lenguaje de Consulta Estructurado (Structured Query Language). Es un lenguaje de bases de datos global, cuenta con sentencias para definir datos, consultas y actualizaciones. Además, dispone de características para definir vistas en la base de datos, especificar temas de seguridad, autorización, definir restricciones de integridad y especificar controles de transacciones. También tiene reglas para incrustar sentencias de SQL en un lenguaje de programación de propósito general, como Java, C/C++, entre otros.

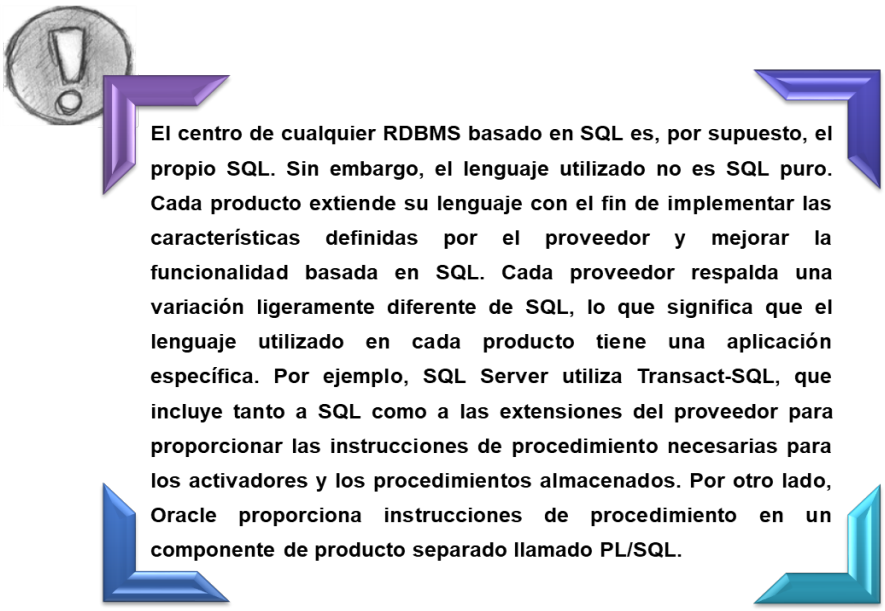
Tipos de instrucciones de SQL

Aunque SQL se considera un sublenguaje debido a su naturaleza de no procesamiento, aun así es un lenguaje completo que le permite crear y mantener objetos en una base

de datos, asegurar esos objetos y manipular la información dentro de los objetos. Un método común usado para categorizar las instrucciones SQL es dividirlas de acuerdo con las funciones que realizan. Basado en este método, SQL se separa en tres tipos de instrucciones:

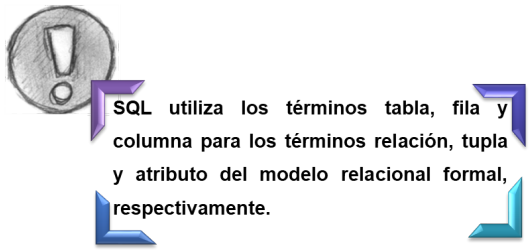
- Lenguaje de definición de datos (**DDL**, Data Definition Language): Las instrucciones DDL se usan para crear, modificar o eliminar objetos en una base de datos tales como tablas, vistas, esquemas, índices, procedimientos, entre otros. Las palabras clave en SQL más frecuentemente asociadas con las instrucciones DDL son: CREATE, ALTER y DROP. Por ejemplo, se usa la instrucción CREATE TABLE para crear una tabla, la instrucción ALTER TABLE para modificar las características de una tabla, y la instrucción DROP TABLE para borrar la definición de la tabla de la base de datos.
- Lenguaje de control de datos (**DCL**, Data Control Language): Las instrucciones DCL permiten controlar quién o qué (un usuario en una base de datos puede ser una persona o un programa de aplicación) tiene acceso a objetos específicos en la base de datos. Con DCL, puede otorgar o restringir el acceso usando las instrucciones GRANT o REVOKE, los dos comandos principales en DCL. Las instrucciones DCL también permiten controlar el tipo de acceso que cada usuario tiene a los objetos de una base de datos. Por ejemplo, puede determinar cuáles usuarios pueden ver un conjunto de datos específico y cuáles usuarios pueden manipular esos datos.
- Lenguaje de manipulación de datos (**DML**, Data Manipulation Language): Las instrucciones DML se usan para recuperar, agregar, modificar o borrar datos almacenados en los objetos de una base de datos. Las palabras clave asociadas con las instrucciones DML son SELECT, INSERT, UPDATE y DELETE, las cuales representan los tipos de instrucciones que probablemente son más usadas. Por ejemplo, puede usar la instrucción

SELECT para recuperar datos de una tabla y la instrucción INSERT para agregar datos a una tabla.



Ejecución de una cláusula en SQL

La ejecución de una cláusula en SQL se cumple en varias etapas. En primer lugar, se tiene una fase de análisis y verificación, en donde se verifica si una cláusula se ha formulado en forma correcta. Posteriormente, la expresión se descompone en componentes más simples para que sean analizados por el optimizador. Además, se verifican los nombres de las tablas y columnas en el catálogo. En segundo lugar se tiene una etapa de optimización, en donde se usan estadísticas del almacenamiento de los datos por medio del catálogo. Así se examinan varios posibles caminos de acceso a los datos, se calcula el costo de cada uno y se escoge el mejor. Finalmente, se genera un plan para la ejecución de la cláusula [6].



Una sentencia SQL es como una frase que dice lo que se quiere obtener y de donde obtenerlo. Todas las sentencias empiezan con un verbo (palabra reservada que indica la acción a realizar), seguido del resto de cláusulas, algunas obligatorias y otras opcionales que completan la frase. Todas las sentencias siguen una sintaxis para que se puedan ejecutar correctamente, dichas sentencias SQL pueden variar entre los diferentes motores.

Capítulo 3

Lenguaje de Definición de Datos (DDL)

"SQL incluye comandos para crear objetos de base de datos como tablas, índices, vistas, entre otros." [7]

Creación de una Base de Datos

Antes de crear una base de datos en cualquier sistema, asegúrese de leer primero la documentación del producto, para tener la certeza que es seguro agregar una base de datos objeto en el entorno de SQL. Una vez que se crea la base de datos, se pueden crear esquemas, tablas, vistas y otros objetos dentro de esa base de datos, y desde allí llenar las tablas con los datos necesarios.

A pesar del hecho de que el estándar SQL no define qué es una base de datos, y mucho menos proporciona una instrucción para crear cualquier tipo de base de datos objeto, la mayoría de los RDBMS respaldan la creación de una base de datos objeto, además puedan utilizar a ese objeto como base para su estructura jerárquica en la gestión de datos objetos.

Sintaxis:

```
CREATE DATABASE <nombre de la base de datos> <parámetros adicionales>
```

Ejemplo:

```
CREATE DATABASE UCAB
```

Tipos de datos y dominios

Los tipos de datos básicos disponibles para los atributos son numérico, cadena de caracteres, cadena de bits, booleano, fecha y hora.

- Tipos de datos de cadena: Los valores permitidos se deben extraer de un conjunto de caracteres, ya sea de un conjunto predeterminado o de un conjunto definido en el momento que la columna se define. Los tipos de datos en la cadena de caracteres incluyen:
  - `CHARACTER o CHAR`
  - `CHARACTER VARYING o VARCHAR`
  - `CHARACTER LARGE OBJECT o CLOB`, destinado a especificar columnas que tienen valores de texto más largos, como los documentos.
- Cadenas de carácter nacional: Los valores permitidos son similares a los de las cadenas de caracteres, salvo que el conjunto de caracteres asociados con estos tipos de datos se definen por la aplicación. Como resultado, cuando una

cadena de carácter nacional se especifica, los valores asociados con ese tipo de datos deben basarse en el conjunto de caracteres especificado por el sistema de gestión de base de datos relacional (RDBMS) para las cadenas de carácter nacional. Éstas son útiles para almacenar cadenas de caracteres en varios lenguajes en la misma base de datos. Los tipos de datos en la cadena de carácter nacional incluyen:

- `NATIONAL CHARACTER`
- `NATIONAL CHARACTER VARYING`
- `NATIONAL CHARACTER LARGE OBJECT`

➔ Cadenas de bits: Los valores permitidos se basan en bits de datos (dígitos binarios), en lugar de conjuntos de caracteres, lo que significa que estos tipos de datos permiten sólo valores de 0 o 1. SQL respalda dos formas de tipos de datos de cadena de bits:

- `BIT`
- `BIT VARYING`
- Existe otro tipo de datos denominado `BINARY LARGE OBJECT` o `BLOB` destinado a especificar las columnas que tienen valores binarios grandes, como las imágenes.

➔ Tipos de datos numéricos: Los valores especificados por los tipos de datos numéricos son números. Todos los tipos de datos numéricos tienen una precisión, y algunos tienen una escala. La precisión se refiere al número de dígitos (dentro de un valor numérico específico) que se pueden almacenar. La escala se refiere al número de dígitos en la parte fraccional de ese valor (los dígitos a la derecha del punto decimal). Por ejemplo, el número 435.27 tiene una precisión de 5 y una escala de 2. La escala no puede ser un número negativo o ser más larga que la precisión. Una escala de 0 indica que el número es un número entero y no contiene ningún componente fraccional. SQL define dos formas de tipos de datos numéricos:

- Numéricos exactos: Tienen precisión y escala. Los tipos de datos numéricos exactos incluyen:

- NUMERIC
- DECIMAL
- INTEGER
- SMALLINT

- Numéricos aproximados: Los valores permitidos tienen precisión pero no escala. Como resultado, el punto decimal puede flotar. Un número de punto flotante es aquel que contiene un punto decimal, pero el punto decimal se puede localizar en cualquier lugar dentro del número, por lo que se dice que un numérico aproximado no tiene escala. Los tipos de datos numéricos aproximados incluyen:
  - REAL
  - DOUBLE PRECISION
  - FLOAT

➔ Tipos de datos de fecha y hora: Como su nombre lo indica, los tipos de datos de fecha y hora se refieren a las formas de seguimiento de fechas y horas. SQL define tres formas de tipos de datos de fecha y hora:

- DATE (fecha)
- TIME (hora)
- TIMESTAMP (fecha y hora)

➔ Tipo de datos booleanos: El tipo de datos booleano es muy sencillo y fácil de aplicar. El tipo de dato respalda una construcción de verdadero/falso que permite sólo tres valores: verdadero, falso o desconocido. Un valor nulo se evalúa como desconocido. (En SQL, un valor nulo se utiliza para indicar que un valor no está definido o no se conoce).

- BOOLEAN

¿Cómo se pueden comparar los tipos de datos predefinidos en SQL con los tipos de datos que se encuentran en otros lenguajes de programación?

En su mayor parte, es poco probable que los tipos de datos de dos diferentes lenguajes sean los mismos. Un conjunto de tipos de datos en un lenguaje pueden variar en estructura y semántica de un conjunto de tipos de datos en otro lenguaje. Estas diferencias, a menudo llamadas desajuste de impedancia, pueden conducir a la pérdida de información cuando una aplicación toma datos de una base de datos de SQL. De hecho, a menudo es una buena idea saber qué lenguaje se usará para las aplicaciones cuando se diseña la base de datos. En algunos casos, el diseño de la base de datos puede afectar el lenguaje de aplicación que más fácilmente se utiliza para manipular los datos en una base de datos de SQL. Sin embargo, SQL incluye una expresión de conversión de datos llamada CAST. La expresión CAST permite convertir datos de un tipo de datos a otro tipo de datos, permitiendo al lenguaje anfitrión acceder valores que no podrían manejarse en su forma original [2].

Especificación de restricciones y valores predeterminados de atributo

SQL permite NULL como valor de atributo, es posible especificar una restricción NOT NULL si no se permite NULL para un atributo en particular. Esto siempre se especifica implícitamente para los atributos que forman parte de la clave principal de cada relación, pero puede especificarse para cualquier otro atributo para cuyo valor se exige que no sea NULL. También es posible definir un valor predeterminado para un atributo añadiendo la cláusula DEFAULT <valor> a su definición. El valor predeterminado se incluye en cualquier tupla nueva si no se proporciona un valor explícito para ese atributo. Si no se especifica una cláusula predeterminada, el valor predeterminado es NULL para los atributos que no tienen la restricción NOT NULL.

Ejemplo:

```
CREATE TABLE EMPLEADO(...

        NOMBRE VARCHAR(20) NOT NULL  DEFAULT 'SIN_NOMBRE' ...)
```

Otro tipo de restricción puede ser restringir los valores de atributo o dominio con la cláusula CHECK. Por ejemplo, suponga que los números de departamento están restringidos a número enteros entre 1 y 20.

Ejemplo:

```
NumeroDpto NUMBER NOT NULL CHECK (NumeroDpto > 0 AND NumeroDpto < 21);
```

Creación de tablas

SQL respalda tres tipos de tablas: tablas base, tablas derivadas y tablas vistas. La mayoría de las tablas base son objetos de esquema que tienen los datos de SQL. Las tablas derivadas son los resultados que se observan cuando se solicitan (consultan) datos de una base de datos. Las tablas vistas son otro nombre para las vista.

- **Tablas base persistentes:** Un objeto de esquema nombrado definido por la definición de una tabla en la instrucción CREATE TABLE. Las tablas base persistentes tienen los datos de SQL que se almacenan en la base de datos. Éste es el tipo más común de tabla base y es a menudo a lo que se refiere la gente cuando menciona tablas base o tablas.
- **Tablas temporales globales:** Un objeto de esquema nombrado definido por una definición de tabla en la instrucción CREATE GLOBAL TEMPORARY TABLE. Aunque la definición de la tabla es parte del esquema, la tabla actual existe sólo cuando se hace referencia dentro del contexto de la sesión SQL en la cual se creó. Cuando la sesión termina, la tabla ya no existe. No se puede acceder a una tabla temporal global creada en una sesión desde otra sesión de SQL. Los contenidos son distintos en cada sesión de SQL.
- **Tablas temporales locales creadas:** Un objeto de esquema nombrado definido por una definición de tabla en la instrucción CREATE LOCAL TEMPORARY TABLE. Al igual que una tabla temporal global, sólo se puede hacer referencia a una tabla temporal local creada dentro del contexto de la sesión de SQL en la cual se creó,

y no se puede acceder desde otra sesión de SQL. Sin embargo, se puede acceder a una tabla global desde cualquier lugar dentro de una sesión asociada de SQL, mientras que en una tabla temporal local sólo se podrá acceder dentro del módulo asociado. Los contenidos son distintos dentro de ese módulo.

- **Tablas temporales locales declaradas:** Una tabla declarada como parte de un procedimiento en un módulo. La definición de la tabla no se incluye en el esquema y no existe hasta que ese procedimiento se ejecuta. Al igual que otras tablas temporales, sólo se hace referencia a una tabla temporal local declarada dentro del contexto de la sesión SQL en la cual se creó.

Se puede utilizar una forma de la instrucción CREATE TABLE para crear todos los tipos de tablas, excepto las tablas temporales locales declaradas. [2]

La sintaxis en la instrucción CREATE TABLE:

```
CREATE [ {GLOBAL | LOCAL} TEMPORARY ] TABLE <nombre de la tabla>
( <elemento de la tabla> [ {, <elemento de la tabla> }... ] )
[ ON COMMIT { PRESERVE | DELETE } ROWS ]
```

Ejemplo:

```
CREATE TABLE ESTUDIANTE (
      ID           INT,
      NOMBRE       VARCHAR(60),
      FNACIMIENTO  DATE,
      INSCRITO     BOOLEAN );
```

Modificación de tablas

La instrucción ALTER TABLE para modificar las definiciones de tablas base almacenadas en la base de datos. En su forma más básica, la sintaxis para la instrucción ALTER TABLE se representa:

```
ALTER TABLE <nombre de la tabla> ADD [COLUMN] <definición de columna>
```

```
        | ALTER [COLUMN] <nombre de columna>    { SET DEFAULT <valor
predeterminado> | DROP DEFAULT }

        | DROP [COLUMN] <nombre columna> { CASCADE | RESTRICT }
```

La instrucción permite tomar tres diferentes acciones: añadir columnas, modificar columnas o eliminar columnas. [2]

Ejemplo:

```
ALTER TABLE ESTUDIANTE ADD COLUMN FECHA_INSCRITO DATE;
```

A diferencia de la cláusula ADD [COLUMN], la cláusula ALTER [COLUMN] se limita a dos acciones: establecer una predeterminación o eliminar una predeterminación (aunque existen aplicaciones de productos que permiten cambiar otras propiedades, tal como el tipo de datos o la precisión y la escala)

Eliminación de tablas

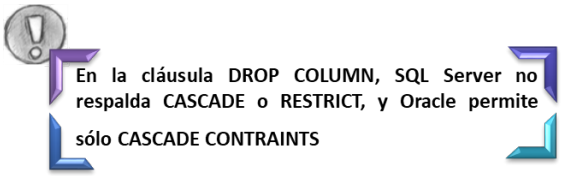
El proceso de eliminación de una tabla y sus datos almacenados es muy sencillo. La siguiente sintaxis muestra el proceso:

```
DROP TABLE <nombre de la tabla> { CASCADE | RESTRICT }
```

La única decisión que necesita tomar cuando se elimina una tabla es si debe escoger la opción CASCADE o RESTRICT. Si se utiliza CASCADE, la tabla y sus datos se eliminan, junto con todas las vistas, restricciones, rutas o activadores que hacen referencia a la tabla. Si se utiliza RESTRICT, la tabla se elimina sólo si no existen dependencias. [2]

Ejemplo:

```
DROP TABLE ESTUDIANTE;
```



Restricción de Integridad

Una base de datos SQL debe hacer más que sólo almacenar datos. Se debe asegurar que el almacenamiento de los datos es el correcto. Si la integridad de los datos se compromete, los datos pueden ser inexactos o inconsistentes, poniendo en cuestionamiento la fiabilidad de la base de datos. Con el fin de asegurar la integridad de los datos, SQL proporciona una serie de restricciones de integridad, reglas que se aplican a la base de datos para restringir los valores que se pueden colocar en esas tablas. Se pueden aplicar restricciones a columnas individuales, a tablas individuales o a múltiples tablas.

- **Restricción NOT NULL:** Nulo significa que un valor no está definido o no se conoce. Esto no es lo mismo que cero, en blanco, una cadena vacía o un valor predeterminado. En lugar de ello, indica que un valor de dato está ausente. Se puede pensar en un valor nulo como una marca. (Una marca es un carácter, número o bit que indica un cierto hecho sobre una columna. La marca sirve como un marcador que designa una condición particular o la existencia de algo.) En el caso nulo, si una columna no proporciona ningún valor, se coloca la marca, indicando que el valor es desconocido, o nulo. Cada columna tiene una característica de nulo que indica si la columna acepta valores nulos. Por predeterminación, todas las columnas aceptan valores nulos. Sin embargo, se puede anular la característica de nulo predeterminada utilizando una restricción NOT NULL, que indica que la columna no acepta valores nulos. La siguiente sintaxis aplica cuando se crea una definición de columna:

```
<nombre de columna> { <tipo de datos> | <dominio> } NOT NULL
```

Ejemplo:

```
CREATE TABLE ESTUDIANTE (
      ID                INT          NOT NULL,
      NOMBRE            VARCHAR(60)  NOT NULL,
      SEGUNDO_NOMBRE    VARCHAR(60) );
```

- **Restricción UNIQUE:** La restricción UNIQUE permite exigir que una columna o conjunto de columnas contengan valores únicos, valores significativos que sean diferentes de todas las demás filas en la misma tabla. Para crear una restricción de columna, se debe añadir como parte de la definición de una columna, como se muestra en la siguiente sintaxis:

```
<nombre de columna> { <tipo de datos> | <dominio> } UNIQUE
```

Ejemplo:

```
CREATE TABLE ESTUDIANTE (
      NOMBRE                    VARCHAR(40),
      EXPEDIENTE                VARCHAR(60) UNIQUE,
      NUMERO_DE_IDENTIFICACION   INT );
```

Si se quiere añadir una restricción única como una restricción de tabla, se debe agregar como un elemento de definición de tabla, como se muestra en la siguiente sintaxis:

```
[ CONSTRAINT <nombre de la restricción> ]

UNIQUE ( <nombre de columna> [{, <nombre de columna>}... ] )
```

Ejemplo:

```
CREATE TABLE ESTUDIANTE (
      NOMBRE       VARCHAR(40),
      APELLIDO     VARCHAR(60),
      INSCRITO     INT,
      CONSTRAINT ESTUDIANTE UNIQUE ( NOMBRE, APELLIDO ) );
```

Las columnas NOMBRE y APELLIDO ahora deben contener combinaciones únicas de valores con el fin de que una fila se añade a la tabla ESTUDIANTE.

- **Restricción FOREIGN KEY:** La restricción FOREIGN KEY se ocupa de cómo los datos en una tabla hacen referencia a los datos en otra tabla, que es la razón por

la que se conoce como una restricción referencial (en relación con otra tabla). Las tablas en una base de datos relacional están unidas entre sí de una manera significativa con el fin de garantizar la integridad de los datos. Esta asociación entre tablas forma una relación que proporciona una integridad referencial entre las tablas. La integridad referencial evita la manipulación de los datos en una tabla que afecte negativamente los datos en otra tabla. [2]

Si se quiere añadir una restricción foreign key como una restricción de tabla, se debe agregar como un elemento de definición de tabla, como se muestra en la siguiente sintaxis:

```
[CONSTRAINT <nombre de la restricción>]
FOREIGN KEY (<columna referenciada> [{, <columna referenciada>}...])REFERENCES <tabla referida> [(columnas de referencia>)]

Ejemplo:
CREATE TABLE TITULOS_CD
( ID_TITULO_CD      INT,
  TITULO_CD         VARCHAR(60),
  ID_EDITOR         INT,
  CONSTRAINT FK_ID_EDITOR FOREIGN KEY (ID_EDITOR)
  REFERENCES EDITORES_CD (ID_EDITOR) );
```

Si se quiere añadir una restricción FOREIGN KEY como una restricción de columna, se debe agregar la restricción a la definición de una columna, como se muestra en la siguiente sintaxis:

```
<nombre de columna> { <tipo de datos> | <dominio> } [NOT NULL]
REFERENCES <tabla referenciada> [( <columnas referenciadas> )]

Ejemplo:
CREATE TABLE TITULOS_CD
( ID_TITULO_CD      INT,
  TITULO_CD         VARCHAR(60),
  ID_EDITOR         INT REFERENCES EDITORES_CD );
```

➔ **Restricción CHECK:** Se puede definir como restricciones de tabla, restricciones de columna, restricciones de dominio, o en afirmaciones. Una restricción CHECK permite especificar qué valores se pueden incluir en una columna. Se puede definir un rango de valores (por ejemplo, entre 10 y 100), una lista de valores (por ejemplo, blues, jazz, pop, country), o una serie de otras condiciones que restringen exactamente qué valores se permiten en una columna. La sintaxis básica para una restricción CHECK es relativamente sencilla. Para crear una restricción CHECK de columna, utilice la siguiente sintaxis en la definición de columna:

```
<nombre de columna> { <tipo de datos> | <dominio> }
CHECK ( <condición de búsqueda> )
```

Para crear una restricción CHECK de tabla, utilice la siguiente sintaxis en la definición de la tabla:

```
[ CONSTRAINT <nombre de restricción> ]
CHECK ( <condición de búsqueda> )
```

Ejemplos:

```
CREATE TABLE TITULOS_CD
( ID_DISCO_COMPACTO        INT,
  TITULO_CD                VARCHAR(60)  NOT NULL,
  ERA                      CHAR(5),
   CONSTRAINT CK_ERA CHECK ( ERA IN ( '1940s', '1950s','1960s',
'1970s', '1980s', '1990s', '2000s' ) ) );


CREATE TABLE TITULOS_CD_2
( ID_DISCO_COMPACTO        INT,
  TITULO_CD                VARCHAR(60) NOT NULL,
  EN_EXISTENCIA            INT    NOT NULL,
  CONSTRAINT CK_EN_EXISTENCIA CHECK
( ( EN_EXISTENCIA BETWEEN 0 AND 30 ) OR
( EN_EXISTENCIA BETWEEN 49 AND 60 ) ) ) ;
```

Vistas en SQL

Una vista en terminología SQL es una tabla que deriva de otras tablas. Esas otras tablas pueden ser tablas base o vistas definidas anteriormente. Una vista no existe necesariamente en formato físico; está considerada como una tabla virtual, en oposición a las tablas base, cuyas tuplas están realmente almacenadas en la base de datos. Esto limita las posibles operaciones de actualización que pueden aplicarse a las vistas, pero no ofrecen limitación alguna al consultar una vista. Una vista es una forma de especificar una tabla a la que nos referimos con frecuencia, aunque no exista físicamente. En SQL se utiliza el comando CREATE VIEW para especificar una vista. A una vista se le asigna un nombre de tabla (virtual), o nombre de vista, una lista de nombres de atributos y una consulta que sirve para especificar el contenido de la vista. Si ninguno de los atributos de la vista resulta de aplicar funciones u operaciones aritméticas, no es necesario especificar nombres de atributos para la vista, puesto que serían idénticos a los de los atributos de las tablas de definición.

Ejemplo:

```
CREATE VIEW TRABAJA AS
       SELECT Nombre, Apellido1, NombreProyecto, Horas
       FROM EMPLEADO,PROYECTO,TRABAJA_EN
       WHERE Dni = DniEmpleado AND NumProy = NumProyecto;
```

Se supone que una vista siempre está actualizada; si se modifican las tuplas de las tablas base sobre las que se define la vista, esta última debe reflejar esos cambios automáticamente. Por tanto, la vista no se materializa al definir la vista, sino al especificar una consulta en la vista. La tarea de que la vista esté actualizada es responsabilidad del DBMS, y no del usuario. Si ya no se necesita una vista, se puede utilizar el comando DROP VIEW para eliminarla.

Ejemplo:

```
DROP VIEW TRABAJA;
```

Capítulo 4

Lenguaje de Manipulación de Datos (DML)

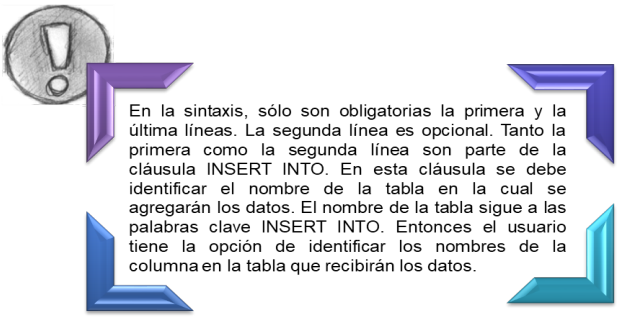
"Una de las funciones principales de cualquier base de datos es la capacidad de manejar los datos almacenados dentro de sus tablas." [2]

Insertar datos en SQL

La instrucción INSERT permite agregar datos a las diferentes tablas en una base de datos. La sintaxis de una instrucción INSERT básica es relativamente sencilla:

Sintaxis:

```
INSERT INTO <nombre de la tabla>
[ ( <nombre de la columna> ) [ { ,<nombre de la columna> }... ] ) ]
 VALUES ( <valor> [ { , <valor> }... ] )
```



Ejemplo:

```
INSERT INTO facultad (fac_codigo, fac_nombre)
 VALUES ( 1 , 'Humanidades')
```

Se puede obviar el nombre de las columnas

```
INSERT INTO facultad VALUES ( 1 , 'Humanidades')
```

1. Si los nombres de columna no se especifican en la cláusula INSERT INTO, entonces deberá haber un valor por cada columna en la tabla y los valores deberán estar en el mismo orden en el que están definidos en la tabla.
2. Si los nombres de columna se especifican en la cláusula INSERT INTO, entonces deberá haber exactamente un valor por cada columna especificada y esos valores deberán estar en el mismo orden en el que están definidos en la cláusula INSERT INTO. Sin embargo, los nombres y valores de columna no tienen que estar en el mismo orden que las columnas en la definición de la tabla.

3. Se debe proporcionar un valor por cada columna en la tabla excepto para las columnas que permiten valores nulos o que tienen un valor definido por defecto.
4. Cada valor con un carácter del tipo de datos de cadena debe estar encerrado en comillas simples.
5. Se puede utilizar la palabra clave NULL (o null) como el valor de los datos en la cláusula VALUES para asignar un valor nulo a cualquier columna que permita valores nulos.

Actualizar datos en SQL

El comando UPDATE se utiliza para modificar los valores de atributo de una o más tuplas seleccionadas. Con la instrucción UPDATE se pueden modificar datos en una o más filas para una o más columnas. La sintaxis para la instrucción UPDATE se puede mostrar de la manera siguiente:

Sintaxis:

```
UPDATE <nombre de la tabla>
SET <determinar expresión de la cláusula> [{,<determinar expresión de la cláusula> }... ] [
 WHERE <condición de búsqueda> ]
```

Ejemplo:

```
UPDATE estudiante
SET inscrito = 'SI'
```

Como se puede ver, la cláusula UPDATE y la cláusula SET son obligatorias, mientras que la cláusula WHERE es opcional. En la cláusula UPDATE se debe especificar el nombre de la tabla que se está actualizando. En la cláusula SET se debe especificar una o más expresiones de cláusula, es decir, se especifica los atributos que se modificarán y sus nuevos valores. En la cláusula WHERE, se debe especificar una condición de búsqueda, se especifica una condición o conjunto de condiciones que actúa como un filtro para las filas que se están actualizando. Solamente las filas que cumplen con estas condiciones son actualizadas.

Insertar valores desde una instrucción SELECT

Para insertar datos se considera la cláusula VALUES como obligatoria, sin embargo, existe una alternativa y es que se puede utilizar una instrucción SELECT para especificar los valores que se quieran ingresar en una tabla. La clave para utilizar una instrucción SELECT, al igual que al utilizar una cláusula VALUES, es asegurarse que el número de valores aplicados por la instrucción SELECT coincida con el número requerido de valores en la tabla, y que éstos cumplan con cualquier restricción de la tabla correspondiente.

Ejemplo:

```
INSERT INTO INVENTARIO_CD_2
SELECT NOMBRE_CD, EN_EXISTENCIA FROM INVENTARIO_CD;
```

Actualizar valores desde una instrucción SELECT

Se puede utilizar una instrucción SELECT en la cláusula SET de una instrucción UPDATE. La instrucción SELECT se debe agregar a la derecha del signo de igual.

Ejemplo:

```
UPDATE INVENTARIO_CD_2
SET EN_EXISTENCIA_2 = ( SELECT AVG(EN_EXISTENCIA) FROM INVENTARIO_CD );
```

Se puede agregar una cláusula WHERE a la instrucción SELECT.

Eliminar datos en SQL

La instrucción DELETE elimina tuplas de una relación. Incluye una cláusula WHERE, parecida a la que se utiliza en una consulta SQL, para seleccionar las tuplas que se van a eliminar. Las tuplas se eliminan explícitamente sólo de una tabla a la vez. Sin embargo, la eliminación se puede propagar a tuplas de otras relaciones si se han especificado acciones de activación referencial en las restricciones de integridad referencial del DDL. En función del número de tuplas seleccionadas por la condición de la cláusula WHERE, ninguna, una o varias tuplas pueden ser eliminadas por un solo comando DELETE. La

ausencia de una cláusula WHERE significa que se borrarán todas las tuplas de la relación; sin embargo, la tabla permanece en la base de datos, pero vacía. [1]

Ejemplo:

```
DELETE FROM EMPLEADO
WHERE Apellido = 'Cabrera';
```

Consultar datos en SQL

La instrucción SELECT permite formar consultas intrincadas que pueden devolver exactamente el tipo de datos que se desea recuperar. Es una de las instrucciones más comunes y también es una de las instrucciones más flexible y amplia en el estándar SQL.

La instrucción SELECT es una expresión de consulta que comienza con la palabra clave SELECT e incluye una serie de elementos que forman la expresión. La sintaxis básica para la instrucción SELECT puede dividirse en varias cláusulas específicas, cada una de las cuales ayuda a refinar la consulta para que sólo se devuelvan los datos requeridos. La sintaxis para la instrucción SELECT puede mostrarse como sigue:

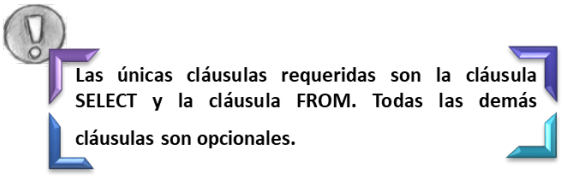
```
SELECT [ DISTINCT | ALL ] { * | <lista de atributos> }
FROM <tabla de referencia> [ {, <tabla de referencia> }... ]
[ WHERE <condición de búsqueda> ]
[ GROUP BY <especificación de agrupación> ]
[ HAVING <condición de búsqueda> ]
[ ORDER BY <condición de orden> ]
```

Donde:

<lista de atributos> es una lista de los atributos cuyos valores serán recuperados por la consulta.

<tabla de referencia> es una lista de las relaciones necesarias para procesar la consulta.

<condición de búsqueda> es una expresión condicional (booleana) que identifica las tuplas que la consulta recuperará.



Las cláusulas FROM, WHERE, GROUP BY y HAVING hacen referencia como expresiones de tabla. Esta porción de la instrucción SELECT siempre se evalúa en el orden indicado en la sintaxis. El resultado de esa evaluación es una tabla virtual que se utiliza en la evaluación posterior.

En otras palabras, los resultados de la primera cláusula evaluada se utilizan en la siguiente cláusula. Los resultados de esa cláusula se utilizan en la siguiente cláusula, hasta que cada cláusula en la expresión de la tabla se evalúa. Por ejemplo, la primera cláusula a evaluar en la instrucción SELECT es la cláusula FROM. Ya que esta cláusula es necesaria, siempre es la primera cláusula evaluada. Los resultados de la cláusula FROM se utilizan en la cláusula WHERE, si se especifica la cláusula WHERE. Si no se especifica la cláusula, entonces los resultados de la cláusula FROM se utilizan en la siguiente cláusula especificada, ya sea la cláusula GROUP BY o la cláusula HAVING. Una vez que se evalúa la cláusula final en la expresión de tabla, los resultados se utilizan en la cláusula SELECT. Después de evaluar la cláusula SELECT, se evalúa la cláusula ORDER BY. [2]

Nombres de atributos ambiguos, alias y variables de tupla

En SQL el mismo nombre se puede utilizar para dos o más atributos, siempre y cuando los atributos se encuentren en relaciones diferentes. Si es el caso, y una consulta se refiere a dos o más atributos que tienen el mismo nombre, se debe calificar el nombre del atributo con el nombre de la relación a fin de evitar la ambigüedad. Esto se consigue colocando como prefijo el nombre de la relación (tabla) al nombre del atributo, y separando los dos nombres con un punto.[1]

Ejemplo:

```
SELECT Nombre, EMPLEADO. Nombre, Dirección
FROM EMPLEADO, DEPARTAMENTO
WHERE DEPARTAMENTO.Nombre='Investígación'
AND DEPARTAMENTO.NumeroDpto = EMPLEADO.NumeroDpto;
```

La ambigüedad también aparece en el caso de las consultas que se refieren dos veces a la misma relación, como en el siguiente ejemplo:

Ejemplo:

```
SELECT E.Nombre, E.Apellido, S.Nombre, S.Apellido
FROM EMPLEADO AS E, EMPLEADO AS S
WHERE E.SuperDni = S.Dni;
```

En este caso, nos permite declarar nombres de relación alternativos, E y S, denominados alias o variables de tupla, para la relación EMPLEADO. Un alias puede seguir a la palabra clave AS, como se muestra en el ejemplo, o puede seguir directamente al nombre de la relación (por ejemplo, escribiendo EMPLEADO E, EMPLEADO S en la cláusula FROM.

LIKE

El LIKE se puede utilizar para la comparación de patrones. Las cadenas parciales se especifican mediante dos caracteres reservados [1]:

- **%** sustituye una cantidad arbitraria de caracteres (de cero o más caracteres),
- _ (guión de subrayado) reemplaza un solo carácter.

Por ejemplo, considere la siguiente consulta:

Recuperar todos los empleados cuya dirección se encuentra en Caracas.

```
SELECT Nombre, Apellido
FROM EMPLEADO
WHERE Dirección LlKE '%Caracas%';
```

Predicado IN

El predicado IN permite determinar si los valores en la columna especificada de una tabla están contenidos en una lista definida o contenidos dentro de otra tabla. En el primer caso, se debe especificar el nombre de la columna, la palabra clave IN y una lista de valores que son comparados a los valores en la columna especificada. En el segundo caso, se debe especificar el nombre de la columna, la palabra clave IN y una subconsulta, que hace referencia a la segunda tabla. En cada caso, si el valor de la columna coincide con uno de los valores en la lista o en los resultados de la subconsulta, el predicado se evalúa como verdadero y la fila es arrojada en los resultados de la consulta.

Ejemplos:

```
SELECT NOMBRE_CD, EN_EXISTENCIA FROM INVENTARIO_DISCO_COMPACTO
WHERE EN_EXISTENCIA IN ( 12, 22, 32, 42 );
```

```
SELECT TITULO, ARTISTA
FROM ARTISTAS_DISCO_COMPACTO
WHERE TITULO IN
      ( SELECT NOMBRE_CD
        FROM INVENTARIO_DISCO_COMPACTO
        WHERE EN_EXISTENCIA > 10 );
```

Predicado NULL

El predicado NULL resulta muy simple de implementar. Utilizado en conjunción con la palabra clave IS, el predicado se agrega a una cláusula WHERE de la misma forma que cualquier otro predicado, y se aplica sólo a los valores nulos que pudieran existir en la columna que se está consultando.

Ejemplo:

```
SELECT *
FROM BIO_ARTISTAS
WHERE LUGAR_DE_NACIMIENTO IS NULL;
```

Predicado BETWEEN

El predicado BETWEEN se utiliza en conjunción con la palabra clave AND para identificar un rango de valores que pueden ser incluidos como una condición de búsqueda en la cláusula WHERE. Los valores en la columna identificada deben entrar en ese rango para poder evaluarse como verdaderos. Cuando se utiliza la cláusula BETWEEN, se debe especificar la columna aplicable, el valor más bajo del rango y el valor más alto del rango.

Ejemplo:

```
SELECT TITULO_CD, PRECIO_MENUDEO
FROM CDS_A_LA_MANO
WHERE PRECIO_MENUDEO BETWEEN 14 AND 16;
```

Resumen de Operadores

Operador	Operación	Ejemplo
=	Igualdad	SELECT * FROM EMPLEADO WHERE NDEPTO = 100;
!= , <> , ^=	Desigualdad	SELECT * FROM EMPLEADO WHERE NDEPTO != 100;
<	Menor que	SELECT * FROM EMPLEADO WHERE NDEPTO < 200;
>	Mayor Que	SELECT * FROM EMPLEADO WHERE NDEPTO > 200;
<=	Menor o igual que	SELECT * FROM EMPLEADO WHERE NDEPTO <= 100;
>=	Mayor o igual que	SELECT * FROM EMPLEADO WHERE NDEPTO >= 100;
IN	Igual a cualquiera de los miembros entre paréntesis	SELECT * FROM EMPLEADO WHERE NDEPTO in(100,300);
NOT IN	Distinto a cualquiera de los miembros entre paréntesis	SELECT * FROM EMPLEADO WHERE NDEPTO NOT IN(200);
BETWEEN	Contenido en el Rango	SELECT * FROM EMPLEADO WHERE NDEPTO BETWEEN 100 AND 300;
NOT BETWEEN	Fuera del Rango	SELECT * FROM EMPLEADO WHERE NDEPTO NOT BETWEEN 100 AND 300;
LIKE '_abc%'	Contiene la cadena 'abc' a parir del segundo carácter y luego cualquier cadena de caracteres	SELECT * FROM EMPLEADO WHERE nombre LIKE 'Ma%'
IS NULL	Compara el valor de un campo con el valor "NULL"	SELECT * FROM EMPLEADO WHERE sueldo IS NULL;
IS NOT NULL	Compara el valor de un campo sea diferente de "NULL"	SELECT * FROM EMPLEADO WHERE fnac IS NOT NULL;

Operador	Operación	Ejemplo
+	Suma	SELECT nombre,sueldo + comision FROM empleado WHERE oficio = 'VENDEDOR';
-	Resta	SELECT nombre, sueldo - descuento FROM empleado WHERE oficio = 'VENDEDOR';
*	Producto	SELECT nombre, sueldo *12 FROM empleado;
/	División	SELECT nombre, sueldo/31 FROM empleado;

GROUP BY

La cláusula GROUP BY tiene una función muy diferente de la cláusula WHERE. Como su nombre lo indica, la cláusula GROUP BY se utiliza para agrupar tipos de información con el fin de resumir datos relacionados. La cláusula GROUP BY se puede incluir en la instrucción SELECT aun si la cláusula WHERE se utiliza o no. Esta cláusula especifica los atributos de agrupamiento, que también deben aparecer en la cláusula SELECT, por lo

que el valor resultante de aplicar la función de agregación a un grupo de tuplas aparece junto con el valor de los atributos de agrupamiento [1].

Ejemplo:

```
SELECT NumProyecto, NombreProyecto, COUNT(*)
FROM PROYECTO, TRABAJA_EN
WHERE NumProyecto = NumProy
GROUP BY NumProyecto, NombreProyecto;
```

La cláusula HAVING es similar a la cláusula WHERE ya que define una condición de búsqueda. Sin embargo, a diferencia de la cláusula WHERE, la cláusula HAVING se refiere a grupos, no a filas individuales; puede aparecer en combinación con una cláusula GROUP BY. HAVING proporciona una condición en el grupo de tuplas asociado a cada valor de los atributos de agrupamiento. En el resultado de la consulta sólo aparecen los grupos que satisfacen la condición, La cláusula HAVING se aplica a los resultados después de haberse agrupado (en la cláusula GROUP BY)

Ejemplo:

```
SELECT PRECIO, CATEGORIA, SUM(A_LA_MANO) AS TOTAL_ A_LA_MANO
FROM EXISTENCIA_DISCO_COMPACTO
GROUP BY PRECIO, CATEGORIA
HAVING SUM(A_LA_MANO) > 10;
```

ORDER BY

La cláusula ORDER BY, cuando se usa en la instrucción SELECT, es la última cláusula procesada. La cláusula ORDER BY toma la salida de la cláusula SELECT y ordena los resultados de la consulta de acuerdo con las especificaciones dentro de la cláusula ORDER BY. La cláusula no agrupa las filas, como se agrupan por la cláusula GROUP BY, ni filtra las filas, como se filtran por la cláusula WHERE o la cláusula HAVING. Sin embargo, se puede especificar si las filas se organizan en un orden ascendente (utilizando la palabra clave ASC) o en orden descendente (usando la palabra clave DESC).

Para utilizar la cláusula ORDER BY, simplemente se especifica una o más columnas y las palabras clave opcionales ASC o DESC (una por columna). Si no se especifica la palabra

clave, se toma ASC. Las filas se organizan de acuerdo con la columna que especifique. Si se define más de una columna en la cláusula ORDER BY, las filas se organizan en el orden en que las columnas se especifican.

Ejemplo:

```
SELECT *
FROM EXISTENCIA_DISCO_COMPACTO
WHERE PRECIO < 16.00
ORDER BY PRECIO, A_LA_MANO DESC
```

Resumen de la sintaxis SQL [1]

```
CREATE TABLE <nombre de tabla> ( <nombre de columna>< tipo de columna>[<restricción de atributo> ]
                                { , <nombre de columna><tipo de columna> [ <restrición de atributo> ] }
                                [ <restricción de tabla> { , <restricción de tabla> } ] )
```

```
DROP TABLE <nombre de tabla>
```

```
ALTER TABLE <nombre de tabla> ADD <nombre de columna><tipo de columna>
```

```
SELECT [ DISTINCT ] <lista de atributos>
FROM ( <nombre de tabla> { <alias> } | <tabla concatenada> )
        { , ( <nombre de tabla> { <alias> } | <tabla concatenada> ) }
[ WHERE <condición>]
[ GROUP BY <atributos de agrupamiento>[ HAVING <condición de selección de grupo> ] ]
[ ORDER BY <nombre de columna> [ <orden>] { , <nombre de columna> [ <orden> ] } ]
```

```
<attribute list> ::= ( * | ( <nombre de columna> | <función> ( ( [ DISTINCT ] <nombre de columna> | * * ) ) )
                    { , ( <nombre de columna> | <función> ( ( [ DISTINCT] <nombre de columna> | * * ) ) } ) )
```

```
<atributos de agrupamiento> ::= <nombre de columna> { , <nombre de columna> }
```

```
<orden> ::= ( ASC | DESC )
```

```
INSERT INTO <nombre de tabla> [ ( <nombre de columna> { , <nombre de columna> } ) ]
( VALUES ( <valor constante> , { <valor constante> } ) { , ( <valor constante> { , <valor constante> } ) }
| <sentencia de selección>)
```

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

```
UPDATE <nombre de tabla>
SET <nombre de columna>= <expresión valor> { , <nombre de columna> = <expresión valor> }
[ WHERE <condición de selección>]
```

```
CREATE [ UNIQUE] INDEX <nombre de índice>
ON <nombre de tabla>( <nombre de columna>[ <orden> ] { , <nombre de columna> [ <orden> ] } )
[ CLUSTER ]
```

```
DROP INDEX <nombre de índice>
```

```
CREATE VIEW <nombre de vista> [ ( <nombre de columna> { , <nombre de columna>>} ) ]
AS <sentencia de selección>
```

```
DROP VIEW <nombre de vista>
```

Capítulo 5

Ejercicios

"Una de las funciones principales de cualquier base de datos es la capacidad de manejar los datos almacenados dentro de sus tablas." [2]

Ejercicio 1 - Viajeros

Dado el siguiente modelo relacional de una base de datos de una agencia de viajes

LUGAR (L_CODIGO, L_NOMBRE, L_TIPO, L_CLIMA, L_TOTAL_HABITANTES,L_DESCRIPCION, ***L_FK_LUGAR***)

VIAJE (V_CODIGO, ***V_FK_PASAJERO***, ***V_FK_LUGAR***, V_FECHA)

PASAJERO (P_CODIGO, P_NOMBRE, P_APELLIDO, P_PERFIL, P_FECHA_NACIMIENTO, P_NUM_TELEFONICO, ***P_FK_LUGAR_PAIS_RESIDENCIA***)

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita.

Genere los siguientes reportes:

1. Listado de todos los lugares para viajar registrados en la base de datos.

```
SELECT *
FROM lugar;
```

2. Listado con el nombre y apellido de todos los pasajeros.

```
SELECT p_nombre, p_apellido
FROM pasajero;
```

3. Seleccionar el nombre y el clima de todos los lugares.

```
SELECT l_nombre, l_clima
FROM lugar;
```

4. Mostrar el nombre, total de habitantes de todos los lugares, además, suponer un incremento de la población de cada lugar en 100000 habitantes.

```
SELECT l_nombre, l_total_habitantes,
       l_total_habitantes + 100000 AS incremento_total_habitantes
FROM lugar;
```

5. Mostrar el nombre, tipo de lugar y el número de habitantes de cada lugar si su población se doblara. Nombrar la columna como "Doble_de_habitantes".

```
SELECT l_nombre, l_tipo, l_total_habitantes,
       L_total_habitantes * 2 AS Doble_de_habitantes
```

```
FROM lugar;
```

6. Listar los diferentes climas de los lugares.

```
SELECT DISTINCT clima
FROM lugar;
```

7. Listar los diferentes perfiles de los pasajeros.

```
SELECT DISTINCT p_perfil
FROM pasajero;
```

8. Listado de los diferentes apellidos y perfiles de los pasajeros.

```
SELECT DISTINCT p_apellido, p_perfil
FROM pasajero;
```

9. Seleccionar el nombre y el apellido de todos los pasajeros de clase turista (perfil).

```
SELECT p_nombre, p_apellido
FROM pasajero
WHERE p_perfil = 'TURISTA';
```

10. Listado de todos los lugares con clima tropical

```
SELECT l_nombre
FROM lugar
WHERE L_clima = 'TROPICAL';
```

11. Listar todos los países con un total de habitantes que supere los 10 millones.

```
SELECT l_nombre, l_total_habitantes
FROM lugar
WHERE l_tipo = 'PAIS' AND l_total_habitantes > 10000000;
```

12. Mostrar todos los pasajeros nacidos entre el 1970 y 1980.

```
SELECT p_nombre, p_apellido, p_fecha_nacimiento
FROM pasajero
WHERE p_fecha_nacimiento BETWEEN '01-01-1970' AND '31-12-1980';
```

13. Mostrar nombre y fecha de nacimiento de los pasajeros de clase business y primera clase. Etiquetar la fecha de nacimiento como "Nacido_el".

```
SELECT p_nombre, p_fecha_nacimiento AS Nacido_el
FROM pasajero
WHERE p_perfil IN ('BUSINESS','PRIMERA_CLASE');
```

14. Mostrar todos los lugares que no tengan clima Mediterráneo.

```
SELECT l_nombre, l_clima
FROM lugar
WHERE l_clima <> 'MEDITERRANEO';
```

15. Listado de todos los pasajeros cuyo nombre empiece por la letra M.

```
SELECT l_nombre, l_apellido
FROM pasajero WHERE l_nombre LIKE 'M%';
```

16. Listado de todos los lugares cuyo nombre tenga una letra A.

```
SELECT l_nombre
FROM lugar
WHERE l_nombre LIKE '%A%';
```

17. Listar los diferentes apellidos de los pasajeros que empiecen por la letra M y terminen en EZ.

```
SELECT DISTINCT p_apellido
FROM pasajero
WHERE p_apellido LIKE 'M%EZ';
```

18. Mostrar todos los lugares que tengan clima registrado.

```
SELECT l_nombre
FROM lugar
WHERE l_clima IS NOT NULL;
```

19. Mostrar los pasajeros que no tengan perfil de pasajero registrado.

```
SELECT p_nombre, p_apellido
FROM pasajero
WHERE p_perfil IS NULL;
```

20. Listado de todos los pasajeros que contengan la letra “Y” en su nombre y que hayan nacido entre el año 1975 y el 1985.

```
SELECT p_nombre, p_apellido
FROM pasajero
WHERE p_nombre LIKE '%Y%'
AND p_fecha_nacimiento BETWEEN '01-01-1975' AND '31-12-1985';
```

21. Listado viajeros que tienen el perfil LOW_COST y tengan apellido Ruiz.

```
SELECT p_nombre, p_apellido
FROM pasajero
WHERE p_apellido = 'RUIZ'
AND p_perfil = 'LOW_COST';
```

22. Listado de los lugares que no tengan informado el número total de habitantes, su nombre empiece por C, con clima desconocido o Mediterráneo.

```
SELECT l_nombre, l_clima
FROM lugar
WHERE l_total_habitantes IS NULL
AND L_nombre LIKE 'C%'
AND (l_clima IS NULL OR l_clima = 'MEDITERRANEO');
```

23. Listado con los lugares de clima desconocido y que tengan la palabra ciudad en su nombre o que el total de sus habitantes sea inferior a 5 millones.

```
SELECT l_nombre, l_clima, l_total_habitantes
FROM lugar
WHERE l_clima IS NULL
AND (l_nombre LIKE '%ciudad%' OR l_total_habitantes < 50000000);
```

24. Listar los pasajeros nacidos después de 1970 que viajen con el perfil business class y que tenga una G en su nombre.

```
SELECT p_nombre, p_apellido
FROM pasajero
WHERE p_nombre LIKE '%G%'
AND fecha_nacimiento > '31-12-1970'
AND p_perfil = 'BUSINESS CLASS';
```

25. Mostrar los países que no tengan entre 50 y 100 millones de habitantes y estén en Europa o Asia y además tengan una M en su nombre.

```
SELECT p.l_nombre AS pais, c.l_nombre AS continente
```

```
FROM lugar p, lugar c
WHERE p.l_total_habitantes NOT BETWEEN 50000000 AND 100000000
AND p.l_tipo = 'PAIS'
AND p.l_nombre LIKE '%M%'
AND p.l_fk_lugar = c.l_codigo
AND c.l_tipo = 'CONTINENTE'
AND c.l_nombre IN ('EUROPA','ASIA');
```

26. Listado de países ordenados por su total de habitantes de mayor a menor.

```
SELECT l_nombre, l_total_habitantes
FROM lugar
WHERE l_tipo = 'PAIS'
ORDER BY l_total_habitantes DESC;
```

27. Listar los pasajeros por orden alfabético de su apellido, mostrarlos de la forma Apellido, Nombre y etiquetar la columna como Nombre y Apellido.

```
SELECT p_apellido || ', ' || p_nombre AS 'Nombre y Apellido'
FROM pasajero
ORDER BY p_apellido;
```

28. Mostrar los paises con clima Tropical ordenados alfabéticamente.

```
SELECT l_nombre, l_clima
FROM lugar WHERE l_clima = 'Tropical' ORDER BY l_nombre;
```

29. Mostrar los países de Europa ordenándolos por su número de habitantes de menor a mayor.

```
SELECT l_nombre, l_total_habitantes
FROM lugar
WHERE l_tipo = 'CONTINENTE'
ORDER BY l_total_habitantes ASC;
```

30. Mostrar los viajeros ordenados por su fecha de nacimiento (de más joven a menos) y por su apellido de forma inversa.

```
SELECT p_apellido, p_nombre, p_fecha_nacimiento
FROM pasajero
ORDER BY p_fecha_nacimiento, p_apellido DESC;
```

31. Mostrar los diferentes continentes ordenados alfabéticamente.

```
SELECT DISTINCT l_nombre
FROM lugar
WHERE l_tipo = 'CONTINENTE'
ORDER BY l_nombre ASC;
```

32. Seleccionar todos aquellos pasajeros que no usen clase business ordenados por su número de teléfono y apellido.

```
SELECT p_nombre, p_apellido, p_num_telefonico
FROM pasajero
WHERE p_perfil <> 'BUSINESS'
ORDER BY p_num_telefonico, p_apellido;
```

33. Hacer un listado con los países que no pertenezcan a Europa u Oceanía ordenados alfabéticamente por su continente y su total de habitantes de mayor a menor.

```
SELECT p.l_nombre pais , c.l_nombre AS Continente,
       p.l_total_habitantes AS total_habitantes
FROM lugar p , lugar AS c
WHERE p.l_tipo = 'PAIS'
AND p.l_fk_lugar = c.l_codigo
AND c.l_tipo = 'CONTINENTE'
AND c.l_nombre NOT IN ('EUROPA','OCEANIA')
ORDER BY c.l_nombre  ASC, p.l_total_habitantes DESC;
```

34. Mostrar todos los pasajeros apellidados Fernandez, que no vivan en Colombia, ordenados por su perfil de pasajero y su nombre.

```
SELECT p.p_apellido, p.p_nombre, p.p_perfil
FROM pasajero as p , lugar as l
WHERE p_apellido = 'FERNANDEZ'
AND p.p_fk_lugar_pais_residencia = l.l_codigo
AND l.l_nombre <> 'COLOMBIA'
AND l.l_tipo =  'PAIS'
ORDER BY p.p_perfil, p.p_nombre;
```

35. Listar todos los lugares con más de 10000 habitantes, de clima conocido y con las palabras "UBICADA" y "SUR" ordenados por su nombre y su número de habitantes, de menor a mayor.

```
SELECT l_nombre, l_clima ,l_total_habitantes
FROM lugar
WHERE l_total_habitantes > 10000
AND l_clima IS NOT NULL
AND l_descripcion LIKE '%UBICADA%SUR%'
ORDER BY l_nombre, l_total_habitantes;
```

Ejercicio 2 - Hospitales [8]

Dado el DDL de la base de datos de hospitales:

```
CREATE TABLE HOSPITALES(
        HOSPITAL_COD        NUMBER(2),
        NOMBRE              VARCHAR(15),
        DIRECCION           VARCHAR(20),
        TELEFONO            CHAR(8),
        NUM_CAMA            NUMBER(3),
        CONSTRAINT HOSPITAL_PK PRIMARY KEY (HOSPITAL_COD));

CREATE TABLE SALA(
        HOSPITAL_COD    NUMBER(2)       NOT NULL,
        SALA_COD        NUMBER(2),
        NOMBRE          VARCHAR2(20),
        NUM_CAMA        NUMBER(3),
        CONSTRAINT SALA_PK PRIMARY KEY (SALA_COD),
        CONSTRAINT HOSPITAL_SALA_FK FOREIGN KEY (HOSPITAL_COD) REFERENCES HOSPITAL (HOSPITAL_COD));

CREATE TABLE PLANTILLA(
        HOSPITAL_ COD       NUMBER(2)           NOT NULL,
        SALA_COD            NUMBER(2)           NOT NULL,
        EMPLEADO_NO         NUMBER(4),
        APELLIDO            VARCHAR(15),
        FUNCION             CHAR(10),
        TURNO               CHAR (1) ,
        SALARIO             NUMBER(10),
        CONSTRAINT EMPLEADO_PK PRIMARY KEY (EMPLEADO_NO),
        CONSTRAINT HOSPITAL_FK FOREIGN KEY (HOSPITAL_COD) REFERENCES HOSPITAL (HOSPITAL_COD),
        CONSTRAINT SALA_FK FOREIGN KEY (SALA_COD) REFERENCES SALA (SALA_COD),
        CONSTRAINT TURNO_CH CHECK (TURNO IN ('T','M','N')),
        CONSTRAINT SALARIO_CH CHECK (SALARIO > 0));

CREATE TABLE ENFERMO(
        INSCRIPCION         NUMBER(5),
        APELLIDO        VARCHAR2(15),
        DIRECCION           VARCHAR2(20),
        FECHA-NAC           DATE,
        S                   VARCHAR2(1),
        NSS                 NUMBER(9),
        CONSTRAINT ENFERMO_PK PRIMARY KEY (INSCRIPCION));

CREATE TABLE OCUPACION(
        INSCRIPCION         NUMBER(5)       PRIMARY KEY,
        HOSPITAL_COD    NUMBER(2)PRIMARY KEY,
        SALA_COD        NUMBER(2)PRIMARY KEY,
        CAMA                NUMBER(4),
        CONSTRAINT HOSPITAL_OCUP_FK FOREIGN KEY (HOSPITAL_COD) REFERENCES HOSPITAL (HOSPITAL_COD),
        CONSTRAINT SALA_OCUP_FK FOREIGN KEY (SALA_COD) REFERENCES SALA (SALA_COD));
```

Se pide:

1. Listado de todos los miembros del personal cuyo nombre empiece por la letra 'A'.

```
SELECT apellido
FROM plantilla WHERE apellido LIKE 'A%';
```

2. ¿Quienes son las enfermeras y enfermeros que trabajan en turnos de Tarde o Mañana?

```
SELECT apellido
FROM plantilla
WHERE funcion IN ('ENFERMERO','ENFERMERA')
AND turno IN ('T','M');
```

3. Listado de las enfermeras que ganan entre 2.000.000 y 2.500.000 Bs.

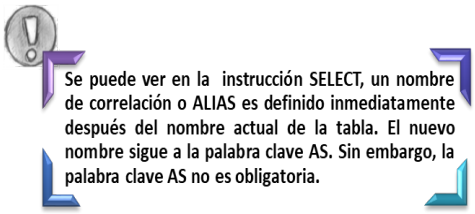
```
SELECT apellido, salario
FROM plantilla
WHERE salario BETWEEN 2000000 AND 2500000
AND funcion = 'ENFERMERA';
```

4. Mostrar, para todos los hospitales, el código de hospital, el nombre completo del hospital y su nombre . Ordenar la recuperación por el nombre.

```
SELECT nombre, hospital_cod
FROM hospital
ORDER BY 1;
```

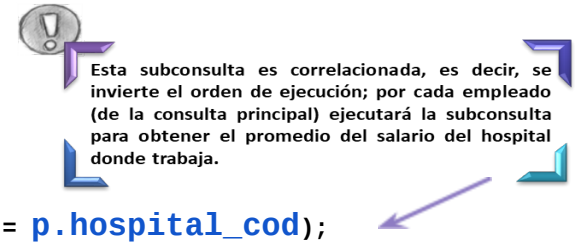
5. Se desea conocer el máximo salario que existe en cada sala de cada hospital.

```
SELECT h.nombre, s.nombre, MAX(salario)
FROM sala AS s, plantilla p, hospital AS h
WHERE h.hospital_cod = p.hospital_cod
AND p.sala_cod = s.sala_cod
GROUP BY h.nombre, s.nombre;
```



6. Recuperar el número (empleado_no) y nombre de las personas que perciban un salario mayor que la media de **su** hospital.

```
SELECT apellido, empleado_no
FROM plantilla p
WHERE salario > (SELECT AVG(salario)
                 FROM plantilla p2
                 WHERE p2.hospital_cod = p.hospital_cod);
```



7. Insertar en la tabla Plantilla al enfermero Garcia J. con un sueldo de 3.000.000 Bs, y número de empleado 1234. Trabaja en el hospital 22, sala 2, turno matutino.

Solución 1:

```
INSERT INTO plantilla(hospital_cod, sala_cod, empleado_no, apellido,
                      funcion, turno, salario)
VALUES (22,2,1234,'Garcia J.','ENFERMERO', 'M',3000000);
```

Solución 2:

```
INSERT INTO plantilla
VALUES (22,2,1234,'Garcia J.','ENFERMERO','M',3000000);
```

8. Cambiar al paciente (tabla ENFERMO) número 74835 la dirección a MADRID 411**.**

```
UPDATE enfermo
SET direccion = 'MADRID 411'
WHERE inscripcion = 74835;
```

9. Actualizar todas las direcciones de la tabla ENFERMO a NULL.

```
UPDATE enfermo
SET direccion = NULL;
```

10. Igualar la dirección del paciente número 10995 al valor correspondiente en la columna almacenada para el paciente número 14024.

```
UPDATE enfermo
SET direccion = (SELECT direccion
                 FROM enfermo
                 WHERE inscripcion = 14024)
WHERE inscripcion = 10995;
```

11. En todos los hospitales del país se ha recibido un aumento del presupuesto, por lo que se incrementará el número de camas disponibles en un 10%.

```
UPDATE hospitales
SET num_cama = num_cama + (num_cama * 0.1);
```

12. Defina una vista porque se va a realizar un programa de consulta de la información sobre enfermos. Los datos a mostrar serán sus apellidos, dirección, fecha de nacimiento y hospital en el que se encuentran.

```
CREATE VIEW informacion_de_enfermos
       AS SELECT e.apellido, e.direccion, e.fecha_nac, h.nombre
          FROM enfermo e, hospital h, ocupación o
          WHERE e.inscripcion = o.inscripcion
          AND o.hospital_cod = h.hospital-cod;
```

Ejercicio 3 - Empresa distribuidora de alimentos [8]

Dado el DDL de la base de datos de la empresa. Se pide:

```
CREATE TABLE DEPT2(
        DEPT_NO  NUMBER(2),
        DNOMBRE VARCHAR2(14),
        LOC          VARCHAR2(14),
        CONSTRAINT DPTO_PK PRIMARY KEY (DEPT_NO));

CREATE TABLE EMP(
        EMP_NO   NUMBER(4),
        APELLIDO  VARCHAR2(10)                  NOT NULL,
        OFICIO          VARCHAR2(10)                  NOT NULL,
        DIR             NUMBER(4),
        FECHA_ALTA      DATE,
        SALARIO   NUMBER(10)                  NOT NULL,
        COMISIÓN        NUMBER(10),
        DEPT_NO  NUMBER(2),
        CONSTRAINT EMPLEADO_PK PRIMARY KEY (EMP_NO),
        CONSTRAINT DEPT2_FK FOREIGN KEY (DEPT_NO) REFERENCES DEPT2(DEPT_NO));
```

1. Se quiere un listado con el siguiente formato: El departamento de CONTABILIDAD está en CARACAS.

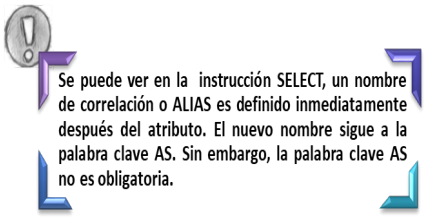
```
SELECT 'El departamento de' || dnombre || 'está en '||localizacion
FROM dept2;
```

2. Para cada empleado cuyo apellido contenga una "N", se quiere el oficio y apellido. La salida debe estar ordenada por apellido ascendentemente.

```
SELECT oficio , apellido
FROM emp
WHERE apellido LIKE'%N%'
ORDER BY apellido;
```

3. Listado de empleados con su salario total (salario más comisión). La salida debe estar ordenada por el salario y el apellido descendentemente.

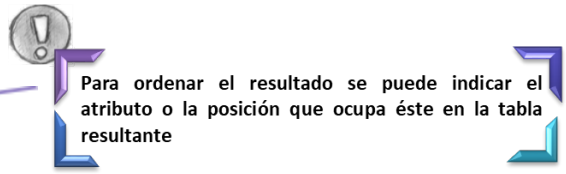
```
SELECT apellido,
       (salario + comision) AS salario_total
FROM emp
ORDER BY  2 DESC, 1;
```



4. Para cada empleado que no tenga comisión o cuya comisión sea mayor que el 15% de su salario, se pide el salario total que tiene. Este será: salario más su comisión más el 15%

del salario. La salida deberá estar ordenada por el oficio y por el salario que le queda descendentemente.

```
SELECT apellido, oficio,
       salario + comision + (salario * 0.15) AS salario_total
FROM emp
WHERE comisión IS NULL
OR comisión > salario * 0.15
ORDER BY oficio, 3
DESC ;
```



13. Listado con el personal de alta, que indique el apellido, oficio y fecha de alta del personal del departamento número 20 que ganan más de 150.000 Bs Mensuales.

```
SELECT apellido, oficio,fecha_alt Alta
FROM emp
WHERE dept_no = 20
AND salario > 150000;
```

14. Se pide el apellido, oficio y la fecha en que han sido dados de alta los empleados de la empresa, pero solo de aquellos cuyo fecha de alta sea posterior al año 2018. Ordenado por oficio.

```
SELECT apellido, oficio, fecha_alt AS Alta
FROM emp
WHERE fecha_alt > '31-12-2018'
ORDER BY  2;
```

15. Encontrar el salario medio de los empleados que son Analistas.

```
SELECT AVG(salario) AS SALARIO_MEDIO
FROM emp
WHERE oficio = 'ANALISTA';
```

16. Encontrar el salario más alto y el salario más bajo de la tabla de empleados, así como la diferencia entre ambos.

```
SELECT MAX(salario) maximo, MIN(salario) mínimo,
       MAX(salario) - MIN(salario) diferencia
FROM emp;
```

17. Calcular el número de oficios diferentes que hay en total, en los departamentos 10 y 20 de la empresa.

```
SELECT COUNT(DISTINCT oficio) tareas
FROM emp
WHERE dept_no IN (10,20);
```

18. Calcular el número de personas que realizan cada oficio en cada departamento.

```
SELECT dept_no, oficio, COUNT(*)
FROM emp
GROUP BY dept_no, oficio;
```

19. Se desea saber el salario medio total (salario más comisión) anual de los vendedores de nuestra empresa.

```
SELECT oficio, AVG(salario + comision) 'SALARIO MEDIO ANUAL'
FROM emp
WHERE oficio = 'VENDEDOR'
GROUP BY oficio;
```

20. Se desea saber el salario medio total (salario más comisión) anual, tanto de los empleados como de los vendedores de nuestra empresa.

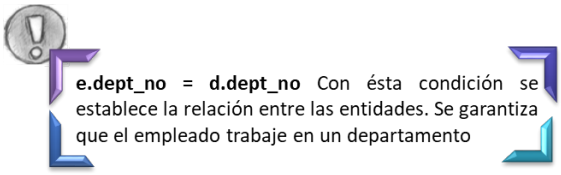
```
SELECT oficio, AVG(salario + comision) 'SALARIO MEDIO ANUAL'
FROM emp
WHERE oficio IN ('VENDEDOR', 'EMPLEADO')
GROUP BY oficio;
```

21. Se desea saber para cada departamento y en cada oficio, el máximo salario y la suma total de salarios, pero solo de aquellos departamentos y oficios cuya suma salarial supere o sea igual que el 50% de su máximo salario. En el muestreo, solo se debe estudiar a aquellos empleados que no tienen comisión o la tengan menor que el 25% de su salario.

```
SELECT dept_no, oficio, SUM(salario) suma, MAX(salario) maximo
FROM emp
WHERE comisión IS NULL
OR comisión = 0
OR comisión < (0.25 * salario)
GROUP BY dept_no, oficio
HAVING SUM(salario) >= 0.5 * MAX(salario);
```

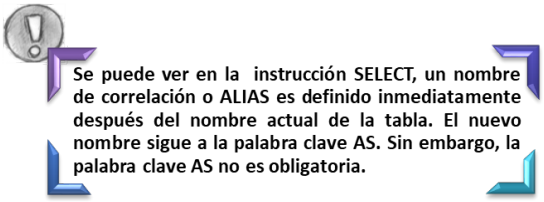
22. Listar el apellido de cada empleado, su oficio, su número de departamento y el nombre del departamento donde trabajan.

```
SELECT e.apellido, e.oficio,
       e.dept_no, d.dnombre
FROM emp e, dept2 d
WHERE e.dept_no = d.dept_no;
```



23. Seleccionar los nombres, profesiones y localidades de los departamentos donde trabajan los Analistas.

```
SELECT e.apellido, e.oficio, d.loc
FROM emp AS e, dept2 d
WHERE e.dept_no = d.dept_no
AND oficio = 'ANALISTA';
```



24. Se desea conocer el nombre y oficio de todos aquellos empleados que trabajan en Madrid. La salida deberá estar ordenada por el oficio.

```
SELECT e.apellido, e.oficio
FROM emp e, dept2 d
WHERE d.loc = 'MADRID'
AND e.dept_no = d.dept_no
ORDER BY e.oficio;
```

25. Se desea conocer cuántos empleados existen en cada departamento. Devolviendo una salida ordenada por el número de empleados descendentemente.

```
SELECT e.dept_no num_dep, d.dnombre departamento, COUNT(*) num_empleados
FROM emp e, dept2 d
WHERE e.dept_no = d.dept_no
GROUP BY e.dept_no, d.dnombre
ORDER BY 3 DESC;
```

26. Se desea conocer, tanto para el departamento de VENTAS, como para el de CONTABILIDAD, su máximo, su mínimo y su media salarial, así como el número de empleados en cada departamento. La salida deberá estar ordenada por el nombre del departamento.

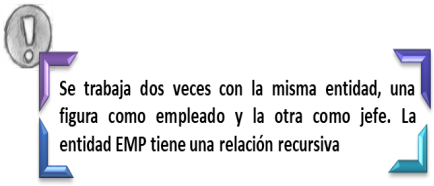
```
SELECT d.dnombre, MAX(e.salario) maximo, MIN(e.salario) mínimo,
       AVG(e.salario) media, COUNT(*) n_empl
FROM emp e, dept2 d
WHERE d.dnombre IN ('VENTAS', 'CONTABILIDAD')
AND e.dept_no = d.dept_no
GROUP BY d.dnombre
ORDER BY d.dnombre;
```

27. El atributo DIR representa la relación entre un empleado y su jefe.

```
ALTER TABLE EMP ADD CONSTRAINT EMP_FK FOREIGN KEY (DIR)
REFERENCES EMP(EMP_NO);
```

28. Se desea un listado en el que se presente el número, nombre y oficio de cada empleado de la empresa que tiene jefe, y lo mismo de su jefe directo. La salida debe estar ordenada por el nombre del empleado.

```
SELECT e.emp_no empleado, e.apellido nombre, e.oficio oficio, e.dir jefe,
       e2.apellido AS nombre_jefe,
       E2.oficio AS oficio_jefe
FROM emp e, emp e2
WHERE e.dir = e2.emp_no
ORDER BY e.apellido;
```



29. Con el uso de subconsultas. Obtener el apellido, departamento y oficio de aquellos empleados que tengan un oficio, que esté en el departamento 20 y que no sea ninguno de los oficios que está en el departamento de VENTAS.

```
SELECT apellido, dept_no, oficio
FROM emp
WHERE oficio IN (SELECT oficio
                 FROM emp
                 WHERE dept_no = 20)
AND oficio NOT IN ( SELECT oficio
                    FROM emp e, dept2 d
                    WHERE e.dept_no = d.dept_no
                    AND d.dnombre = 'VENTAS');
```

30. Con el uso de subconsultas. Obtener el número de empleado, número de departamento y apellido de todos los empleados que trabajen en el departamento 20 o 30 y su salario sea mayor que dos veces el mínimo salario de la empresa. No queremos que el oficio sea PRESIDENTE.

```
SELECT emp_no, dept_no, apellido
FROM emp
WHERE dept_no  IN (20,30)
AND salario > (SELECT MIN(salario) * 2 FROM emp)
AND oficio <> 'PRESIDENTE';
```

31. Queremos conocer el apellido, oficio, salario y departamento en el que trabajan, de todos los individuos cuyo salario sea mayor que el mayor salario del departamento 30

```
SELECT apellido, oficio, salario, dept_no
FROM emp
WHERE salario > (SELECT MAX(salario)
                 FROM emp
                 WHERE dept_no = 30);
```

32. Presentar los nombres y oficios de todos los empleados del departamento 20, cuyo trabajo sea idéntico al de cualquiera de los empleados del departamento de VENTAS.

```
SELECT apellido, oficio
FROM emp
WHERE dept_no = 20
AND oficio IN (SELECT oficio
           FROM emp e, dept2 d
           WHERE dnombre = 'VENTAS' AND e.dept_no = d.dept_no);
```

33. Se desea obtener todos los empleados de los departamentos que no ganan ni el máximo ni el mínimo salarial de la empresa.

```
SELECT apellido, oficio
FROM emp
WHERE salario <> (SELECT MAX(salario) FROM emp)
AND salario <> (SELECT MIN(salario) FROM emp);
```

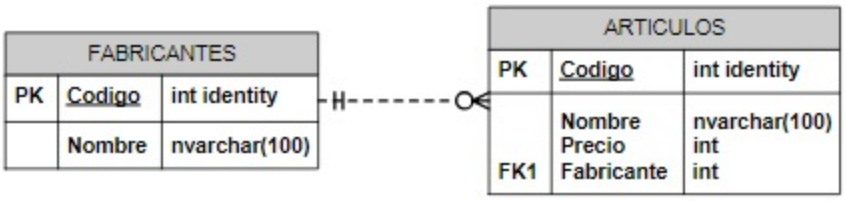
34. Se desea obtener, para cada departamento, su mínimo y su máximo salarial. Para ello, no se tendrá en cuenta a los empleados cuyo apellido, coincida con nombre del departamento en que trabajan. Asimismo, se tendrá en cuenta a aquellos departamentos cuya diferencia entre el máximo y el mínimo excede la media salarial de toda la empresa.

```
SELECT dept_no, MIN (salario) mínimo, MAX(salario) maximo
FROM emp e
WHERE apellido NOT IN
      (SELECT dnombre
       FROM dept2 d
       WHERE e.dept_no = d.dept_no)
GROUP BY dept_no
HAVING MAX(salario) - MIN(salario) > (SELECT AVG(salario) FROM emp);
```

Esta subconsulta es correlacionada, es decir, se invierte el orden de ejecución; por cada empleado (de la consulta principal) ejecutará la subconsulta para obtener el nombre del departamento en el que trabaja.

Ejercicio 4 - Tienda de informática [9]

Dado el siguiente Modelo Entidad Relación, se pide:



1. Obtener los nombres de los productos de la tienda.

```
SELECT Nombre
FROM ARTICULOS;
```

2. Obtener los nombres y los precios de los productos de la tienda.

```
SELECT Nombre, Precio
FROM ARTICULOS;
```

3. Obtener el nombre de los productos cuyo precio sea menor o igual a 200 Bs.

```
SELECT Nombre
FROM ARTICULOS
WHERE Precio > 200;
```

4. Obtener todos los datos de los artículos cuyo precio esté entre los 60 Bs y los 120 Bs (ambas cantidades incluidas).

   Solución 1:

   ```
   SELECT * FROM ARTICULOS
   WHERE Precio >= 60 AND Precio <= 120;
   ```

   Solución 2:

   ```
   SELECT * FROM ARTICULOS
   WHERE Precio BETWEEN 60 AND 120;
   ```

5. Obtener el nombre y el precio multiplicado por 166386, la columna debe presentarse como precio_total.

```
SELECT Nombre, Precio * 166.386 AS precio_total
FROM ARTICULOS;
```

6. Seleccionar el precio medio de todos los productos.

```
SELECT AVG(Precio)
FROM ARTICULOS;
```

7. Obtener el precio medio de los artıculos cuyo código de fabricante sea 2.

```
SELECT AVG(Precio)
FROM ARTICULOS
WHERE Fabricante=2;
```

8. Obtener el número de artículos cuyo precio sea mayor o igual a 180 Bs.

```
SELECT COUNT(*)
FROM ARTICULOS
WHERE Precio >= 180;
```

9. Obtener el nombre y precio de los artículos cuyo precio sea mayor o igual a 180 Bs y ordenarlos descendentemente por precio, y luego ascendentemente por nombre.

```
SELECT Nombre, Precio
FROM ARTICULOS
WHERE Precio >= 180
ORDER BY Precio DESC, Nombre;
```

10. Obtener un listado completo de artículos, incluyendo por cada artículo los datos del artículo y de su fabricante.

Solución 1 con inner join implicito:

```
SELECT *
FROM ARTICULOS, FABRICANTES
WHERE ARTICULOS.Fabricante = FABRICANTES.Codigo;
```

Solución 2 con inner join explicito:

```
SELECT *
FROM ARTICULOS INNER JOIN FABRICANTES
ON ARTICULOS.Fabricante = FABRICANTES.Codigo;
```

11. Obtener un listado de artículos, incluyendo el nombre del artículo, su precio, y el nombre de su fabricante.

    Solución 1 con inner join implicito:

```
SELECT ARTICULOS.Nombre, Precio, FABRICANTES.Nombre
FROM ARTICULOS, FABRICANTES
WHERE ARTICULOS.Fabricante = FABRICANTES.Codigo;
```

    Solución 2 con inner join explicito:

```
SELECT ARTICULOS.Nombre, Precio, FABRICANTES.Nombre
FROM ARTICULOS INNER JOIN FABRICANTES
ON ARTICULOS.Fabricante = FABRICANTES.Codigo;
```

12. Obtener el precio medio de los productos de cada fabricante, mostrando solo los códigos de fabricante.

```
SELECT AVG(Precio), Fabricante
FROM ARTICULOS
GROUP BY Fabricante;
```

13. Obtener el precio medio de los productos de cada fabricante, mostrando el nombre del fabricante.

    Solución 1 con inner join implicito:

```
SELECT AVG(Precio), FABRICANTES.Nombre
FROM ARTICULOS, FABRICANTES
WHERE ARTICULOS.Fabricante = FABRICANTES.Codigo
GROUP BY FABRICANTES.Nombre;
```

    Solución 2 con inner join explicito:

```
SELECT AVG(Precio), FABRICANTES.Nombre
FROM ARTICULOS INNER JOIN FABRICANTES
ON ARTICULOS.Fabricante = FABRICANTES.Codigo
GROUP BY FABRICANTES.Nombre;
```

14. Obtener los nombres de los fabricantes que ofrezcan productos cuyo precio medio sea mayor o igual a 150 Bs. Indicar el precio medio.

    Solución 1 con inner join implicito:

```
SELECT AVG(Precio), FABRICANTES.Nombre
FROM ARTICULOS, FABRICANTES
WHERE ARTICULOS.Fabricante = FABRICANTES.Codigo
GROUP BY FABRICANTES.Nombre
HAVING AVG(Precio) >= 150;
```

Solución 2 con inner join explicito:

```
SELECT AVG(Precio), FABRICANTES.Nombre
FROM ARTICULOS INNER JOIN FABRICANTES
ON ARTICULOS.Fabricante = FABRICANTES.Codigo
GROUP BY FABRICANTES.Nombre
HAVING AVG(Precio) >= 150;
```

15. Obtener el nombre y precio del artículo más barato.

```
SELECT Nombre, Precio
FROM ARTICULOS
WHERE Precio = (SELECT MIN(Precio) FROM ARTICULOS);
```

16. Obtener una lista con el nombre y precio de los artículos más caros de cada proveedor (incluyendo el nombre del proveedor).

Solución 1 con inner join implicito:

```
SELECT A.Nombre, A.Precio, F.Nombre
FROM ARTICULOS A, FABRICANTES F
WHERE A.Fabricante = F.Codigo
AND A.Precio = (SELECT MAX(A.Precio)
                FROM ARTICULOS A
                WHERE A.Fabricante = F.Codigo);
```

Solución 2 con inner join explicito:

```
SELECT A.Nombre, A.Precio, F.Nombre
FROM ARTICULOS A INNER JOIN FABRICANTES F
ON A.Fabricante = F.Codigo
AND A.Precio = (SELECT MAX(A.Precio)
                FROM ARTICULOS A
                WHERE A.Fabricante = F.Codigo);
```

17. Añadir un nuevo producto: Altavoces de 70 Bs (del fabricante 2).

```
INSERT INTO ARTICULOS( Nombre , Precio , Fabricante)
VALUES ( 'Altavoces' , 70 , 2 );
```

18. Cambiar el nombre del producto 8 a “Impresora Laser”.

```
UPDATE ARTICULOS
SET Nombre = 'Impresora Laser'
WHERE Codigo = 8;
```

19. Aplicar un descuento del 10 % a todos los productos.

```
UPDATE ARTICULOS
SET Precio = Precio * 0.9;
```

20. Aplicar un descuento de 10 Bs a todos los productos cuyo precio sea mayor o igual a 120.

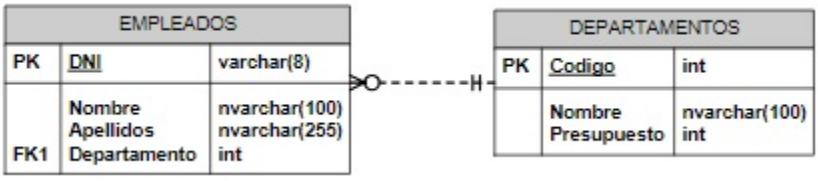
```
UPDATE ARTICULOS
SET Precio = Precio - 10
WHERE Precio >= 120;
```

21. Borrar el producto de código 6

```
DELETE FROM ARTICULOS
WHERE codigo = 6;
```

Ejercicio 5 - Empleados [9]

Dado el siguiente Modelo Entidad Relación, se pide:



1. Obtener los apellidos de los empleados.

```
SELECT Apellidos
FROM EMPLEADOS;
```

2. Obtener los apellidos de los empleados sin repeticiones

```
SELECT DISTINCT Apellidos
FROM EMPLEADOS;
```

3. Obtener todos los datos de los empleados que se apellidan ’LOPEZ’.

```
SELECT *
FROM EMPLEADOS
WHERE Apellidos = 'LOPEZ';
```

4. Obtener todos los datos de los empleados que se apellidan ’LOPEZ’ y los que se apellidan ’PEREZ’.

   Solución 1 :

```
SELECT *
FROM EMPLEADOS
WHERE Apellidos = 'LOPEZ' OR Apellidos = 'PEREZ';
```

   Solución 2 :

```
SELECT *
FROM EMPLEADOS
WHERE Apellidos IN ('LOPEZ' , 'PEREZ');
```

5. Obtener todos los datos de los empleados que trabajan para el departamento 14.

```
SELECT *
FROM EMPLEADOS
WHERE Departamento = 14;
```

6. Obtener todos los datos de los empleados que trabajan para el departamento 37 y para el departamento 77.

   Solución 1 :

```
SELECT *
FROM EMPLEADOS
WHERE Departamento = 37 OR Departamento = 77;
```

   Solución 2:

```
SELECT *
FROM EMPLEADOS
WHERE Departamento IN (37,77);
```

7. Obtener todos los datos de los empleados cuyo apellido comience por 'P'.

```
SELECT *
FROM EMPLEADOS
WHERE Apellidos LIKE 'P%';
```

8. Obtener el presupuesto total de todos los departamentos.

```
SELECT SUM(Presupuesto)
FROM DEPARTAMENTOS;
```

9. Obtener el número de empleados en cada departamento.

```
SELECT Departamento, COUNT(*)
FROM EMPLEADOS
GROUP BY Departamento;
```

10. Obtener un listado completo de empleados, incluyendo por cada empleado los dato del empleado y de su departamento.

```
SELECT *
FROM EMPLEADOS INNER JOIN DEPARTAMENTOS
ON EMPLEADOS.Departamento = DEPARTAMENTOS.Codigo;
```

11. Obtener un listado completo de empleados, incluyendo el nombre y apellidos del empleado junto al nombre y presupuesto de su departamento.

```
SELECT E.Nombre, Apellidos, D.Nombre, Presupuesto
FROM EMPLEADOS E INNER JOIN DEPARTAMENTOS D
ON E.Departamento = D.Codigo;
```

12. Obtener los nombres y apellidos de los empleados que trabajen en departamentos cuyo presupuesto sea mayor de 60.000 Bs.

Solución 1 :

```
SELECT EMPLEADOS.Nombre, Apellidos
FROM EMPLEADOS INNER JOIN DEPARTAMENTOS
ON EMPLEADOS.Departamento = DEPARTAMENTOS.Codigo
AND DEPARTAMENTOS.Presupuesto > 60000;
```

Solución 2:

```
SELECT Nombre, Apellidos
FROM EMPLEADOS
WHERE Departamento IN (SELECT Codigo
                       FROM DEPARTAMENTOS
                       WHERE Presupuesto > 60000);
```

13. Obtener los datos de los departamentos cuyo presupuesto es superior al presupuesto medio de todos los departamentos.

```
SELECT *
FROM DEPARTAMENTOS
WHERE Presupuesto > (SELECT AVG(Presupuesto)
                     FROM DEPARTAMENTOS);
```

14. Obtener los nombres de los departamentos que tienen más de dos empleados.

Solución 1 :

```
SELECT Nombre FROM DEPARTAMENTOS
WHERE Codigo IN ( SELECT Departamento
                  FROM EMPLEADOS
                  GROUP BY Departamento HAVING COUNT(*) > 2);
```

Solución 2 :

```
SELECT DEPARTAMENTOS.Nombre
FROM EMPLEADOS INNER JOIN DEPARTAMENTOS
ON Departamento = Codigo
GROUP BY DEPARTAMENTOS.Nombre
HAVING COUNT(*) > 2;
```

15. Añadir un nuevo departamento: 'Calidad', con presupuesto de 40.000 Bs. y código 11. Añadir un empleado vinculado al departamento recién creado: ESTHER VAZQUEZ, DNI: 89267109.

```
INSERT INTO DEPARTAMENTOS VALUES ( 11 , 'Calidad' , 40000);
INSERT INTO EMPLEADOS VALUES ( '89267109' , 'Esther' , 'Vazquez' , 11);
```

16. Aplicar un recorte presupuestario del 10 % a todos los departamentos.

```
UPDATE DEPARTAMENTOS
SET Presupuesto = Presupuesto * 0.9;
```

17. Reasignar a los empleados del departamento de investigación (código 77) al departamento de informática (código 14).

```
UPDATE EMPLEADOS
SET Departamento = 14
WHERE Departamento = 77;
```

18. Despedir a todos los empleados que trabajan para el departamento de informática (código 14).

```
DELETE FROM EMPLEADOS
WHERE Departamento = 14;
```

19. Despedir a todos los empleados que trabajen para departamentos cuyo presupuesto sea superior a los 60.000 Bs.

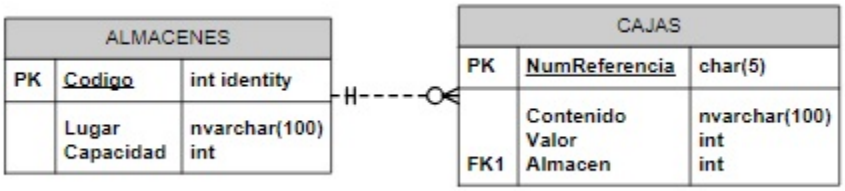
```
DELETE FROM EMPLEADOS
WHERE Departamento IN (SELECT Codigo
                       FROM DEPARTAMENTO
                       WHERE Presupuesto >= 60000);
```

20. Despedir a todos los empleados.

```
DELETE FROM EMPLEADOS;
```

Ejercicio 6 - Los almacenes [9]

Dado el siguiente Modelo Entidad Relación, se pide:



1. Obtener todos los almacenes.

```
SELECT *
FROM ALMACENES;
```

2. Obtener todas las cajas cuyo contenido tenga un valor superior a 150 Bs.

```
SELECT *
FROM CAJAS
WHERE Valor > 150;
```

3. Obtener los tipos de contenidos de las cajas.

```
SELECT DISTINCT Contenido
FROM CAJAS;
```

4. Obtener el valor medio de todas las cajas.

```
SELECT AVG(Valor)
FROM CAJAS;
```

5. Obtener el valor medio de las cajas de cada almacén.

```
SELECT Almacen, AVG(Valor)
FROM CAJAS
GROUP BY Almacen;
```

6. Obtener los códigos de los almacenes en los cuales el valor medio de las cajas sea superior a 150 Bs.

```
SELECT Almacen, AVG(Valor)
FROM CAJAS
GROUP BY Almacen
HAVING AVG(Valor) > 150;
```

7. Obtener el número de referencia de cada caja junto con el nombre de la ciudad en el que se encuentra.

```
SELECT NumReferencia, Lugar
FROM ALMACENES INNER JOIN CAJAS
ON ALMACENES.Codigo = CAJAS.Almacen;
```

8. Obtener el número de cajas que hay en cada almacén.

Solución 1 : Esta consulta no tiene en cuenta los almacenes vacíos

```
SELECT Almacen, COUNT(*)
FROM CAJAS
GROUP BY Almacen;
```

Solución 2: Esta consulta tiene en cuenta los almacenes vacíos

```
SELECT Codigo, COUNT(NumReferencia)
FROM ALMACENES LEFT JOIN CAJAS
ON ALMACENES.Codigo = CAJAS.Almacen
GROUP BY Codigo;
```

9. Obtener los códigos de los almacenes que están saturados (los almacenes donde el número de cajas es superior a la capacidad).

```
SELECT Codigo
FROM ALMACENES
WHERE Capacidad < ( SELECT COUNT(*)
                    FROM CAJAS
                    WHERE Almacen = Codigo);
```

10. Obtener los números de referencia de las cajas que están en Bilbao

Solución 1 :

```
SELECT NumReferencia
FROM ALMACENES LEFT JOIN CAJAS
ON ALMACENES.Codigo = CAJAS.Almacen
WHERE Lugar = 'Bilbao';
```

Solución 2 :

```
SELECT NumReferencia
```

```
FROM CAJAS
WHERE Almacen IN (SELECT Codigo
                  FROM ALMACENES
                  WHERE Lugar = 'Bilbao');
```

11. Insertar un nuevo almacén en Barcelona con capacidad para 3 cajas.

```
INSERT INTO ALMACENES(Lugar,Capacidad) VALUES ('Barcelona',3);
```

12. Insertar una nueva caja, con número de referencia 'H5RT', con contenido 'Papel', valor 200, y situada en el almacén 2.

```
INSERT INTO CAJAS VALUES ('H5RT','Papel',200,2);
```

13. Rebajar el valor de todas las cajas un 15 %.

```
UPDATE CAJAS SET Valor = Valor * 0.85;
```

14. Rebajar un 20 % el valor de todas las cajas cuyo valor sea superior al valor medio de todas las cajas.

```
UPDATE CAJAS SET Valor = Valor * 0.80
WHERE Valor > (SELECT AVG(Valor)
               FROM CAJAS);
```

15. Eliminar todas las cajas cuyo valor sea inferior a 100 Bs.

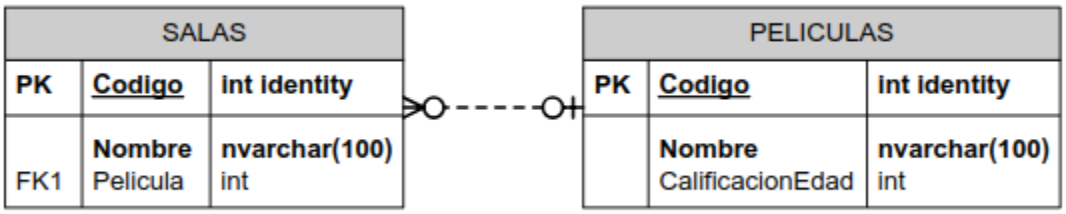
```
DELETE FROM CAJAS
WHERE Valor < 100;
```

16. Vaciar el contenido de los almacenes que están saturados.

```
DELETE FROM CAJAS
WHERE Almacen IN ( SELECT Codigo
                   FROM ALMACENES
                   WHERE Capacidad < (SELECT COUNT(*)
                                      FROM CAJAS
                                      WHERE Almacen = Codigo));
```

Ejercicio 7 - Películas y salas [9]

Dado el siguiente Modelo Entidad Relación, se pide:



1. Mostrar el nombre de todas las películas.

```
SELECT Nombre
FROM PELICULAS;
```

2. Mostrar las distintas calificaciones de edad que existen.

```
SELECT DISTINCT CalificacionEdad
FROM PELICULAS;
```

3. Mostrar todas las pel´ıculas que no han sido calificadas.

```
SELECT *
FROM PELICULAS
WHERE CalificacionEdad IS NULL;
```

4. Mostrar todas las salas que no proyectan ninguna película.

```
SELECT *
FROM SALAS
WHERE Pelicula IS NULL;
```

5. Mostrar la información de todas las salas y, si se proyecta alguna película en la sala, mostrar también la información de la película.

```
SELECT *
FROM SALAS LEFT JOIN PELICULAS
ON SALAS.Pelicula = PELICULAS.Codigo;
```

6. Mostrar la información de todas las películas y, si se proyecta en alguna sala, mostrar también la información de la sala.

```
SELECT *
FROM SALAS RIGHT JOIN PELICULAS
ON SALAS.Pelicula = PELICULAS.Codigo;
```

7. Mostrar los nombres de las películas que no se proyectan en ninguna sala.

Solución 1 :

```
SELECT PELICULAS.Nombre
FROM SALAS RIGHT JOIN PELICULAS
ON SALAS.Pelicula = PELICULAS.Codigo
WHERE SALAS.Pelicula IS NULL;
```

Solución 2 :

```
SELECT Nombre FROM PELICULAS
WHERE Codigo NOT IN (SELECT Pelicula
                     FROM SALAS
                     WHERE Pelicula IS NOT NULL);
```

8. Añadir una nueva pelıcula ‘Uno, Dos, Tres’, para mayores de 7 años.

```
INSERT INTO PELICULAS(Nombre,CalificacionEdad)
VALUES ('Uno, Dos, Tres',7);
```

9. Hacer constar que todas las películas no calificadas han sido calificadas ‘no recomendables para menores de 13 años’.

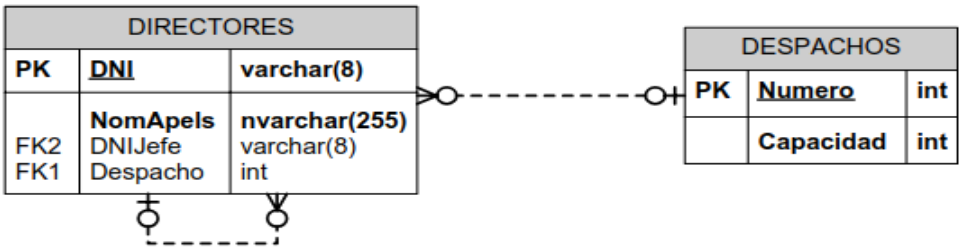
```
UPDATE PELICULAS SET CalificacionEdad = 13
WHERE CalificacionEdad IS NULL;
```

10. Eliminar todas las salas que proyectan películas recomendadas para todos los públicos.

```
DELETE FROM SALAS
WHERE Pelicula IN (SELECT Codigo
                   FROM PELICULAS
                   WHERE CalificacionEdad = 0);
```

Ejercicio 8 - Los Directores [9]

Dado el siguiente Modelo Entidad Relación, se pide:



1. Mostrar el DNI, nombre y apellidos de todos los directores.

```
SELECT DNI, NomApels
FROM DIRECTORES;
```

2. Mostrar los datos de los directores que no tienen jefes.

```
SELECT *
FROM DIRECTORES
WHERE DNIJefe IS NULL;
```

3. Mostrar el nombre y apellidos de cada director, junto con la capacidad del despacho en el que se encuentra.

```
SELECT NomApels, Despacho, Capacidad
FROM DIRECTORES INNER JOIN DESPACHOS
ON DIRECTORES.Despacho = DESPACHOS.Numero;
```

4. Mostrar el número de directores que hay en cada despacho.

   Solución 1 : Sin tener en cuenta despachos vacíos

```
SELECT Despacho, COUNT(*)
FROM DIRECTORES
GROUP BY Despacho;
```

   Solución 2 : Teniendo en cuenta despachos vacíos

```
SELECT Numero, COUNT(DNI)
FROM DESPACHOS LEFT JOIN DIRECTORES
ON DESPACHOS.Numero = DIRECTORES.Despacho
GROUP BY Numero;
```

5. Mostrar los datos de los directores cuyos jefes no tienen jefes.

```
SELECT *
FROM DIRECTORES
WHERE DNIJefe IN (SELECT DNI
                  FROM DIRECTORES
                  WHERE DNIJefe IS NULL);
```

6. Mostrar los nombres y apellidos de los directores junto con los de su jefe.

   Solución 1: Con INNER JOIN. No muestra directores que no tienen jefes.

```
SELECT d1.NomApels, d2.NomApels
FROM DIRECTORES d1 INNER JOIN DIRECTORES d2
ON d1.DNIJefe = d2.DNI;
```

   Solución 2 : Con LEFT JOIN. Si muestra directores sin jefe.

```
SELECT d1.NomApels, d2.NomApels
FROM DIRECTORES d1 LEFT JOIN DIRECTORES d2
ON d1.DNIJefe = d2.DNI;
```

7. Mostrar el número de despachos que están sobre utilizados.

```
SELECT Numero
FROM DESPACHOS
WHERE Capacidad < (SELECT COUNT(*)
                   FROM DIRECTORES
                   WHERE Despacho = Numero);
```

8. Añadir un nuevo director llamado Paco Pérez, DNI 28301700, sin jefe, y situado en el despacho 124.

```
INSERT INTO DIRECTORES VALUES ('28301700', 'Paco Perez', NULL, 124);
```

9. Asignar a todos los empleados de apellido Pérez un nuevo jefe con DNI 74568521.

```
UPDATE DIRECTORES SET DNIJefe = '74568521'
WHERE NomApels LIKE '%Perez%';
```

10. Despedir a todos los directores, excepto a los que no tienen jefe.

```
DELETE FROM DIRECTORES
WHERE DNIJefe IS NOT NULL;
```

Ejercicio 9 - Corretaje de Propiedades [10]

Dado el modelo relacional de una base de datos de una corredora de propiedades:

Arrendatario(RUT, Nombre, Apellido)

Arrienda(***RUTArrendatario, Id_casa***, Deuda) NOTA: si deuda es 0, no hay deuda

Telefonos(RUT, numero)

Dueño(RUT, Nombre, Apellido)

Casa(Id_casa, ***RUTDueno***, Nro, Calle, Sector)

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita

1. Los arrendatarios que arriendan la casa ubicada en la calle Carrera nº 1024, en el sector Santiago.

```
SELECT A.RUT, A.Nombre, A.Apellido
FROM Arrendatario A, Arrienda B, Casa C
WHERE A.RUT = B.RUTArrendatario
AND B.Id_casa = C.Id_casa
AND C.Calle = 'Carrera'
AND C.Nro = '1024'
AND C.Sector = 'Santiago';
```

2. ¿Cuánto le deben a María Pérez?

   Se supondrá que solo existe una María Pérez.

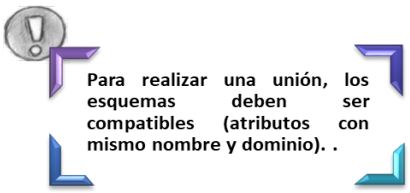
```
SELECT SUM(A.Deuda)
FROM Arrienda A, Casa B, Dueño C
WHERE A.Id_casa = B.Id_casa
AND B.RUTDueno = C.RUT
AND C.Nombre = 'María'
AND C.Apellido = 'Pérez';
```

3. ¿Cuál es la deuda total para cada dueño?

```
SELECT SUM(A.Deuda), C.RUT, C.Nombre, C.Apellido
FROM Arrienda A, Casa B, Dueño C
WHERE A.Id_casa = B.Id_casa
AND B.RUTDueno = C.RUT
GROUP BY C.RUT, C.Nombre, C.Apellido;
```

4. Liste todas las personas de la base de datos. Las personas de la BD son los arrendatarios y los dueños.

```
SELECT Nombre, Apellido
FROM Arrendatario
UNION
SELECT Nombre, Apellido
FROM Dueño;
```

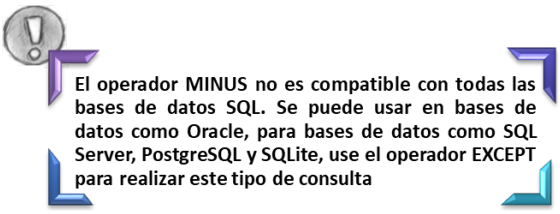


5. Indique los dueños que poseen tres o más casas.

```
SELECT A.RUT, A.Nombre, A.Apellido
FROM Dueño A, Casa C
WHERE A.RUT = C.RUTDueno
GROUP BY A.RUT , A.Nombre, A.Apellido
HAVING COUNT(DISTINCT C.Id_casa)>= 3;
```

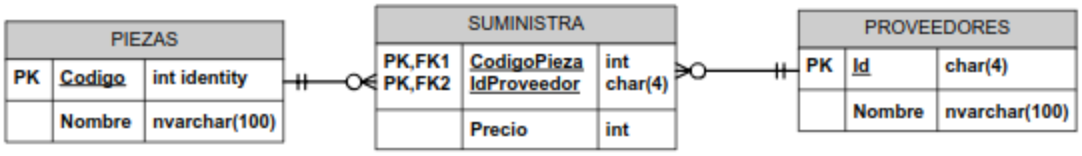
6. Liste los dueños que tengan deudores en todas sus casas.

```
SELECT D.RUT, D.Nombre, D.Apellido
FROM Dueño D, Casa C
WHERE D.RUT = C.RUTDueno
EXCEPT
SELECT D.RUT, D.Nombre, D.Apellido
FROM Dueño D, Casa C, Arrienda A
WHERE D.RUT = C.RUTDueno
AND NOT EXISTS ( SELECT *
                 FROM Arrienda A
                 WHERE C.Id_casa = A.Id_casa
                 AND A.Deuda > 0);
```



Ejercicio 10 - Piezas y Proveedores [9]

Dado el siguiente Modelo Entidad Relación, se pide:



1. Obtener los nombres de todas las piezas.

```
SELECT Nombre
FROM PIEZAS;
```

2. Obtener todos los datos de todos los proveedores.

```
SELECT *
FROM PROVEEDORES;
```

3. Obtener el precio medio al que se nos suministran las piezas.

```
SELECT CodigoPieza, AVG(Precio)
FROM SUMINISTRA
GROUP BY CodigoPieza;
```

4. Obtener los nombres de los proveedores que suministran la pieza 1.

   Solución 1:

```
SELECT PROVEEDORES.Nombre
FROM PROVEEDORES INNER JOIN SUMINISTRA
ON PROVEEDORES.Id = SUMINISTRA.IdProveedor
AND SUMINISTRA.CodigoPieza = 1;
```

   Solución 2:

```
SELECT Nombre
FROM PROVEEDORES
WHERE Id IN (SELECT IdProveedor
             FROM SUMINISTRA
             WHERE CodigoPieza = 1);
```

5. Obtener los nombres de las piezas suministradas por el proveedor cuyo código es HAL.

   Solución 1:

```
SELECT PIEZAS.Nombre
FROM PIEZAS INNER JOIN SUMINISTRA
ON PIEZAS.Codigo = SUMINISTRA.CodigoPieza
AND SUMINISTRA.IdProveedor = 'HAL';
```

Solución 2:

```
SELECT Nombre
FROM PIEZAS
WHERE Codigo IN (SELECT CodigoPieza
                 FROM SUMINISTRA
                 WHERE IdProveedor = 'HAL');
```

Solución 3:

```
SELECT Nombre
FROM PIEZAS
WHERE EXISTS (SELECT *
              FROM SUMINISTRA
              WHERE IdProveedor = 'HAL'
              AND CodigoPieza = PIEZAS.Codigo);
```

6. Obtener los nombres de los proveedores que suministran las piezas más caras, indicando el nombre de la pieza y el precio al que la suministran.

```
SELECT p1.Nombre, pr1.Nombre, Precio
FROM PIEZAS p1 INNER JOIN
     SUMINISTRA s1 INNER JOIN PROVEEDORES pr1
       ON s1.IdProveedor = pr1.Id
       ON p1.Codigo = s1.CodigoPieza
WHERE Precio IN
            (SELECT MAX(Precio) FROM SUMINISTRA s2
             GROUP BY s2.CodigoPieza
             HAVING s2.CodigoPieza = p1.Codigo);
```

7. Hacer constar en la base de datos que la empresa ”Skellington Supplies” (código TNBC) va a empezar a suministrarnos tuercas (código 1) a 7 dólares cada tuerca.

```
INSERT INTO SUMINISTRA
VALUES ( 'TNBC' , 1 , 7 );
```

8. Aumentar los precios en una unidad.

```
UPDATE SUMINISTRA SET Precio = Precio + 1;
```

9. Hacer constar en la base de datos que la empresa ”Susan Calvin Corp.”(RBT) no va a suministrarnos ninguna pieza (aunque la empresa en sí va a seguir constando en nuestra base de datos).

```
DELETE FROM SUMINISTRA
WHERE IdProveedor = 'RBT';
```

10. Hacer constar en la base de datos que la empresa ”Susan Calvin Corp.”(RBT) ya no va a suministrarnos clavos (código 4)

```
DELETE FROM SUMINISTRA
WHERE IdProveedor = 'RBT'
AND CodigoPieza = 4;
```

Ejercicio 11 - Departamentos y Empleados [10]

Dado el DDL de la base de datos de hospitales:

```
CREATE TABLE temple1 (
        nuempl CHAR(6) NOT NULL,
        nombre CHAR(12) NOT NULL,
        inicial CHAR(1) NOT NULL,
        apellido CHAR(15) NOT NULL,
        dept CHAR(3) NOT NULL,
        tlfn CHAR(4),
        feching  DATE NOT NULL,
        codtra SMALLINT NOT NULL,
        niveduc SMALLINT NOT NULL,
        sexo CHAR(1) NOT NULL,
        fechnac DATE  NOT NULL,
        salario DECIMAL(9,2) NOT NULL );

CREATE TABLE tdepar2 (
        numdep CHAR(3) NOT NULL,
        nomdep CHAR(36) NOT NULL,
        numdirec CHAR(6) NOT NULL);
```

1. Obtener una lista de todas las empleadas de los departamentos que empiecen por D o por E. La lista anterior debe contener información sobre el número de personal, nombre, apellido y número de teléfono.

```
SELECT nuempl, nombre, apellido, tlfn , nomdep
FROM temple1 INNER JOIN tdepar2 ON temple1.dept = tdepar2.numdep
WHERE sexo = 'F'
AND (tdepar2.nomdep LIKE 'D%' OR tdepar2.nomdep LIKE 'E%');
```

2. Obtener un listado de todos los empleados (nombre y apellido) que ganan más de 2000 Bs al mes y que entraron en la compañía después del 1 de Enero de 1975. También se quiere la información correspondiente a su código de trabajo y al número de personal de sus directores.

```
SELECT nombre, apellido, codtra, numdirec
FROM temple1 INNER JOIN tdepar2 ON temple1.dept = tdepar2.numdep
WHERE salario > 2000
AND feching > '01/01/1975';
```

3. Obtener una lista con el apellido, número de departamento y salario mensual de los empleados de los departamentos ‘A00’, ‘B01’, ‘C01’ y ‘D01’. La salida se quiere en orden descendente de salario dentro de cada departamento.

```
SELECT apellido, dept, salario
FROM temple1
WHERE dept IN ( 'A00', 'B01', 'C01' , 'D01')
ORDER BY dept, salario DESC;
```

4. Se pide una lista que recupere el salario medio de cada departamento junto con el número de empleados que tiene. El resultado no debe incluir empleados que tengan un código de trabajo mayor que 54, ni departamentos con menos de tres empleados. Se quiere ordenada por número de departamento.

   Solución 1:

```
SELECT salario, numempl
FROM (SELECT AVG(salario) as salario, COUNT(*) AS numempl, dept
      FROM temple1
      WHERE codtra <= 54
      GROUP BY dept ) AS result
WHERE numempl > 2
ORDER BY dept ASC;
```

   Solución 2:

```
SELECT dept, AVG(salario), COUNT(*)
FROM temple1
WHERE codtra <= 54
GROUP BY dept
HAVING COUNT(*) > 2
ORDER BY dept;
```

5. Seleccionar todos los empleados de los departamentos ‘D11’ y ‘E11’ cuyo primer apellido empiece por S.

```
SELECT *
FROM temple1
WHERE dept IN ( 'D11', 'E11' )
AND apellido LIKE 'S%';
```

6. Obtener el nombre, apellido y fecha de ingreso de los directores de departamento ordenados por número de personal.

   Solución 1:

```
SELECT nombre, apellido, feching
FROM temple1
WHERE nuempl IN (SELECT numdirec FROM tdepar2);
```

   Solución 2:

```
SELECT nuempl, nombre, apellido, feching
FROM temple1 , tdepar2
WHERE temple1.nuempl = tdepar2.numdirec
ORDER BY nuempl;
```

7. Obtener un listado de las mujeres de los departamentos que empiecen por D y por E cuyo nivel de educación sea superior a la media; en este caso también ordenados por número de personal.

```
SELECT nuempl, nombre, apellido, tlfn , nomdep
FROM temple1 INNER JOIN tdepar2 ON (temple1.dept = tdepar2.numdep)
WHERE sexo = 'F'
AND (tdepar2.nomdep LIKE 'D%' OR tdepar2.nomdep LIKE 'E%')
AND niveduc > ( SELECT AVG(niveduc)
                FROM temple1 )
ORDER BY nuempl;
```

8. Seleccionar todos los empleados cuyo nombre sea igual al de algunas personas del departamento D21 y cuyo código de trabajo sea diferente de todos los del E21 (la lista debe contener el número de personal, nombre, apellido, departamento y código de trabajo).

Solución 1:

```
SELECT *
FROM temple1
WHERE  nombre IN (SELECT nombre
                  FROM temple1
                  WHERE dept = 'D21')
AND codtra NOT IN (SELECT codtra
                   FROM temple1
                   WHERE dept = 'E21')
AND dept <> 'D21';
```

Solución 2:

```
SELECT nuempl, nombre, apellido, dept, codtra
FROM temple1
WHERE nombre = ANY (SELECT nombre
                    FROM temple1
                    WHERE dept = 'D21')
AND codtra <> ALL ( SELECT codtra
                    FROM temple1 WHERE dept = 'E21')
AND dept <> 'D21';
```

9. Listar los empleados que no sean directores (la información que debe aparecer es el número de personal, apellido y departamento).

```
SELECT *
FROM temple1
WHERE nuempl NOT IN (SELECT numdirec FROM tdepar2);
```

10. Seleccionar parejas de empleados (de sexo opuesto) que hayan nacido el mismo día (con información acerca de apellido y fecha de nacimiento).

```
SELECT  e1.apellido, e1.fechnac, e2.apellido, e2.fechnac
FROM temple1  e1, temple1 e2
WHERE e1.fechnac = e2.fechnac
AND e1.sexo <> e2.sexo;
```

11. Obtener un listado de todos los empleados que pertenecen al mismo departamento que Tomás Soler.

```
SELECT *
FROM temple1
      WHERE dept = (SELECT dept
                    FROM temple1
                    WHERE nombre = 'TOMAS'
                    AND apellido = 'SOLER');
```

Ejercicio 12 - Control de Stock y facturación [11]

Dado el siguiente modelo relacional de una base de datos, se pide:

PROVINCIAS(CODPRO, NOMBRE)
PUEBLOS(CODPUE, NOMBRE, CODPRO)
CLIENTES(CODCLI, NOMBRE, DIRECCION, CODPOSTAL, ***CODPUE***)
VENDEDORES(CODVEN, NOMBRE, DIRECCION, CODPOSTAL, ***CODPUE***, ***CODJEFE***)
ARTICULOS(CODART, DESCRIP, PRECIO, STOCK, STOCK_MIN)
FACTURAS(CODFAC, FECHA, ***CODCLI***, ***CODVEN***, IVA, DTO)
LINEAS_FAC(***CODFAC***, LINEA, CANT, ***CODART***, PRECIO, DTO)

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita.

1. Mostrar las provincias

```sql
SELECT *
FROM provincias;
```

2. Nombre y código de las provincias.

```sql
SELECT nombre, codpro
FROM provincias;
```

3. Mostrar el código de los artículos y el doble del precio de cada artículo.

```sql
SELECT codart, precio * 2
FROM articulos;
```

4. Mostrar el código de la factura, número de línea e importe de cada línea (sin considerar impuestos ni descuentos.

```sql
SELECT codfac, linea, cant * precio AS IMPORTE
FROM lineas_fac;
```

5. Mostrar los distintos tipos de IVA aplicados en las facturas.

```sql
SELECT DISTINCT iva
FROM facturas ;
```

6. Mostrar el código y nombre de aquellas provincias cuyo código es menor a 20.

```
SELECT codpro, nombre
FROM provincias
WHERE codpro < 20;
```

7. Mostrar los distintos tipos de descuento de aplicados por los vendedores que cuyos códigos no superan el valor 50.

```
SELECT DISTINCT dto
FROM facturas
WHERE codven <= 50;
```

8. Mostrar el código y descripción de aquellos artículos cuyo stock es igual o supera los 50 unidades.

```
SELECT codart, descrip
FROM articulos
WHERE stock >= 50;
```

9. Mostrar el código y fechas de las facturas con IVA 16 y que pertenecen al cliente de código 100.

```
SELECT codfac, fecha
FROM facturas
WHERE iva = 16
AND codcli = 100 ;
```

10. Mostrar el código y fechas de las facturas con IVA 16 o con descuento 20 y que pertenecen al cliente de código 100.

```
SELECT codfac, fecha
FROM facturas
WHERE (iva = 16 OR dto = 20)
AND codcli = 100;
```

11. Mostrar el código de la factura y el número de línea de las facturas cuyas líneas superan 100 Bs sin considerar descuentos ni impuestos.

```
SELECT codfac, linea
FROM lineas_fac
WHERE cant * precio > 100;
```

12. Importe medio por factura, sin considerar descuentos ni impuestos. El importe de una factura se calcula sumando el producto de la cantidad por el precio de sus líneas.

```
SELECT SUM( cant * precio ) / COUNT( DISTINCT codfac ), 2 )
FROM lineas_fac;
```

13. Stock medio, máximo, y mínimo de los artículos que contienen la letra A en la segunda posición de su descripción y cuyo stock mínimo es superior a la mitad de su stock actual.

```
SELECT AVG(stock), MAX(stock), MIN(stock)
FROM articulos
WHERE stock_min > stock / 2
AND descrip LIKE '_A%' ;
```

14. Número de facturas para cada año. Junto con el año debe aparecer el número de facturas de ese año.

```
SELECT to_char(fecha, 'yyyy'), COUNT(*)
FROM  facturas
GROUP BY to_char(fecha, 'yyyy');
```

15. Número de facturas de cada cliente, pero sólo se deben mostrar aquellos clientes que tienen más de 15 facturas.

```
SELECT codcli, COUNT(*)
FROM facturas
GROUP BY codcli
HAVING COUNT(*) > 15;
```

16. Cantidades totales vendidas para cada artículo cuyo código empieza por “F”. La cantidad total vendida de un artículo se calcula sumando las cantidades de todas sus líneas de factura.

```
SELECT codart, SUM(cant)
FROM lineas_fac
WHERE codart LIKE 'F%'
GROUP BY codart ;
```

17. Código de aquellos artículos de los que se ha facturado más de 6000 euros.

```
SELECT codart
FROM lineas_fac
GROUP BY codart
HAVING SUM(cant * precio) > 6000;
```

18. Número de facturas de cada uno de los clientes cuyo código está entre 241 y 250, con cada IVA distinto que se les ha aplicado. En cada línea del resultado se debe mostrar un código de cliente, un IVA y el número de facturas de ese cliente con ese IVA.

```
SELECT codcli, iva, COUNT(*)
FROM facturas
WHERE codcli BETWEEN 241 AND 250
GROUP BY codcli, iva;
```

19. Vendedores y clientes cuyo nombre coincide (vendedores que a su vez han comprado algo a la empresa)

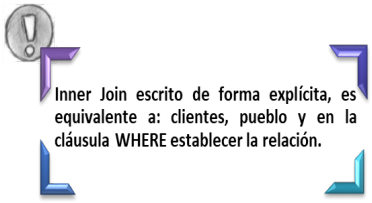
```
SELECT v.nombre
FROM vendedores v
INTERSECT
SELECT c.nombre
FROM clientes e;
```

20. Creación de una vista que muestre únicamente los códigos postales de los clientes que inicien con el número 12.

```
CREATE VIEW codigos_clientes AS
SELECT DISTINCT codpostal
FROM clientes
WHERE codpostal LIKE '12%';
```

21. Mostrar el código y el nombre de los clientes de Castellón (posee código 12) que han realizado facturas con vendedores de más de dos provincias distintas. El resultado debe quedar ordenado ascendentemente respecto del nombre del cliente.

```
SELECT codcli, c.nombre
FROM clientes e INNER JOIN pueblos p
ON p.codpue = c.codpue
WHERE p.codpro = '12'
AND (SELECT COUNT(DISTINCT p2.codpro )
      FROM pueblos p2, vendedores v, facturas f
      WHERE p2.codpue = v.codpue
      AND v.codven = f.codven
      f.codcli = c.codcli ) > 2
ORDER BY 2;
```



Ejercicio 13 - Pedidos [12]

Dado el DDL de la base de datos, se pide:

```
CREATE TABLE EMPLEADOS(
EMPLEADOID NUMBER(8) ,
NOMBRE VARCHAR(30),
APELLIDO VARCHAR(30),
FECHA_NAC DATE,
REPORTA_A NUMBER(8),
EXTENSION NUMBER(8),
CONSTRAINT PK_EMPLEADOS PRIMARY KEY
(EMPLEADOID));

CREATE TABLE CATEGORIAS(
CATEGORIAID NUMBER(8),
NOMBRECAT VARCHAR(50) NOT NULL,
CONSTRAINT PK_CATEGORIAS PRIMARY
KEY(CATEGORIAID) ) ;

CREATE TABLE CLIENTES(
CLIENTEID NUMBER(8),
CEDULA_RUC VARCHAR(10) NOT NULL,
NOMBRECIA VARCHAR(30) NOT NULL,
NOMBRECONTACTO VARCHAR(50) NOT NULL,
DIRECCIONCLI VARCHAR(50) NOT NULL,
FAX VARCHAR(12),
EMAIL VARCHAR(50),
CELULAR VARCHAR(12),
FIJO VARCHAR(12),
CONSTRAINT PK_CLIENTES PRIMARY
KEY(CLIENTEID) );

CREATE TABLE DETALLE_ORDENES(
ORDENID NUMBER(8),
DETALLEID NUMBER(8),
PRODUCTOID NUMBER(8) NOT NULL,
CANTIDAD NUMBER(8) NOT NULL,
CONSTRAINT PK_DETALLE_ORDENES PRIMARY
KEY (ORDENID,DETALLEID) );

CREATE TABLE PROVEEDORES(
PROVEEDORID NUMBER(8) ,
NOMBREPROV VARCHAR(50) NOT NULL,
CONTACTO VARCHAR(50) NOT NULL,
CELUPROV VARCHAR(12),
FIJOPROV VARCHAR(12),
CONSTRAINT PK_PROVEEDORES PRIMARY
KEY(PROVEEDORID ) );

CREATE TABLE ORDENES(
ORDENID NUMBER(8),
EMPLEADOID NUMBER(8) NOT NULL,
CLIENTEID NUMBER(8) NOT NULL,
FECHAORDEN DATE NOT NULL,
DESCUENTO NUMBER(8) NULL,
CONSTRAINT PK_ORDENES PRIMARY
KEY(ORDENID) );

CREATE TABLE PRODUCTOS(
PRODUCTOID NUMBER(8),
PROVEEDORID NUMBER(8) NOT NULL,
CATEGORIAID NUMBER(8) NOT NULL,
DESCRIPCION VARCHAR(50),
PRECIOUNIT NUMBER(8) NOT NULL,
EXISTENCIA NUMBER(8) NOT NULL,
CONSTRAINT PK_PRODUCTOS PRIMARY KEY
(PRODUCTOID)) ;
```

ALTER TABLE ORDENES ADD CONSTRAINT FK_ORDENES_CLIEN_ORD_CLIENTES FOREIGN KEY (CLIENTEID) REFERENCES CLIENTES (CLIENTEID);

ALTER TABLE ORDENES ADD CONSTRAINT FK_ORDENES_EMPLE_ORD_EMPLEADO FOREIGN KEY (EMPLEADOID) REFERENCES EMPLEADOS (EMPLEADOID);

ALTER TABLE DETALLE_ORDENES ADD CONSTRAINT FK_DETALLE__ORDEN_DET_ORDENES FOREIGN KEY (ORDENID)REFERENCES ORDENES (ORDENID);

ALTER TABLE DETALLE_ORDENES ADD CONSTRAINT FK_DETALLE__PROD_DETA_PRODUCTO FOREIGN KEY (PRODUCTOID) REFERENCES PRODUCTOS (PRODUCTOID);

ALTER TABLE PRODUCTOS ADD CONSTRAINT FK_PRODUCTO_CATE_PROD_CATEGORI FOREIGN KEY (CATEGORIAID) REFERENCES CATEGORIAS (CATEGORIAID);

ALTER TABLE PRODUCTOS ADD CONSTRAINT FK_PRODUCTO_PROV_PROD_PROVEEDO FOREIGN KEY (PROVEEDORID) REFERENCES PROVEEDORES (PROVEEDORID);

ALTER TABLE EMPLEADOS ADD CONSTRAINT FK_EMPLEADO_REPORTA FOREIGN KEY (REPORTA_A) REFERENCES EMPLEADOS (EMPLEADOID);

1. Actualizar el precio unitario de los productos de la categoría CÁRNICOS, subiéndose en un 10%

```
UPDATE PRODUCTOS
SET PRECIOUNIT = (PRECIOUNIT * 0.1) + PRECIOUNIT
WHERE CATEGORIAID = (SELECT CATEGORIAID
                     FROM CATEGORIAS
                     WHERE NOMBRECAT = 'CARNICOS');
```

2. Actualizar el teléfono celular del proveedor cuyo contacto es MANUEL ANDRADE, con el valor 099010291

```
UPDATE PROVEEDORES
SET CELUPROV = '0990101291'
WHERE CONTACTO = 'MANUEL ANDRADE';
```

3. Eliminar el producto YOGURT DE SABORES

```
DELETE FROM PRODUCTOS
WHERE DESCRIPCION ='YOGURT DE SABORES';
```

4. Mostrar todas las órdenes. Se pide el id de la orden, el apellido y nombre del empleado que la atendió el nombre de la compañía cliente y la fecha de orden.

```
SELECT ORD.ORDENID, EMP.NOMBRE, EMP.APELLIDO, ORD.FECHAORDEN,
       CLI.CLIENTEID, CLI.NOMBRECIA
FROM ORDENES ORD, EMPLEADOS EMP, CLIENTES CLI
WHERE ORD.EMPLEADOID = EMP.EMPLEADOID
AND ORD.CLIENTEID = CLI.CLIENTEID ;
```

5. Mostrar la suma total de cada tipo de producto pedidos en todas las órdenes.

```
SELECT PRODUCTOID, SUM(CANTIDAD) AS SUMA_TOTAL
FROM DETALLE_ORDENES
GROUP BY PRODUCTOID;
```

6. Mostrar el número de órdenes atendidas por cada empleado, incluidos los que tienen 0 órdenes.

```
SELECT NOMBRE, APELLIDO, COUNT(ORD.EMPLEADOID) AS NUM_ORDENES
FROM EMPLEADOS EMP LEFT JOIN ORDENES ORD
ON EMP.EMPLEADOID = ORD.EMPLEADOID
GROUP BY NOMBRE, APELLIDO;
```

7. Muestre los proveedores y la suma de dinero vendido en los productos de ese proveedor.

```
SELECT PROV.NOMBREPROV, SUM(D.CANTIDAD * PROD.PRECIOUNIT) AS SUMA_DINERO
FROM PROVEEDORES PROV, PRODUCTOS PROD, DETALLE_ORDENES D
WHERE D.PRODUCTOID = PROD.PRODUCTOID
AND PROV.PROVEEDORID = PROD.PROVEEDORID
GROUP BY NOMBREPROV;
```

Ejercicio 14 - Locales Nocturnos

PUB
#COD_PUB
NOMBRE
LICENCIA_FISCAL
DOMICILIO
FECHA_APERTURA
HORARIO
COD_LOCALIDAD

TITULAR
#DNI_TITULAR
NOMBRE
DOMICILIO
COD_PUB

EMPLEADO
#DNI_EMPLEADO
NOMBRE
DOMICILIO

EXISTENCIAS
#COD_ARTICULO
NOMBRE
CANTIDAD
PRECIO
COD_PUB

LOCALIDAD
#COD_LOCALIDAD
NOMBRE

PUB_EMPLEADO
#COD_PUB
#DNI_EMPLEADO
#FUNCION

Se pide escribir los comandos SQL (DDL) que permitan la creación de las tablas anteriores teniendo en cuenta las siguientes restricciones:

- Todos los valores son de tipo carácter excepto los campos FECHA_APERTURA (fecha) , CANTIDAD, PRECIO y COD_LOCALIDAD (numéricos).
- Los únicos campos que no son obligatorios son los campos DOMICILIO.
- Los valores del campo horario sólo pueden ser HOR1, HOR2 y HOR3.
- No es posible dar de alta EXISTENCIAS a precio 0.
- El campo función de la tabla PUB_EMPLEADO sólo puede tener los valores CAMARERO, SEGURIDAD, LIMPIEZA.
- Se ha de mantener la integridad referencial entre las tablas.
- Las claves primarias vienen marcadas con el símbolo #.

```
CREATE TABLE pub (
    cod_pub         VARCHAR(5)     NOT NULL,
    nombre          VARCHAR(60)    NOT NULL,
    licencia_fiscal VARCHAR(60)    NOT NULL,
    domicilio       VARCHAR(60),
    fecha_apertura  DATE           NOT NULL,
    horario         VARCHAR(10)    NOT NULL,
    cod_localidad   NUMBER(5)      NOT NULL,
   CONSTRAINT pk_pub PRIMARY KEY (cod_pub),
   CONSTRAINT  fk_pub_localidad  FOREIGN  KEY  (cod_localidad)  REFERENCES
localidad (cod_localidad),
    CONSTRAINT ck_horario  CHECK (horario IN ('HOR1', 'HOR2', 'HOR3'))) ;

CREATE TABLE titular (
    dni_titular     VARCHAR(8)   NOT NULL,
    nombre          VARCHAR(60)  NOT NULL,
    domicilio       VARCHAR(60),
    cod_pub         VARCHAR(5)   NOT NULL ,
    CONSTRAINT pk_titular PRIMARY KEY (dni_titular),
     CONSTRAINT  fk_titular_pu  FOREIGN  KEY  (cod_pub)  REFERENCES  pub
(cod_pub)) ;

CREATE TABLE empleado (
    dni_empleado    VARCHAR(8)   NOT NULL,
    nombre          VARCHAR(60)  NOT NULL,
    domicilio       VARCHAR(60),
    CONSTRAINT pk_empleado PRIMARY KEY (dni_empleado) ) ;

CREATE TABLE existencias (
    cod_articulo    VARCHAR(10)      NOT NULL,
    nombre          VARCHAR(60)      NOT NULL,
    cantidad        NUMBER(8)        NOT NULL,
    precio          NUMBER(8,2)      NOT NULL,
    cod_pub         VARCHAR(5)       NOT NULL,
    CONSTRAINT pk_existencias PRIMARY KEY (cod_articulo),
    CONSTRAINT  fk_existencias_pub FOREIGN KEY (cod_pub)  REFERENCES pub
(cod_pub),
    CONSTRAINT ck_precio  CHECK (precio > 0) ) ;

CREATE TABLE localidad (
    cod_localidad   NUMBER(5)      NOT NULL,
    nombre          VARCHAR(60)    NOT NULL,
    CONSTRAINT pk_localidad PRIMARY KEY (cod_localidad) ) ;
```

```
CREATE TABLE pub_empleado (
    cod_pub         VARCHAR(5)   NOT NULL,
    dni_empleado    VARCHAR(8)   NOT NULL,
    funcion         VARCHAR(9)   NOT NULL,
     CONSTRAINT   pk_pub_empleado  PRIMARY  KEY  (cod_pub,  dni_empleado,
funcion),
   CONSTRAINT   fk_pubemple_pub   FOREIGN   KEY   (cod_pub)   REFERENCES
pub(cod_pub),
CONSTRAINT  fk_pubemple_empleado  FOREIGN  KEY  (dni_empleado)  REFERENCES
empleado (dni_empleado),
  CONSTRAINT   ck_funcion   CHECK   (funcion   IN   ('CAMARERO','SEGURIDAD',
'LIMPIEZA')) ;
```

Ejercicio 15 - Empleados de una empresa

Dado el DDL de la base de datos, se pide:

EMPLEADOS

Column Name	DataType
DNI	NUMBER(8)
NOMBRE	VARCHAR(10)
APELLIDO1	VARCHAR(15)
APELLIDO2	VARCHAR(15)
DIRECC1	VARCHAR(25)
DIRECC2	VARCHAR(20)
CIUDAD	VARCHAR(20)
PROVINCIA	VARCHAR(20)
COD_POSTAL	VARCHAR(5)
SEXO	VARCHAR(1)
FECHA_NAC	DATE

DEPARTAMENTOS

Column Name	DataType
DPTO_COD	NUMBER(5)
NOMBRE_DPTO	VARCHAR(30)
DPTO_PADRE	NUMBER(5)
PRESUPUESTO	NUMBER
PRES_ACTUAL	NUMBER

ESTUDIOS

Column Name	Data Type
EMPLEADO_DNI	NUMBER(8)
UNIVERSIDAD	NUMBER(5)
AÑO	NUMBER
GRADO	VARCHAR(3)
ESPECIALIDAD	VARCHAR(20)

HISTORIAL_LABORAL

Column Name	Data Type
EMPLEADO_DNI	NUMBER(8)
TRABAJO_COD	NUMBER(5)
FECHA_INICIO	DATE
FECHA_FIN	DATE
DPTO_COD	NUMBER(5)
SUPERVISOR_DNI	NUMBER(8)

UNIVERSIDADES

Column Name	Data Type
UNIV_COD	NUMBER(5)
NOMBRE_UNIV	VARCHAR(25)
CIUDAD	VARCHAR(20)
MUNICIPIO	VARCHAR(2)
COD_POSTAL	VARCHAR(5)

HISTORIAL_SALARIAL

Column Name	Data Type
EMPLEADO_DNI	NUMBER(8)
SALARIO	NUMBER
FECHA_COMIENZO	DATE
FECHA_FIN	DATE

TRABAJOS

Column Name	Data Type
TRABAJO_COD	NUMBER(5)
NOMBRE_TRAB	VARCHAR(20)
SALARIO_MIN	NUMBER(2)
SALARIO_MAX	NUMBER(2)

1. El atributo SEXO en EMPLEADOS sólo puede tomar los valores H y M.

```
ALTER TABLE empleados
ADD CONSTRAINT ck_sexo  CHECK (sexo IN ('H','M')) ;
```

2. Dos DEPARTAMENTOS no se llaman igual. Dos TRABAJOS tampoco.

```
ALTER TABLE departamentos
ADD CONSTRAINT uk_nombre_dpto UNIQUE (nombre_dpto) ;

ALTER TABLE trabajos
ADD CONSTRAINT uk_nombre_trab UNIQUE (nombre_trab) ;
```

3. Añada un nuevo atributo VALORACIÓN en la tabla de EMPLEADOS que indique de 1 a 10 la valoración que obtuvo el empleado en su entrevista de trabajo al iniciar su andadura en la empresa. Ponga el valor por defecto 5 para ese campo.

```
ALTER TABLE empleados ADD COLUMN valoracion NUMBER DEFAULT 5 ;

ALTER TABLE empleados
ADD CONSTRAINT ck_valoracion CHECK (valoracion BETWEEN 1 AND 10) ;
```

4. Elimine la restricción de que el atributo NOMBRE de la tabla EMPLEADOS no puede ser nulo.

```
ALTER TABLE empleados MODIFY nombre VARCHAR(10) NULL ;
```

5. Modificar el tipo de datos de DIREC1 de la tabla EMPLEADOS a cadena de caracteres de 40 como máximo

```
ALTER TABLE empleados MODIFY direcc1 VARCHAR(40) ;
```

6. Crear una nueva tabla llamada INFORMACIÓN UNIVERSITARIA que tenga el NOMBRE y los dos APELLIDOS (en un solo atributo) de todos los EMPLEADOS junto con la UNIVERSIDAD donde estudiaron.

```
CREATE TABLE información_universitaria AS
SELECT
    Nombre||' '||apellido1||' '||apellido2 as nombre_empl,
    nombre_univ as nombre_univ
FROM
    empleados e, estudios es, universidades u
WHERE
    e.dni=es.empleado_dni
    AND es.universidad = u.univ_cod;
```

7. Crear una vista llamada NOMBRE_EMPLEADOS con el NOMBRE y los dos APELLIDOS (en un solo atributo) de todos los EMPLEADOS que son de Vargas.

```
CREATE VIEW empleados_malaga AS
SELECT
    Nombre||' '||apellido1||' '||apellido2 as nombre_empl
FROM
    empleados
WHERE
    municipio = 'Vargas';
```

8. Crear otra vista llamada INFORMACION_EMPLEADOS con el NOMBRE y los dos APELLIDOS (en un solo atributo) y EDAD (no fecha de nacimiento) de todos los EMPLEADOS.

```
CREATE VIEW informacion_empleados AS
SELECT
    Nombre||' '||apellido1||' '||apellido2 as nombre_empl,
    round(SYSDATE - fecha_nacimiento/365) edad
FROM
    empleados;
```

9. Crear otra vista sobre la anterior llamada INFORMACION_ACTUAL que dispone de toda la información de INFORMACION_EMPLEADOS junto con el SALARIO que está cobrando en este momento

```
CREATE VIEW informacion_actual AS
SELECT
    nombre_empl,
    salario
FROM
    informacion_empleados ie,
    historial_salarial hs
WHERE
    Ie.dni = hs.empleado_dni
AND fecha_fin IS NULL;
```

Ejercicio 16 - Departamentos y empleados [13]

Dado el siguiente modelo relacional de una base de datos de una agencia de viajes

> EMPLEADO (NDIEMP, NOMEMP, SEXEMP, FECNAC, FECINCORPORACION, SALEMP, COMIS, CODSUC, *JEFEDI*, CARGO, ***NRODEPTO***)
>
> DEPARTAMENTO (CODDEPTO, NOMBREDPTO, CIUDAD, DIRECTOR)

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita.

1. Obtener los datos completos de los empleados.

```
SELECT *
FROM Empleado;
```

2. Obtener los datos completos de los departamentos

```
SELECT *
FROM Departamento;
```

3. Obtener los datos de los empleados con cargo 'Secretaria'.

```
SELECT *
FROM Empleado
WHERE cargoE = 'Secretaria';
```

4. Obtener el nombre y salario de los empleados.

```
SELECT nomEmp, salEmp
FROM Empleado;
```

5. Obtener los datos de los empleados vendedores, ordenado por nombre.

```
SELECT *
FROM Empleado
WHERE cargoE = 'Vendedor'
ORDER BY nomEmp;
```

6. Listar el nombre de los departamentos, ordenado por nombre

```
SELECT DISTINCT nombreDpto
FROM Departamento
ORDER BY nombreDpto;
```

7. Obtener el nombre y cargo de todos los empleados, ordenado por salario.

```
SELECT nomEmp, cargoE
FROM Empleado
ORDER BY salEmp;
```

8. Obtener el nombre y cargo de todos los empleados, ordenado por cargo y por salario.

```
SELECT nomEmp, cargoE, salEmp
FROM Empleado
ORDER BY cargoE, salEmp;
```

9. Obtener el nombre y cargo de todos los empleados, en orden inverso por cargo.

```
SELECT nomEmp, cargoE
FROM Empleado
ORDER BY cargoE DESC;
```

Ejercicio 17 - Empleados de una empresa [14]

Dada la siguiente base de datos relacional de una empresa, se pide:

EMPLEADO : Tabla

codigo_c	nombre	edad	oficio	dir	fecha_alt	salario	comision	depto_no
281-160483-0005F	Rocha Vargas Hector	27	Vendedor	Leon	12/05/1983	12000	0	40
281-040483-0056P	López Hernandez Julio	27	Analista	Chinandega	14/07/1982	13000	1500	20
081-130678-0004S	Esquivel José	31	Director	Juigalpa	05/06/1981	16700	1200	30
281-160473-0009Q	Delgado Carmen	37	Vendedor	Leon	02/03/1983	13400	0	40
281-160493-0005F	Castillo Montes Luis	17	Vendedor	Masaya	12/08/1982	16309	1000	40
281-240784-0004Y	Esquivel Leonel Alfonso	26	Presidente	Nagarote	12/09/1981	15000	0	30
281-161277-0008R	Perez Luis	32	Empleado	Managua	02/03/1980	16890	0	10

DEPARTAMENTO : Tabla

depto_no	nombre_depto	localizacion
10	Desarrollo Software	El Coyolar
20	Analisis Sistema	Guadalupe
30	Contabilidad	Subtiava
40	Ventas	San Felipe
0		

1. Mostrar los nombres de los empleados ordenados alfabéticamente (Z...A)

```
SELECT nombre
FROM EMPLEADO
ORDER BY nombre DESC;
```

2. Seleccionar el nombre, el oficio y la localidad de los departamentos donde trabajan los vendedores.

```
SELECT nombre, oficio, localizacion
FROM EMPLEADO,DEPARTAMENTO
WHERE oficio = 'Vendedor'
AND EMPLEADO.dept_no = DEPARTAMENTO.dept_no;
```

3. Listar los nombres de los empleados cuyo nombre termine con la letra ‘o’.

```
SELECT nombre
FROM EMPLEADO
WHERE nombre LIKE '%o';
```

4. Seleccionar el nombre, el oficio y salario de los empleados que la localización donde trabajan es San Felipe.

```
SELECT nombre, oficio, salario
FROM EMPLEADO, DEPARTAMENTO
WHERE EMPLEADO.dept_no = DEPARTAMENTO.dept_no
AND localizacion = 'San Felipe';
```

5. Seleccionar el nombre, salario y localidad donde trabajan los empleados que tengan un salario entre 10000 y 13000.

```
SELECT nombre, salario, localizacion
FROM EMPLEADO, DEPARTAMENTO
WHERE EMPLEADO.dept_no = DEPARTAMENTO.dept_no
AND salario BETWEEN 10000 and 13000;
```

6. Visualizar los departamentos con más de 5 empleados.

```
SELECT EMPLEADO.depto_no, COUNT(*)
FROM EMPLEADO, DEPARTAMENTO
WHERE EMPLEADO.dept_no = DEPARTAMENTO.dept_no
GROUP BY EMPLEADO.depto_no
HAVING COUNT(*) > 5;
```

7. Mostrar el nombre, salario y nombre del departamento de los empleados que tengan el mismo oficio que ‘Leonel Alfonso Esquivel’.

```
SELECT nombre, salario, nombre_depto
FROM EMPLEADO, DEPARTAMENTO
WHERE EMPLEADO.dept_no = DEPARTAMENTO.dept_no
AND oficio = (SELECT oficio
              FROM EMPLEADO
              WHERE nombre = 'Leonel Alfonso Esquivel');
```

8. Mostrar el nombre, salario y nombre del departamento de los empleados que tengan el mismo oficio que “Castillo Montes Luis” y que no tengan comisión.

```
SELECT nombre, salario, nombre_depto
FROM EMPLEADO, DEPARTAMENTO
WHERE EMPLEADO.dept_no = DEPARTAMENTO.dept_no
AND oficio = (SELECT oficio
              FROM EMPLEADO
              WHERE nombre = 'Castillo Montes Luis')
AND comision IS NULL;
```

9. Mostrar los datos de los empleados que trabajan en el departamento de contabilidad, ordenados por nombre.

```
SELECT *
FROM EMPLEADO, DEPARTAMENTO
WHERE EMPLEADO.dept_no = DEPARTAMENTO.dept_no
AND nombre_depto = 'Contabilidad'
ORDER BY nombre;
```

10. Nombre de los empleados que trabajan en Caracas y cuyo oficio sea analista o empleado.

```
SELECT nombre
FROM EMPLEADO, DEPARTAMENTO
WHERE EMPLEADO.depto_no = Departamento.depto_no
AND localizacion = 'Caracas'
AND oficio IN ('Analista','Empleado');
```

11. Calcula el salario medio de todos los empleados.

```
SELECT AVG(salario)
FROM EMPLEADO;
```

12. ¿Cuál es el máximo salario de los empleados del departamento 10?

```
SELECT MAX(salario)
FROM EMPLEADO
WHERE depto_no = 10;
```

13. Calcular el salario mínimo de los empleados del departamento ‘VENTAS’.

```
SELECT MIN(salario)
FROM EMPLEADO, DEPARTAMENTO
WHERE nombre_depto = 'Ventas'
AND EMPLEADO.dept_no = DEPARTAMENTO.dept_no;
```

14. Calcular el promedio del salario del departamento de ‘CONTABILIDAD’.

```
SELECT AVG(salario)
FROM EMPLEADO , DEPARTAMENTO
WHERE EMPLEADO.dept_no = DEPARTAMENTO.dept_no
AND nombre_depto = 'Contabilidad';
```

15. ¿Cuántos empleados hay en el departamento número 10?

```
SELECT COUNT(*)
FROM EMPLEADO
WHERE depto_no = 10;
```

16. ¿Cuántos empleados hay en el departamento de 'VENTAS'?

```
SELECT COUNT(*)
FROM EMPLEADO, DEPARTAMENTO
WHERE nombre_depto = 'VENTAS';
```

17. Calcular el número de empleados que no tienen comisión.

```
SELECT COUNT(*)
FROM EMPLEADO
WHERE comision IS NULL;
```

18. Visualizar cuántos nombres de los empleados empiezan por la letra 'A'.

```
SELECT COUNT(*)
FROM EMPLEADO
WHERE nombre LIKE 'A%';
```

19. Visualizar el número de empleados de cada departamento.

```
SELECT EMPLEADO.depto_no, COUNT(*)
FROM EMPLEADO, DEPARTAMENTO
WHERE EMPLEADO.dept_no = DEPARTAMENTO.dept_no
GROUP BY EMPLEADO.depto_no;
```

20. Para cada oficio obtener la suma de salarios.

```
SELECT oficio, SUM(salario)
FROM EMPLEADO
GROUP BY oficio;
```

21. Mostrar los datos de los empleados cuyo salario sea mayor que la media de todos los salarios.

```
SELECT *
FROM EMPLEADO
WHERE salario > (SELECT AVG(salario)
                 FROM EMPLEADO);
```

22. Seleccionar el nombre del empleado que tiene máximo salario.

```
SELECT nombre
FROM EMPLEADO
WHERE salario = (SELECT MAX(salario) FROM EMPLEADO);
```

23. Mostrar el nombre del empleado que tiene el salario más bajo.

```
SELECT nombre
FROM EMPLEADO
WHERE salario = (SELECT MIN(salario) FROM EMPLEADO);
```

24. Mostrar los datos del empleado que tiene el salario más alto en el departamento de 'VENTAS'.

```
SELECT nombre
FROM EMPLEADO,DEPARTAMENTO
WHERE EMPLEADO.depto_no = DEPARTAMENTO.depto_no
AND nombre_depto = 'VENTAS'
AND salario = (SELECT MAX(salario)
               FROM EMPLEADO,DEPARTAMENTO
               WHERE EMPLEADO.depto_no = DEPARTAMENTO.depto_no
               AND nombre_depto = 'VENTAS');
```

25. Mostrar el número de oficios distintos de cada departamento.

```
SELECT depto_no,oficio, COUNT(*)
FROM EMPLEADO
GROUP BY depto_no,oficio;
```

26. Mostrar los departamentos que trabajando más de dos personas de la misma profesión.

```
SELECT depto_no, oficio, COUNT(*)
FROM EMPLEADO
GROUP BY depto_no, oficio
HAVING COUNT(*) > 2;
```

27. Insertar en la tabla EMPLEADO un empleado con código 081-220678-0008U, nombre 'Pérez Luis Carlos' de 32 años, oficio Analista, vive en Matagalpa, fecha de alta en la empresa el 22-06-2010, su salario es 15600, no tiene comisión y pertenece al departamento 20.

```
INSERT INTO EMPLEADO
(codigo_c,nombre,edad,oficio,dir,fecha_alt,salario,comision, depto_no)
VALUES ('220678-0008U', 'Pérez Luis
Carlos',32,'Analista','Matagalpa','22-06-2010',15600,null,20);
```

28. Insertar en la tabla DEPARTAMENTO un departamento cuyo número sea 50, de nombre ‘GENERAL’ y cuya localización sea ‘Caracas’.

```
INSERT INTO DEPARTAMENTO
VALUES (50,'General','Caracas');
```

29. Insertar en la tabla DEPARTAMENTO un departamento cuyo número sea 60 y de nombre ‘PRUEBAS’.

```
INSERT INTO DEPARTAMENTO(depto_no,nombre_depto)
VALUES (60,'Prueba');
```

30. Insertar en la tabla PRUEBA los datos de los empleados que pertenecen al departamento número 30.

```
INSERT INTO PRUEBA (SELECT * FROM EMPLEADO WHERE depto_no = 30);
```

31. Insertar en la tabla PRUEBA2 el codigo_c, número de departamento y salario de los empleados que pertenecen al departamento número 20.

```
INSERT INTO PRUEBA2 (SELECT codigo_c,depto_no,salario
                     FROM EMPLEADO
                     WHERE depto_no = 20);
```

32. Doblar el salario a todos los empleados del departamento 30.

```
UPDATE EMPLEADO
SET salario = salario * 2
WHERE depto_no = 30;
```

33. Cambiar todos los empleados del departamento número 30 al departamento número 20.

```
UPDATE EMPLEADO
SET depto_no = 20
WHERE depto_no = 30;
```

34. Incrementar en un 10% el sueldo de los empleados del departamento 10.

```
UPDATE EMPLEADO
SET salario = salario * 1.1
WHERE depto_no = 10;
```

35. Cambiar la localidad del departamento número 10 a ‘Maracaibo’.

```
UPDATE EMPLEADO
SET localizacion = 'Maracaibo' WHERE depto_no = 10;
```

36. Igualar el salario de ‘Rodriguez Jose’ al salario de ‘Ruiz Leonel Alfonso’, de la tabla PRUEBA.

```
UPDATE EMPLEADO
SET salario = (SELECT salario FROM PRUEBA
               WHERE nombre = 'Ruiz Leonel Alfonso')
WHERE nombre = 'Rodriguez Jose';
```

37. En la tabla DEPARTAMENTO borrar el departamento número 40.

```
DELETE FROM DEPARTAMENTO
WHERE depto_no = 40;
```

38. En la tabla EMPELADO borrar todos los empleados que sean del departamento 20 y sean ‘ANALISTAS’.

```
DELETE FROM EMPLEADO
WHERE depto_no = 20
AND oficio = 'Analista';
```

39. Borrar de la tabla EMPLEADO todos los empleados que no tengan comisión.

```
DELETE FROM EMPLEADO WHERE comision IS NULL;
```

Ejercicio 18 - Libros [15]

Dado el siguiente modelo relacional, se pide:

LIBRO (IDLIBRO, TITULO, EDITORIAL, AREA)
AUTOR (IDAUTOR, NOMBRE, NACIONALIDAD)
ESTUDIANTE (IDLECTOR, CI, NOMBRE, DIRECCION, CARRERA, EDAD)
LIBAUTOR (***IDAUTOR, IDLIBRO***)
PRESTAMO (*FECHAPRESTAMO,* ***IDLECTOR, IDLIBRO,*** FECHADEVOLUCION,DEVUELTO)

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita.

1. Listar los datos de los autores.

```
SELECT *
FROM AUTOR;
```

2. Listar nombre y edad de los estudiantes.

```
SELECT nombre, edad
FROM ESTUDIANTE;
```

3. ¿ Qué estudiantes pertenecen a la carrera de Informática?

```
SELECT nombre
FROM ESTUDIANTE
WHERE carrera = 'Informatica';
```

4. Listar el nombre de los estudiantes cuyo nombre comience por la letra G.

```
SELECT nombre
FROM ESTUDIANTE
WHERE nombre LIKE 'G%';
```

5. Listar los nombre de los autores del libro Base de Datos I, utilizando subconsulta.

```
SELECT A.nombre
FROM AUTOR A, LIBAUTOR L
WHERE A.IDAUTOR = L.IDAUTOR
AND L.IDLIBRO IN (SELECT IDLIBRO
                  FROM LIBRO WHERE titulo = 'Base de Datos I');
```

6. ¿ Qué autores son de nacionalidad francesa o venezolana ?

```
SELECT *
FROM AUTOR
WHERE nacionalidad IN ('Francesa', 'Venezolana');
```

7. ¿ Cuáles son los libros que no pertenecen al área de informática ?

```
SELECT *
FROM LIBRO
WHERE area <> 'Informatica';
```

8. Utilizando subconsulta, que libros fueron prestados al lector Pedro Ruiz.

```
SELECT *
FROM LIBRO
WHERE idlibro IN (SELECT P.idlibro
                  FROM PRESTAMO P, ESTUDIANTE E
                  WHERE P.idlector = E.idlector
                  AND E.nombre = 'Pedro Ruiz');
```

9. Listar los nombres de los estudiantes de menor edad.

```
SELECT nombre
FROM ESTUDIANTE
WHERE edad = (SELECT MIN(edad)
              FROM ESTUDIANTE );
```

10. Utilizando subconsulta, listar los estudiantes a los que se les prestó libros del área de Bases de Datos.

```
SELECT nombre
FROM ESTUDIANTE
WHERE idlector IN (SELECT P.idlector
                   FROM PRESTAMO P, LIBRO L
                   WHERE P.idlibro = L.idlibro
                   AND L.area = 'Bases de Datos');
```

11. Listar los libros de la editorial UCAB.

```
SELECT *
FROM LIBRO
WHERE editorial = 'UCAB';
```

12. Listar los libros que son del autor Mario Benedetti.

```
SELECT *
FROM LIBRO L , AUTOR A , LIBAUTOR LA
WHERE L.idlibro = LA.idlibro
AND LA.idautor = A.idautor
AND A.nombre = 'Mario Benedetti';
```

13. Listar los títulos de los libros que debían devolverse el 01-09-2019.

```
SELECT titulo
FROM LIBRO
WHERE idlibro IN (SELECT idlibro
                  FROM PRESTAMO
                  WHERE fechadevolucion = '01-09-2019'
                  AND devuelto = 'NO' );
```

14. Hallar la suma de las edades de los estudiantes.

```
SELECT SUM(edad) AS 'LA SUMA DE EDADES'
FROM ESTUDIANTE;
```

15. Listar los datos de los estudiantes cuya edad es mayor al promedio de la edad de los estudiantes.

```
SELECT *
FROM ESTUDIANTE
WHERE edad > (SELECT AVG(edad)
                  FROM ESTUDIANTE);
```

Ejercicio 19 - Juegos de la NBA

Dado el siguiente modelo relacional, se pide:

EQUIPOS (NOMBRE, CIUDAD, CONFERENCIA, DIVISION)
JUGADORES (CODIGO, NOMBRE, PROCEDENCIA, ALTURA, PESO, POSICION, ***NOMBRE_EQUIPO***)
PARTIDOS (CODIGO, ***EQUIPO_LOCAL***, ***EQUIPO_VISITANTE***, PUNTOS_LOCAL, PUNTOS_VISITANTE, TEMPORADA)
ESTADISTICAS (TEMPORADA, *JUGADOR*, PUNTOS_POR_PARTIDO, ASISTENCIA_POR_PARTIDO, TAPONES_POR_PARTIDO, REBOTES_POR_PARTIDO)

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita.

1. Mostrar el nombre de todos los jugadores ordenados alfabéticamente.

```
SELECT NOMBRE
FROM JUGADORES
ORDER BY NOMBRE;
```

2. Mostrar el nombre de los jugadores cuya posición contenga la letra C y con más de 200 libras de peso, ordenados alfabéticamente.

```
SELECT NOMBRE
FROM JUGADORES
WHERE POSICION LIKE '%C%'
AND PESO > 200 ORDER BY NOMBRE;
```

3. Mostrar el nombre de todos los equipos ordenados alfabéticamente.

```
SELECT NOMBRE
FROM EQUIPOS
ORDER BY NOMBRE;
```

4. Mostrar el nombre de los equipos cuya conferencia sea este, ordenado por nombre.

```
SELECT NOMBRE
FROM EQUIPOS
WHERE CONFERENCIA = 'ESTE'
ORDER BY NOMBRE;
```

5. Mostrar los equipos donde su ciudad empieza por C, ordenado por nombre.

```
SELECT NOMBRE
FROM EQUIPOS
WHERE CIUDAD LIKE 'C%'
ORDER BY NOMBRE;
```

6. Mostrar todos los jugadores y su equipo ordenado por nombre del equipo.

```
SELECT J.NOMBRE, J.NOMBRE_EQUIPO
FROM JUGADORES J
ORDER BY J.NOMBRE_EQUIPO;
```

7. Mostrar todos los jugadores del equipo “Raptors” ordenado por nombre de jugador.

```
SELECT J.NOMBRE
FROM JUGADORES J
WHERE J.NOMBRE_EQUIPO = 'RAPTORS'
ORDER BY J.NOMBRE;
```

8. Mostrar los puntos por partido de ‘Luis Ruiz’.

```
SELECT E.PUNTOS_POR_PARTIDO
FROM ESTADISTICAS E, JUGADORES J
WHERE E.JUGADOR = J.CODIGO
AND J.NOMBRE = 'LUIS RUIZ';
```

9. Mostrar los puntos por partido de ‘Riu Morales’ en la temporada ’04/05’.

```
SELECT E.PUNTOS_POR_PARTIDO
FROM ESTADISTICAS E, JUGADORES J
WHERE E.JUGADOR = J.CODIGO
AND J.NOMBRE = 'RIU MORALES'
AND TEMPORADA = '04/05';
```

10. Mostrar el total de puntos de cada jugador en toda su carrera, ordenado por nombre del jugador.

```
SELECT J.NOMBRE, SUM(E.PUNTOS_POR_PARTIDO) AS PUNTOS
FROM JUGADORES J, ESTADISTICAS E
WHERE J.CODIGO = E.JUGADOR
GROUP BY J.NOMBRE ORDER BY J.NOMBRE;
```

11. Mostrar el número de jugadores de cada equipo.

```
SELECT J.NOMBRE_EQUIPO, COUNT(*)
FROM JUGADORES J
GROUP BY J.NOMBRE_EQUIPO;
```

12. Mostrar el jugador que más puntos ha realizado en toda su carrera.

```
SELECT T.NOMBRE
FROM ( SELECT J.NOMBRE, SUM(E.PUNTOS_POR_PARTIDOS) AS PUNTOS
        FROM JUGADORES J, ESTADISTICAS E
        WHERE J.CODIGO = E.JUGADOR
        GROUP BY J.NOMBRE
        ORDER BY J.NOMBRE) T
WHERE T.PUNTOS = (SELECT MAX(T2.PUNTOS)
                  FROM (SELECT J.NOMBRE,
                               SUM(E.PUNTOS_POR_PARTIDOS) AS PUNTOS
                        FROM JUGADORES J, ESTADISTICAS E
                        WHERE J.CODIGO = E.JUGADOR
                        GROUP BY J.NOMBRE) T2);
```

13. Mostrar el nombre del equipo, conferencia y división del jugador más alto de la NBA.

```
SELECT J.NOMBRE AS JUGADOR, E.NOMBRE AS EQUIPO, E.CONFERENCIA, E.DIVISION
FROM EQUIPOS E, JUGADORES J
WHERE E.NOMBRE = J.NOMBRE_EQUIPO
AND J.ALTURA = (SELECT MAX(ALTURA)
                FROM JUGADORES);
```

14. Mostrar la suma de los puntos por partido de todos los jugadores venezolanos que juegan en equipos de la ciudad de ‘Los Ángeles’.

```
SELECT SUM(PUNTOS_POR_PARTIDO) AS PUNTOS
FROM ESTADISTICAS
WHERE JUGADOR = (SELECT CODIGO
                 FROM JUGADORES
                 WHERE PROCEDENCIA = 'VENEZUELA'
                 AND NOMBRE_EQUIPO IN (SELECT NOMBRE
                                       FROM EQUIPOS
                                       WHERE CIUDAD = 'LOS ANGELES'));
```

15. Mostrar la media de puntos en partidos de los equipos de la división Pacific.

```
SELECT AVG(PUNTOS)
FROM (SELECT SUM(PUNTOS_LOCAL) AS PUNTOS
      FROM PARTIDOS
      WHERE EQUIPO_LOCAL IN (SELECT NOMBRE
                             FROM EQUIPOS
                             WHERE DIVISION = 'PACIFIC')
      UNION
      SELECT SUM(PUNTOS_VISITANTE) AS PUNTOS
      FROM PARTIDOS
      WHERE EQUIPO_VISITANTE IN (SELECT NOMBRE
                                 FROM EQUIPOS
                                 WHERE DIVISION = 'PACIFIC')) T;
```

16. Mostrar el partido o partidos (equipo local, equipo visitante y diferencia) con mayor diferencia de puntos.

```
SELECT EQUIPO_LOCAL, EQUIPO_VISITANTE, DIFERENCIA
FROM (SELECT  EQUIPO_LOCAL,
              EQUIPO_VISITANTE,
              (PUNTOS_LOCAL - PUNTOS_VISITANTE) AS DIFERENCIA
      FROM PARTIDOS)
WHERE DIFERENCIA = (SELECT MAX(PUNTOS_LOCAL - PUNTOS_VISITANTE)
                    FROM PARTIDOS);
```

17. Mostrar los puntos de cada equipo en los partidos, tanto de local como de visitante. Crear una vista.

```
CREATE OR REPLACE VIEW PUNTOS_EQUIPO_PARTIDOS AS
SELECT T.EQUIPO, SUM(T.PUNTOS) AS PUNTOS
FROM (SELECT EQUIPO_LOCAL AS EQUIPO,
        SUM(PUNTOS_LOCAL) AS PUNTOS
        FROM PARTIDOS
        GROUP BY EQUIPO_LOCAL
        UNION ALL
        SELECT EQUIPO_VISITANTE AS EQUIPO,
        SUM(PUNTOS_EQUIPO_VISITANTE) AS PUNTOS
        FROM PARTIDOS
        GROUP BY EQUIPO_EQUIPO_VISITANTE)
GROUP BY T.EQUIPO;
```

18. Mostrar quien gana en cada partido (código, equipo local, equipo visitante, equipo ganador), en caso de empate será null. Crear una vista.

```
CREATE OR REPLACE VIEW EQUIPOS_GANADORES AS
SELECT CODIGO,
       EQUIPO_LOCAL,
       EQUIPO_VISITANTE,
       CASE WHEN P.PUNTOS_LOCAL > P.PUNTOS_VISITANTE THEN EQUIPO_LOCAL
            WHEN P.PUNTOS_LOCAL < P.PUNTOS_VISITANTE THEN EQUIPO_VISIT
ELSE NULL END AS EQUIPO_GANADOR
FROM PARTIDOS P;
```

Ejercicio 20 - Proveedores y componentes [16]

Dado las siguientes tablas, se pide:

PROVEEDORES

P#	PNOMBRE	CATEGORIA	CIUDAD
P1	CARLOS	20	SEVILLA
P2	JUAN	10	MADRID
P3	JOSE	30	SEVILLA
P4	INMA	20	SEVILLA
P5	EVA	30	CACERES

COMPONENTES

C#	CNOMBRE	COLOR	PESO	CIUDAD
C1	X3A	ROJO	12	SEVILLA
C2	B85	VERDE	17	MADRID
C3	C4B	AZUL	17	MALAGA
C4	C4B	ROJO	14	SEVILLA
C5	VT8	AZUL	12	MADRID
C6	C30	ROJO	19	SEVILLA

ARTICULOS

T#	TNOMBRE	CIUDAD
T1	CLASIFICADORA	MADRID
T2	PERFORADORA	MALAGA
T3	LECTORA	CACERES
T4	CONSOLA	CACERES
T5	MEZCLADORA	SEVILLA
T6	TERMINAL	BARCELONA
T7	CINTA	SEVILLA

ENVIOS

P#	C#	T#	CANTIDAD
P1	C1	T1	200
P1	C1	T4	700
P2	C3	T1	400
P2	C3	T2	200
P2	C3	T3	200
P2	C3	T4	500
P2	C3	T5	600
P2	C3	T6	400
P2	C3	T7	800
P2	C5	T2	100
P3	C3	T1	200
P3	C4	T2	500
P4	C6	T3	300
P4	C6	T7	300
P5	C2	T2	200
P5	C2	T4	100
P5	C5	T4	500
P5	C5	T7	100
P5	C6	T2	200
P5	C1	T4	100
P5	C3	T4	200
P5	C4	T4	800
P5	C5	T5	400
P5	C6	T4	500

1. Obtener todos los detalles de todos los artículos de CACERES.

```
SELECT *
FROM artículos
WHERE ciudad = 'CACERES';
```

2. Obtener todos los valores de P# para los proveedores que abastecen el artículo T1.

```
SELECT DISTINCT p#
FROM envios
WHERE t# = 'T1';
```

3. Obtener la lista de pares de atributos (COLOR, CIUDAD) de la tabla componentes eliminando los pares duplicados.

```
SELECT DISTINCT color , ciudad
FROM componentes;
```

4. Obtener de la tabla de artículos los valores de T# y CIUDAD donde el nombre de la ciudad acaba en D o contiene al menos una E.

```
SELECT t#, ciudad
FROM artículos
WHERE ciudad LIKE '%D'
OR ciudad LIKE '%E%';
```

5. Obtener los valores de P# para los proveedores que suministran para el artículo T1 el componente C1.

```
SELECT p#
FROM envios
WHERE t# = 'T1'
AND c# = 'C1';
```

6. Obtener los valores de TNOMBRE en orden alfabético para los artículos abastecidos por el proveedor P1

```
SELECT tnombre
FROM artículos a, envios e
WHERE e.p# = 'P1'
AND e.t# = a.t#
ORDER BY tnombre;
```

7. Obtener los valores de C# para los componentes suministrados para cualquier artículo de MADRID.

```
SELECT DISTINCT c#
FROM envios
WHERE t# IN ( SELECT t#
              FROM artículos
              WHERE ciudad = 'MADRID');
```

8. Obtener todos los valores de C# de los componentes tales que ningún otro componente tenga un valor de peso inferior.

```
SELECT c#
FROM componentes
WHERE peso = (SELECT MIN(peso)
              FROM componentes );
```

9. Obtener los valores de P# para los proveedores que suministren los artículos T1 y T2.

```
SELECT p# AS codigo
FROM envios
WHERE t# = 'T1'
INTERSECT
SELECT p# AS codigo
FROM envios
WHERE t# = 'T2';
```

10. Obtener los valores de P# para los proveedores que suministran para un artículo de SEVILLA o MADRID un componente ROJO.

```
SELECT p#
FROM envios e, componentes c, artículos a
WHERE e.c# = c.c#
AND e.t# = a.t#
AND c.color = 'ROJO'
AND a.ciudad IN ('SEVILLA', 'MADRID');
```

11. Obtener, mediante subconsultas, los valores de C# para los componentes suministrados para algún artículo de SEVILLA por un proveedor de SEVILLA.

```
SELECT c#
FROM envios
WHERE t# IN (SELECT t#
             FROM artículos
             WHERE ciudad = 'SEVILLA')
AND p# IN (SELECT p#
           FROM proveedores
       WHERE ciudad = 'SEVILLA');
```

12. Obtener los valores de T# para los artículos que usan al menos un componente que se puede obtener con el proveedor P1.

```
SELECT DISTINCT t#
FROM envios
WHERE c# IN ( SELECT DISTINCT c#
              FROM envios
              WHERE p# = 'P1' );
```

13. Obtener todas las ternas (CIUDAD, C#, CIUDAD) tales que un proveedor de la primera ciudad suministre el componente especificado para un artículo montado en la segunda ciudad.

```
SELECT p.ciudad ,e.c#, a.ciudad
FROM envios e, proveedores p , artículos a
WHERE e.p# = p.p#
AND e.t# = a.t#;
```

14. Repetir el ejercicio anterior pero sin recuperar las ternas en los que los dos valores de ciudad sean los mismos.

```
SELECT p.ciudad ,c#, a.ciudad
FROM envios e, proveedores p , articulos a
WHERE e.p# = p.p#
AND e.t# = a.t#
AND p.ciudad <> a.ciudad;
```

15. Obtener el número de suministros, el de artículos distintos suministrados y la cantidad total de artículos suministrados por el proveedor P2.

```
SELECT COUNT(*) AS 'Número de Suministros',
       COUNT (DISTINCT t#) AS 'Artículos Suministrados',
       SUM(cantidad) AS 'Total Artículos Suministrados'
FROM envios
WHERE p# = 'P2';
```

16. Para cada artículo y componente suministrado obtener los valores de C#, T# y la cantidad total correspondiente.

```
SELECT c#,t#, SUM(cantidad) Total
FROM envios
GROUP BY c#,t#;
```

17. Obtener los valores de T# de los artículos abastecidos al menos por un proveedor que no viva en MADRID y que no esté en la misma ciudad en la que se monta el artículo.

```
SELECT DISTINCT e.t#
FROM envios e,articulos a
WHERE e.t# = a.t#
AND EXISTS (SELECT *
            FROM proveedores p
            WHERE p.ciudad != a.ciudad
             AND p.p# = e.p#
             AND p.ciudad != 'MADRID' );
```

18. Obtener los valores de P# para los proveedores que suministran al menos un componente suministrado al menos por un proveedor que suministra al menos un componente ROJO.

```
SELECT DISTINCT p#
FROM envios
WHERE c# IN (SELECT c#
             FROM envios
             WHERE p# IN ( SELECT e.p#
                           FROM envios e , componentes c
                           WHERE c.color = 'ROJO'
                           AND c.c# = e.c# ) );
```

19. Obtener los identificadores de artículos, T#, para los que se ha suministrado algún componente del que se haya suministrado una media superior a 320 artículos.

```
SELECT DISTINCT t#
FROM envios
WHERE c# IN ( SELECT c#
              FROM envios
              GROUP BY c#
              HAVING AVG(cantidad) > 320 );
```

20. Seleccionar los identificadores de proveedores que hayan realizado algún envío con Cantidad mayor que la media de los envíos realizados para el componente a que corresponda dicho envío.

```
SELECT DISTINCT p#
FROM envios a
WHERE cantidad > (SELECT AVG(cantidad)
                  FROM envios b
                  WHERE b.c# = a.c# );
```

21. Seleccionar los identificadores de componentes suministrados para el artículo 'T2' por el proveedor 'P2'.

```
SELECT c#
FROM envios
WHERE t# = 'T2'
AND p# = 'P2';
```

22. Seleccionar todos los datos de los envíos realizados de componentes cuyo color no sea 'ROJO'.

```
SELECT e.*
FROM envios e, componentes c
WHERE e.c# = c.c#
AND color <> 'ROJO';
```

23. Seleccionar los identificadores de componentes que se suministren para los artículos 'T1' y 'T2'.

```
SELECT c#
FROM envios
WHERE t# = 'T1'
INTERSECT
SELECT c#
FROM envios
WHERE t# = 'T2';
```

24. Seleccionar el identificador de proveedor y el número de envíos de componentes de color 'ROJO' llevados a cabo por cada proveedor.

```
SELECT p#, COUNT(*) A
FROM envios
WHERE c# IN ( SELECT c#
              FROM componentes
              WHERE color = 'ROJO')
GROUP BY p# ;
```

25. Seleccionar los colores de componentes suministrados por el proveedor 'P1'.

```
SELECT DISTINCT color
FROM componentes
WHERE c# IN ( SELECT DISTINCT c#
              FROM envios
              WHERE p# = 'P1');
```

26. Seleccionar los datos de envío y nombre de ciudad de aquellos envíos que cumplan que el artículo, proveedor y componente son de la misma ciudad.

```
SELECT e.*, c.ciudad
FROM envios e, componentes c, articulos a, proveedores p
WHERE e.t# = a.t#
AND e.c# = c.c#
AND e.p# = p.p#
AND p.ciudad = c.ciudad
AND p.ciudad = a.ciudad;
```

27. Seleccionar los nombres de los componentes que son suministrados en una cantidad total superior a 500.

```
SELECT DISTINCT cnombre
FROM componentes
WHERE c# IN ( SELECT  c#
              FROM envios
              GROUP BY c#
              HAVING SUM(cantidad) > 500);
```

28. Seleccionar los identificadores de proveedores que residan en Sevilla y no suministren más de dos artículos distintos.

```
SELECT p#
FROM proveedores
WHERE ciudad = 'SEVILLA'
MINUS
SELECT p#
FROM envios
GROUP BY p#
HAVING COUNT(DISTINCT t#) > 2;
```

29. Seleccionar los identificadores de artículos para los cuales todos sus componentes se fabrican en una misma ciudad.

```
SELECT t#
FROM envios e, componentes c
WHERE e.c# = c.c#
GROUP BY t#
HAVING COUNT(DISTINCT ciudad) = 1 ;
```

30. Seleccionar los identificadores de artículos para los que se provean envíos de todos los componentes existentes en la base de datos.

```
SELECT t#
FROM Envíos
GROUP BY t#
HAVING COUNT(DISTINCT c#) = ( SELECT COUNT(*)
                              FROM Componentes) ;
```

31. Seleccionar los códigos de proveedor y artículo que suministran al menos dos componentes de color 'ROJO'.

```
SELECT p#, t#
FROM envios e,componentes c
WHERE e.c# = c.c#
AND c.color = 'ROJO'
GROUP BY p#, t#
HAVING COUNT (*) > 1;
```

Ejercicio 21 -Venta de Productos [18]

Dado el siguiente modelo relacional de una base de datos, se pide:

CLIENTE (codigo ,nombre, domicilio, provincia)
PRODUCTO (codigo_producto, nombre_producto)
ITEM_VENTAS (***número_factura***, ***codigo_producto***, cantidad, precio)
VENTAS (numero_factura, ***codigo_cliente***, fecha)

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita.

1. Obtener el nombre y el domicilio de los clientes que viven en la provincia de Misiones.

```sql
SELECT Nombre, Domicilio
FROM Cliente
WHERE Provincia = 'Misiones';
```

2. Obtener el nombre, domicilio y provincia de los clientes que viven en la provincia de Misiones o de Río Negro.

```sql
SELECT Nombre, Domicilio, Provincia
FROM Cliente
WHERE Provincia = 'Misiones'
OR Provincia = 'Rio Negro';
```

3. Obtener el importe total en pesos por factura y producto, especificando el número de factura, el código del producto y el importe total .

```sql
SELECT numero_factura, codigo_producto, (precio * cantidad) AS Total
FROM item_ventas;
```

4. Sobre la consulta 3, obtener solo el importe total para el producto a.

```sql
SELECT numero_factura, codigo_producto, (precio * cantidad) AS Total
FROM item_ventas
WHERE codigo_producto = 'a';
```

5. Sobre la consulta 3, obtener solo el importe total para las facturas mayores iguales a 2 y menores iguales a 5 y solo para el producto código c.

```
SELECT numero_factura, codigo_producto, (precio * cantidad) AS Total
FROM item_ventas
WHERE numero_factura BETWEEN 2 AND 5
AND codigo_producto = 'c';
```

6. Sobre la consulta 3, obtener solo el importe total para los registros cuyo importe total sea mayor a 200.

```
SELECT numero_factura, codigo_producto, (precio * cantidad) AS Total
FROM item_ventas
WHERE cantidad * precio > 200;
```

7. Obtener un listado de las facturas realizadas especificando número de factura, nombre del producto y cantidad vendida.

```
SELECT numero_factura, nombre_producto, cantidad
FROM Producto AS p, item_ventas AS iv
WHERE iv.codigo_producto = p.codigo_producto;
```

8. Obtener un listado de las facturas realizadas cuya cantidad sea mayor igual a 15 especificando número de factura, nombre del producto y cantidad vendida.

```
SELECT numero_factura, nombre_producto, cantidad
FROM Producto AS p, item_ventas AS iv
WHERE iv.codigo_producto = p.codigo_producto
AND cantidad >= 15;
```

9. Obtener un listado de las facturas realizadas indicando número de factura, nombre del cliente, nombre del producto, cantidad y precio y el importe total.

```
SELECT item_ventas.Numero_factura, nombre_cliente, nombre_producto,
       cantidad, precio, (cantidad * precio) AS Total
FROM Cliente, Ventas, Item_Ventas, Producto
WHERE Cliente.codigo_cliente = Ventas.codigo_cliente
AND Ventas.numero_factura = Item_Ventas.Numero_factura
AND Item_Ventas.codigo_producto = Producto.codigo_producto;
```

10. Obtener la cantidad de unidades máxima.

```
SELECT MAX(cantidad) AS Cantidad
FROM item_ventas;
```

11. Obtener la cantidad total de unidades vendidas del producto c.

```
SELECT SUM(cantidad) AS TOTAL
FROM item_ventas
WHERE codigo_producto = 'c';
```

12. Cantidad de unidades vendidas por producto, indicando la descripción del producto, ordenado de mayor a menor por las cantidades vendidas.

```
SELECT nombre_producto AS Producto, SUM(cantidad) AS Cantidad
FROM Producto AS p, item_ventas AS iv
WHERE iv.codigo_producto = p.codigo_producto
GROUP BY nombre_producto
ORDER BY SUM(cantidad) DESC;
```

13. Cantidad de unidades vendidas por producto, indicando la descripción del producto, ordenado alfabéticamente por nombre de producto para los productos que vendieron más de 30 unidades.

```
SELECT nombre_producto AS Producto, sum(cantidad) AS Cantidad
FROM Producto AS p, item_ventas AS iv
WHERE iv.codigo_producto = p.codigo_producto
GROUP BY nombre_producto
HAVING SUM(cantidad) > 30
ORDER BY nombre_producto;
```

14. Obtener cuantas compras (1 factura = 1 compra) realizo cada cliente indicando el código y nombre del cliente ordenado de mayor a menor.

```
SELECT v.codigo_cliente, nombre_cliente, COUNT(numero_factura) AS compras
FROM cliente AS c, ventas AS v
WHERE v.codigo_cliente = c.codigo_cliente
GROUP BY v.codigo_cliente, nombre_cliente
ORDER BY COUNT(numero_factura) DESC;
```

15. Promedio de unidades vendidas por producto, indicando el código del producto para el cliente 1.

```
SELECT codigo_producto, AVG(cantidad) AS promedio
FROM Ventas AS v, item_ventas AS iv
WHERE iv.numero_factura = v.numero_factura
AND codigo_cliente = 1
GROUP BY codigo_producto;
```

Ejercicio 22 - Oficina ferretera [17]

Dado el siguiente modelo relacional, se pide:

EMPLEADO (numemp, nombre, edad, ***oficina***, empleado, titulo, fechacontratado, ***jefe***, cuota, ventas)
OFICINA (oficina, ciudad, region, dir, objetivo, ventas)
CLIENTE (numclie, nombre, ***repclie***, limitecredito)
PRODUCTO (idfab, idproducto, descripcion, precio, existencias)
PEDIDO (codigo, numpedido, fechapedido, ***clie, rep, fab, producto***, cant, importe)

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita.

1. Listar el nombre, oficina y fecha de contrato de todos los empleados.

```
SELECT nombre, oficina, fechacontratado
FROM empleado;
```

2. Obtener un listado de los empleados por orden de antigüedad en la empresa empezando por los más recientemente incorporados.

```
SELECT nombre, numemp, fechacontratado
FROM empleado
ORDER BY fechacontratado DESC;
```

3. Obtener un listado de los empleados ordenados por volumen de ventas sacando primero los de mayores ventas.

```
SELECT nombre, numemp, ventas
FROM empleado
ORDER BY ventas DESC;
```

4. Muestra las ventas de cada oficina , ordenadas por orden alfabético de región y dentro de cada región por ciudad.

```
SELECT region, ciudad, ventas
FROM oficina
ORDER BY region, ciudad;
```

5. Lista las oficinas clasificadas por región y dentro de cada región por ganancia (ventas - objetivo) de modo que las de mayor ganancia aparezcan las primeras.

```
SELECT region, ciudad, (ventas - objetivo) AS ganancia
FROM oficina
ORDER BY region, 3 DESC;
```

6. Lista el nombre de los empleados de la oficina 12 que tengan más de 30 años.

```
SELECT nombre
FROM empleado
WHERE oficina = 12
AND edad > 30;
```

7. Listar los empleados cuyas ventas superan su cuota.

```
SELECT numemp, nombre
FROM empleado
WHERE ventas > cuota;
```

8. Lista las oficinas cuyas ventas estén por debajo del 80% de su objetivo.

```
SELECT oficina
FROM oficina
WHERE ventas < objetivo * 0.8;
```

9. Lista los empleados cuyas ventas estén comprendidas entre 100.000 y 500.00

```
SELECT numemp, nombre
FROM empleado
WHERE ventas BETWEEN 100000 AND 500000;
```

10. Lista los empleados de las oficinas 12, 14 y 16.

```
SELECT numemp, nombre, oficina
FROM empleado
WHERE oficina IN (12,14,16);
```

11. Lista las oficinas que no tienen director.

```
SELECT oficina, ciudad
FROM oficina
WHERE dir IS NULL;
```

12. Lista los empleados cuyo nombre empiece por Luis.

```
SELECT numemp, nombre
FROM empleado
WHERE nombre LIKE 'Luis%';
```

13. Obtener una lista de todos los productos, indicando para cada uno su idfab, idproducto, descripción, precio y precio con IVA (es el precio aumentado en un 16%).

```
SELECT idfab, idproducto, descripcion, precio,
(precio * 1.16) AS iva_incluido
FROM productos;
```

14. De cada pedido se quiere saber el número de pedido, fab, producto, cantidad, precio unitario e importe.

```
SELECT numpedido, fab, producto, cant,
(importe / cant) AS precio_unitario, importe
FROM pedidos;
```

15. Obtener las oficinas ordenadas por orden alfabético de región y dentro de cada región por ciudad, si hay más de una oficina en la misma ciudad, aparecerá primero la que tenga el número de oficina mayor.

```
SELECT *
FROM oficinas
ORDER BY region, ciudad, oficina DESC;
```

16. Obtener los pedidos ordenados por fecha de pedido.

```
SELECT *
FROM pedidos
ORDER BY fechapedido;
```

17. Listar las cuatro líneas de pedido más caras (las de mayor importe).

```
SELECT TOP 4 *
FROM pedidos
ORDER importe DESC;
```

18. Obtener de los pedidos: fab, producto, cantidad, precio unitario e importe, pero sacando únicamente las 5 líneas de pedido de menor precio unitario.

```
SELECT TOP 5 numpedido, fab, producto, cant,
(importe / cant) AS precio_unitario, importe
FROM pedidos
ORDER BY 5;
```

19. Listar los números de los empleados que tienen una oficina asignada.

```
SELECT numemp
FROM empleados
WHERE oficina IS NOT NULL;
```

20. Listar los números de las oficinas que no tienen director.

```
SELECT oficina
FROM oficinas
WHERE dir IS NULL;
```

21. Listar los datos de las oficinas de las regiones del norte y del este (tienen que aparecer primero las del norte y después las del este).

```
SELECT *
FROM oficinas
WHERE region IN ('norte','este')
ORDER BY region DESC;
```

22. Listar los empleados de nombre Julia.

```
SELECT *
FROM empleados
WHERE nombre LIKE 'Julia%';
```

23. Listar los productos cuyo idproducto acabe en x.

```
SELECT *
FROM productos
WHERE idproducto LIKE '%x';
```

24. Listar las oficinas del este indicando para cada una de ellas su número, ciudad, números y nombres de sus empleados. Hacer una versión en la que aparecen sólo las que tienen empleados, y hacer otra en las que aparezcan las oficinas del este que no tienen empleados.

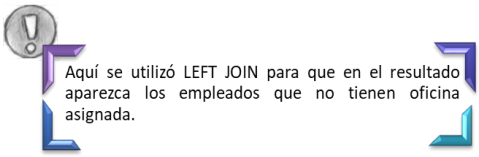
```
SELECT oficinas.oficina, ciudad, numemp, nombre
FROM oficinas INNER JOIN empleados
     ON oficinas.oficina = empleados.oficina
WHERE region = 'este';

SELECT oficinas.oficina, ciudad, numemp, nombre
FROM oficinas LEFT JOIN empleados
     ON oficinas.oficina = empleados.oficina
WHERE region = 'este';
```

25. Listar los pedidos mostrando su número, importe, nombre del cliente, y el límite de crédito del cliente correspondiente (todos los pedidos tienen cliente y representante).

```
SELECT numpedido, importe, clientes.nombre AS cliente, limitecredito
FROM pedidos INNER JOIN clientes
ON pedidos.clie = clientes.numclie;
```

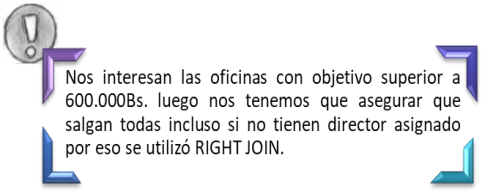
26. Listar los datos de cada uno de los empleados, la ciudad y región en donde trabaja.

```
SELECT empleados.*, ciudad, region
FROM empleados LEFT JOIN oficinas
ON empleados.oficina = oficinas.oficina;
```



27. Listar las oficinas con objetivo superior a 600.000 Bs. indicando para cada una de ellas el nombre de su director.

```
SELECT oficinas.*, nombre AS director
FROM empleados RIGHT JOIN oficinas
ON empleados.numemp = oficinas.dir
WHERE objetivo > 600000;
```



28. Listar los pedidos superiores a 25.000 Bs, incluyendo el nombre del empleado que tomó el pedido y el nombre del cliente que lo solicitó.

```
SELECT numpedido, importe,
       empleados.nombre AS representante,
       clientes.nombre AS cliente
FROM pedidos
INNER JOIN clientes ON pedidos.clie = clientes.numclie
INNER JOIN empleados ON pedidos.rep = empleados.numemp
WHERE importe > 25000;
```

29. Hallar los empleados que realizaron su primer pedido el mismo día en que fueron contratados.

```
SELECT empleados.*
FROM empleados INNER JOIN pedidos
ON pedidos.rep = empleados.numemp
WHERE fechapedido = contrato;
```

30. Listar los empleados con una cuota superior a la de su jefe; para cada empleado sacar sus datos y el número, nombre y cuota de su jefe.

```
SELECT empleados.*, jefes.numemp AS num_jefe,
jefes.nombre AS nombre_jefe, jefes.cuota AS cuota_jefe
FROM empleados INNER JOIN empleados jefes
ON empleados.jefe = jefes.numemp
WHERE empleados.cuota > jefes.cuota;
```

31. Listar los códigos de los empleados que tienen una línea de pedido superior a 10.000 Bs o que tengan una cuota inferior a 10.000 Bs.

```
SELECT rep
FROM pedidos
WHERE importe > 10000
UNION
SELECT numemp
FROM empleados
WHERE cuota < 10000;
```

32. Obtener el importe total de ventas de todos los empleados y el mayor objetivo de las oficinas asignadas a los empleados:

```
SELECT SUM(empleados.ventas), MAX(objetivo)
FROM empleados LEFT JOIN oficinas
ON empleados.oficina = oficinas.oficina;
```

33. Obtener de cada precio unitario el importe total vendido.

```
SELECT importe/cant , SUM(importe)
FROM pedidos
GROUP BY importe/cant;
```

34. ¿Cuál es la cuota media y las ventas medias de todos los empleados?

```
SELECT AVG(cuota) AS cuota_media,
       AVG(ventas) AS ventas_media
FROM empleados;
```

35. Hallar el importe medio de pedidos, el importe total de pedidos y el precio medio de venta (el precio de venta es el precio unitario en cada pedido).

```
SELECT AVG(importe) AS importe_medio,
SUM(importe) AS importe_total,
AVG(importe/cant) AS precio_venta_medio
FROM pedidos;
```

36. Hallar el precio medio de los productos del fabricante ACI.

```
SELECT AVG(precio) AS p_medio_ACI
FROM productos
WHERE idfab = 'ACI';
```

37. ¿ Cuál es el importe total de los pedidos realizados por el empleado Pedro Roriz?

```
SELECT SUM(importe) AS total_pedidos_V_Pantalla
FROM empleados INNER JOIN pedidos
ON empleados.numemp = pedidos.rep
WHERE nombre = 'Pedro Roriz';
```

38. Hallar en la fecha en la que se realizó el primer pedido (suponiendo que en la tabla de pedidos se tienen todos los pedidos realizados hasta la fecha).

```
SELECT MIN(fechapedido) AS primer_pedido
FROM pedidos ;
```

39. Hallar cuántos pedidos hay de más de 250.000 Bs.

```
SELECT COUNT(*) AS cuantos_pedidos_mayores
FROM pedidos
WHERE importe > 250000;
```

40. Listar cuántos empleados están asignados a cada oficina, indicar el número de oficina y cuántos hay asignados.

```
SELECT oficina, COUNT(*) AS cuantos_empleados
FROM empleados
GROUP BY oficina;
```

41. Para cada empleado, obtener su número, nombre e importe vendido por ese empleado a cada cliente indicando el número de cliente.

```
SELECT numemp, nombre, clie AS cliente,
SUM(importe) AS total_vendido
FROM empleados INNER JOIN pedidos
ON pedidos.rep = empleados.numemp
GROUP BY numemp, nombre, clie;
```

42. Para cada empleado cuyos pedidos suman más de 300.000 Bs, hallar su importe medio de pedidos. En el resultado indicar el número de empleados y su importe medio de pedidos.

```
SELECT rep, AVG(importe) AS importe_medio
FROM pedidos
GROUP BY rep
HAVING SUM(importe) > 300000;
```

43. Listar de cada producto, su descripción, precio y cantidad total pedida, incluyendo sólo los productos cuya cantidad total pedida sea superior al 75% del stock y ordenado por cantidad total pedida.

```
SELECT descripcion, precio, SUM(importe) AS total_pedido
FROM productos INNER JOIN pedidos ON pedidos.fab = productos.idfab AND
     pedidos.producto = productos.idproducto
GROUP BY idfab, idproducto, descripcion, precio, existencias
HAVING SUM(importe) > existencias * 0.75
ORDER BY 3;
```

44. ¿Cuántas oficinas tienen empleados con ventas superiores a su cuota, no se quiere saber cuáles sino cuántas hay?

```
SELECT COUNT(*) AS cuantas_oficinas
FROM (SELECT DISTINCT oficina
       FROM empleados
       WHERE ventas > cuota);
```

45. Listar el número y nombre de los empleados cuya fecha de contrato sea igual a la primera fecha de todos los pedidos de la empresa.

```
SELECT numemp, nombre
FROM empleados
WHERE contrato = (SELECT MIN(fechapedido) FROM pedidos);
```

46. Listar los nombres de los clientes que tienen asignado el representante Alvaro Jaumes (suponiendo que no pueden haber representantes con el mismo nombre).

```
SELECT nombre
FROM clientes
WHERE repclie = (SELECT numemp
                 FROM empleados
                 WHERE nombre = 'Alvaro Jaumes');
```

47. Listar los vendedores (numemp, nombre, y nº de oficina) que trabajan en oficinas "buenas" (las que tienen ventas superiores a su objetivo).

Solución 1:

```
SELECT numemp, nombre, oficina
FROM empleados
WHERE oficina IN ( SELECT oficina
                   FROM oficinas
                   WHERE ventas > objetivo );
```

Solución 2:

```
SELECT numemp, nombre, oficina
FROM empleados
WHERE EXISTS (SELECT *
              FROM oficinas
              WHERE empleados.oficina = oficinas.oficina
              AND ventas > objetivo );
```

Solución 3:

```
SELECT numemp, nombre, oficina
FROM empleados
WHERE oficina = ANY (SELECT oficina
                     FROM oficinas
                     WHERE ventas > objetivo );
```

48. Listar los vendedores que no trabajan en oficinas dirigidas por el empleado 108.

```
SELECT numemp, nombre, oficina
FROM empleados
WHERE oficina NOT IN (SELECT oficina
                      FROM oficinas
                      WHERE dir = 108);
```

49. Listar los productos (idfab, idproducto y descripción) para los cuales no se ha recibido ningún pedido de 25000 Bs o más.

```
SELECT idfab, idproducto, descripcion
FROM productos
WHERE NOT EXISTS (SELECT *
                  FROM pedidos
                  WHERE fab = idfab
                  AND producto = idproducto
                  AND importe >= 25000);
```

50. Listar los clientes asignados a Ana Bustamante que no han remitido un pedido superior a 3000 Bs.

```
SELECT numclie, nombre
FROM clientes
WHERE repclie IN ( SELECT numemp
                   FROM empleados
                   WHERE nombre = 'Ana Bustamante' )
AND numclie NOT IN ( SELECT clie
                   FROM pedidos
                   WHERE importe > 3000
                   AND clie IS NOT NULL);
```

51. Listar las oficinas en donde haya un vendedor cuyas ventas representan más del 55% del objetivo de su oficina.

```
SELECT *
FROM oficinas
WHERE EXISTS (SELECT *
              FROM empleados
              WHERE empleados.oficina = oficinas.oficina
              AND ventas > objetivo * 0.55);
```

52. Listar las oficinas en donde todos los vendedores tienen ventas que superan al 50% del objetivo de la oficina.

Solución 1:

```
SELECT *
FROM oficinas
WHERE (objetivo * 0.5) <= ALL ( SELECT ventas
                                FROM empleados
                                WHERE       empleados.oficina       =
                                    oficinas.oficina
                                AND ventas IS NOT NULL);
```

Solución 2:

```
SELECT *
FROM oficinas
WHERE (objetivo * .5) <= (SELECT MIN(ventas)
                          FROM empleados
                          WHERE empleados.oficina = oficinas.oficina);
```

53. Listar las oficinas que tengan un objetivo mayor que la suma de las cuotas de sus vendedores.

```
SELECT *
FROM oficinas
WHERE objetivo > ( SELECT SUM(cuota)
                   FROM empleados
                   WHERE empleados.oficina = oficinas.oficina);
```

54. Subir un 5% el precio de todos los productos del fabricante ACI.

```
UPDATE productos
SET precio = precio * 1.05
WHERE idfab = 'ACI';
```

55. Añadir una oficina para la ciudad de Caracas, con el número de oficina 30, con un objetivo de 100000 y región Centro.

Solución 1:

```
INSERT INTO oficinas ( oficina, region, ciudad, objetivo )
VALUES ( 30, 'centro','Madrid', 100000 );
```

Solución 2:

```
INSERT INTO oficinas (oficina,region,ciudad,dir,objetivo,ventas)
VALUES (30, 'centro', 'Madrid', null, 100000,0) ;
```

Solución 3:

```
INSERT INTO oficinas
VALUES (30, 'Madrid', 'centro', null, 100000,0) ;
```

56. Cambiar los empleados de la oficina 21 a la oficina 30.

```
UPDATE empleados
SET oficina = 30
WHERE oficina = 21;
```

57. Eliminar los pedidos del empleado 105.

```
DELETE FROM pedidos WHERE rep = 105;
```

58. Eliminar las oficinas que no tengan empleados.

```
DELETE FROM oficinas
WHERE oficina NOT IN (SELECT oficina
                      FROM empleados
                      WHERE oficina IS NOT NULL) ;
```

59. Recuperar los precios originales a partir de la tabla nuevosproductos.

```
UPDATE productos INNER JOIN nuevaproductos
ON ( productos.idfab = nuevaproductos.idfab) AND (productos.idproducto
    = nuevaproductos.idproducto)
SET productos.precio = nuevaproductos.precio;
```

60. Recuperar las oficinas borradas a partir de la tabla nuevaoficinas.

```
INSERT INTO oficinas
SELECT * FROM nuevaoficinas
WHERE oficina NOT IN (SELECT oficina FROM oficinas);
```

61. Recuperar los pedidos borrados, a partir de la tabla nuevapedidos.

```
INSERT INTO pedidos
SELECT * FROM nuevapedidos WHERE rep = 105;
```

62. A los empleados de la oficina 30 asignarles la oficina 21.

```
UPDATE empleados
SET oficina = 21 WHERE oficina = 30;
```

63. Crear la tabla empleados.

```
CREATE TABLE empleados (
       numemp INTEGER PRIMARY KEY,
       nombre VARCHAR(30) NOT NULL,
       edad INTEGER,
       oficina INTEGER,
       titulo VARCHAR(20),
       contrato DATE NOT NULL,
       director INTEGER,
       cuota NUMERIC,
       ventas NUMERIC);
```

64. Crear la tabla oficinas y establecer la relación con empleado director.

```
CREATE TABLE oficinas (
       oficina INT PRIMARY KEY,
       ciudad VARCHAR(30),
       region VARCHAR(20),
       dir INTEGER CONSTRAINT cf_dir REFERENCES empleados,
       objetivo NUMERIC,
       ventas NUMERIC );
```

65. Crear la tabla productos.

```
CREATE TABLE productos (
       idfab VARCHAR(10),
       idproducto VARCHAR(20),
       descripcion VARCHAR(30) NOT NULL,
       precio NUMERIC NOT NULL,
       existencias INTEGER ,
       CONSTRAINT cp PRIMARY KEY (idfab,idproducto) );
```

66. Crear la tabla clientes sin el atributo limitecredito.

```
CREATE TABLE clientes (
       numclie INTEGER ,
       nombre VARCAHAR(30) NOT NULL,
       repclie INT CONSTRAINT cf_repclie REFERENCES empleados,
       CONSTRAINT cp PRIMARY KEY (numclie) ) ;
```

67. Añadir a la definición de clientes la columna limitecredito.

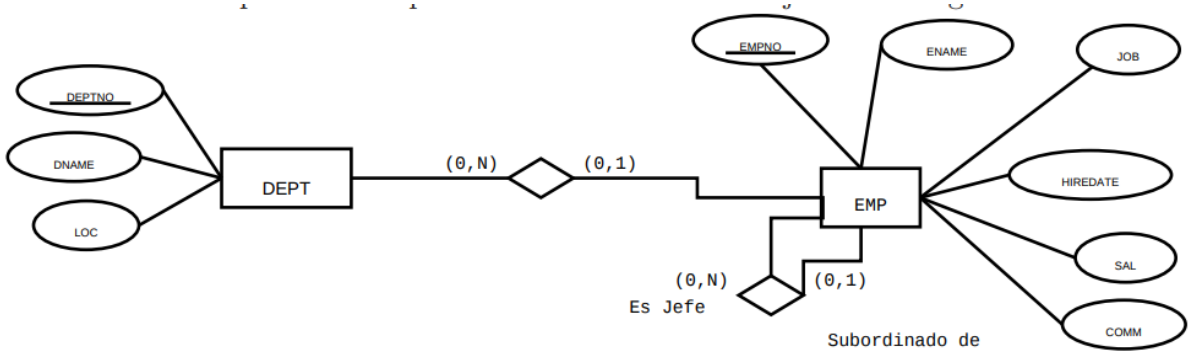
```
ALTER TABLE clientes
ADD COLUMN limitecredito NUMERIC;
```

68. Definir un índice sobre la columna región de la tabla oficinas.

```
CREATE INDEX i_region ON oficinas (region) ;
```

Ejercicio 23 - Empresa [19]

Dado el siguiente modelo entidad relación, se pide:



COMM: Si tiene un valor nulo, indica que el empleado no tiene comisión.

MGR: Si tiene un nulo, indica que el empleado no tiene jefe.

1. Obtener todos los datos de todos los empleados.

```
SELECT *
FROM EMP;
```

2. Obtener todos los datos de todos los departamentos.

```
SELECT *
FROM DEPT;
```

3. Obtener todos los datos de los administrativos (su trabajo es, en inglés, 'CLERK').

```
SELECT *
FROM EMP
WHERE JOB='CLERK';
```

4. Obtener el nombre y empleo de todos los empleados, ordenado primero por su trabajo y luego por su salario.

```
SELECT ENAME, JOB
FROM EMP
ORDER BY JOB, SAL;
```

5. Obtener los salarios y las comisiones de los empleados del departamento 30, pero ordenado por comisión.

```
SELECT SAL, COMM
FROM EMP
WHERE DEPTNO = 30
ORDER BY COMM;
```

6. Obtener las comisiones de los empleados de forma que no se repitan.

```
SELECT DISTINCT COMM
FROM EMP;
```

7. Obtener el nombre de empleado y su comisión SIN FILAS repetidas.

```
SELECT DISTINCT ENAME, COMM
FROM EMP;
```

8. Obtener los nombres de los empleados y sus salarios, de forma que no se repitan filas.

```
SELECT DISTINCT ENAME, SAL
FROM EMP;
```

9. Obtener las comisiones de los empleados y sus números de departamento, de forma que no se repitan filas.

```
SELECT DISTINCT COMM, DEPTNO
FROM EMP;
```

10. Listar los empleados que tienen una comisión superior a la mitad de su salario.

```
SELECT ENAME
FROM EMP
WHERE COMM > SAL/2;
```

11. Listar los empleados que no tienen comisión, o que la tengan menor o igual que el 25 % de su salario.

```
SELECT ENAME
FROM EMP
WHERE COMM IS NULL
OR COMM <= (0.25 * SAL);
```

12. Listar el código, salario y comisión de los empleados cuyo código sea mayor que 7500.

```
SELECT EMPNO, SAL, COMM
FROM EMP
WHERE EMPNO > 7500;
```

13. Obtener el salario, comisión y salario total (salario+comisión) de los empleados con comisión, ordenando el resultado por número de empleado.

```
SELECT EMPNO, SAL, COMM, SAL+COMM
FROM EMP
WHERE COMM IS NOT NULL
ORDER BY EMPNO;
```

14. Muestra el nombre de los empleados que, teniendo un salario superior a 1000, tengan como jefe al empleado cuyo c´odigo es 7698.

```
SELECT ENAME
FROM EMP
WHERE SAL > 1000
AND MGR = 7698;
```

15. Indicar para cada empleado el porcentaje que supone su comisión sobre su salario, ordenando el resultado por el nombre del mismo.

```
SELECT ENAME, (COMM / SAL) * 100
FROM EMP
ORDER BY ENAME;
```

16. Listar los empleados del departamento 10 cuyo nombre no contiene la cadena LA.

```
SELECT ENAME
FROM EMP
WHERE DEPTNO = 10
AND ENAME NOT LIKE '%LA%';
```

17. Obtener los empleados que no son supervisados por ningún otro.

```
SELECT ENAME
FROM EMP
WHERE MGR IS NULL;
```

18. Obtener los nombres de los departamentos que no sean Ventas (SALES) ni de investigación (RESEARCH). Ordena el resultado por la localidad del departamento.

```
SELECT DNAME
FROM DEPT
WHERE DNAME NOT IN ('SALES','RESEARCH')
ORDER BY LOC;
```

19. Listar el nombre de los empleados y el código del departamento de los administrativos (CLERK) que no trabajan en el departamento 10, y cuyo salario es superior a 800 Bs, ordenado por fecha de contratación.

```
SELECT ENAME, DEPTNO
FROM EMP
WHERE JOB = 'CLERK'
AND DEPTNO <> 10
AND SAL > 800
ORDER BY HIREDATE;
```

20. Para los empleados que tengan comisión, obtener sus nombres y el cociente entre su salario y su comisión (excepto cuando la comisión sea cero), ordenando el resultado por nombre.

```
SELECT ENAME, SAL/COMM
FROM EMP
WHERE COMM > 0
ORDER BY 1;
```

21. Listar toda la información sobre los empleados cuyo nombre completo tenga exactamente 5 caracteres.

```
SELECT *
FROM EMP
WHERE ENAME LIKE '_____';
```

22. Igual a la 21, pero para los empleados cuyo nombre tenga al menos cinco letras.

```
SELECT *
FROM EMP
WHERE ENAME LIKE '_____%';
```

23. Hallar los datos de los empleados que, o bien su nombre empieza por A y su salario es superior a 1000, o bien reciben comisión y trabajan en el departamento 30.

```
SELECT *
FROM EMP
WHERE (ENAME LIKE 'A%' AND SAL > 1000)
OR (COMM IS NOT NULL AND DEPTNO = 30);
```

24. Listar el nombre, el salario y el sueldo total de todos los empleados, ordenando el resultado primero por salario y luego por el sueldo total.

```
SELECT ENAME, SAL
FROM EMP ORDER BY 2,3;
```

25. Obtener el nombre, salario y la comisión de los empleados que perciben un salario que está entre la mitad de la comisión y la propia comisión.

```
SELECT ENAME, SAL,COMM
FROM EMP
WHERE SAL BETWEEN COMM/2 AND COMM;
```

26. Listar los nombres y empleos de aquellos empleados cuyo empleo acaba en MAN y cuyo nombre empieza por A.

```
SELECT ENAME, JOB
FROM EMP
WHERE JOB LIKE '%MAN'
AND ENAME LIKE 'A%';
```

27. Listar los nombres y fecha de contratación de aquellos empleados que no son vendedores (SALESMAN).

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE JOB <> 'SALESMAN';
```

28. Para los empleados que tengan como jefe a un empleado con código mayor que el suyo, obtener los que reciben de salario más de 1000 Bs y menos de 2000 Bs, o que están en el departamento 30.

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE MGR > EMPNO
AND (SAL BETWEEN 1000 AND 2000 OR DEPTNO = 30);
```

29. Obtener los empleados que trabajan en Caracas o Valencia.

```
SELECT ENAME,LOC
FROM EMP E INNER JOIN DEPT D
ON E.DEPTNO = D.DEPTNO
WHERE LOC IN ('Caracas','Valencia');
```

30. Para cada empleado obtener el nombre del departamento donde trabaja.

```
SELECT ENAME,DNAME
FROM EMP E INNER JOIN DEPT D
ON E.DEPTNO = D.DEPTNO;
```

31. Listar los datos de los empleados del departamento de ventas (’SALES’).

```
SELECT EMPNO, ENAME, JOB, HIREDATE, SAL, COMM
FROM EMP E INNER JOIN DEPT D
ON E.DEPTNO = D.DEPTNO
WHERE DNAME = 'SALES';
```

Ejercicio 24 - Parque Temático [20]

Dado la siguiente definición, se pide:

CREATE TABLE ESCUELA (CODIGO_ESCUELA CHAR(10) PRIMARY KEY, NOMBRE_ESCUELA VARCHAR(30) NOT NULL, DOMICILIO_ESCUELA TEXT(30) NOT NULL);	CREATE TABLE GUIA (CODIGO_GUIA CHAR(10) PRIMARY KEY, NOMBRE_GUIA VARCHAR(20) NOT NULL, APELLIDO_GUIA VARCHAR(20) NOT NULL);
CREATE TABLE RESERVA (NUMERO_RESERVA CHAR (10) PRIMARY KEY, FECHA_VISITA_RESERVADA DATETIME NOT NULL, HORA_VISITA_RESERVADA DATETIME NOT NULL);	CREATE TABLE TIPO_VISITA (CODIGO_TIPO_VISITA CHAR(10) PRIMARY KEY, DESCRIPCIÓN_VISITA VARCHAR(30), ARANCEL_POR_ALUMNO MONEY);
CREATE TABLE Telefono_Escuela(telefono_escuela INT, codigo_escuela CHAR(10) CONSTRAINT codigo_escuela REFERENCES Escuela, CONSTRAINT cp PRIMARY KEY (telefono_escuela, codigo_escuela));	CREATE TABLE Reserva_Por_Grado (numero_reserva CHAR(10) CONSTRAINT numero_reserva REFERENCES Reserva, codigo_tipo_visita CHAR(10) CONSTRAINT Tipo_Visita REFERENCES Tipo_Visita, numero_grado INT, CONSTRAINT cp PRIMARY KEY (numero_reserva, codigo_tipo_visita, numero_grado));
CREATE TABLE Reserva_Tipo_Visita (numero_reserva CHAR(10) CONSTRAINT numero_reserva2 REFERENCES Reserva, codigo_tipo_visita CHAR(10) CONSTRAINT codigo_tipo_visita2 REFERENCES Tipo_Visita, cantidad_alumnos_reservados INT, cantidad_alumno_reales INT, codigo_guia CHAR(10), CONSTRAINT cf_codigo_guia FOREIGN KEY (codigo_guia) REFERENCES Guia, CONSTRAINT cp PRIMARY KEY (numero_reserva, codigo_tipo_visita));	CREATE TABLE Distrito_Escolar (codigo_distrito CHAR(10) PRIMARY KEY, descripcion_distrito CHAR(50));

1. Añadir a la tabla Guía la columna sueldo_hora.

```
ALTER TABLE guia
ADD sueldo_hora NUMERIC;
```

2. Hacer que no puedan haber dos escuelas con el mismo nombre.

```
ALTER TABLE escuela
ADD CONSTRAINT u_nombre UNIQUE (nombre_escuela);
```

3. Eliminar la columna de domicilios de la tabla Escuela.

```
ALTER TABLE Escuela DROP COLUMN domicilio_escuela;
```

4. Agregar columnas calle_escuela y altura_escuela a la tabla Escuela

```
ALTER TABLE Escuela DROP COLUMN domicilio_escuela;

ALTER TABLE Escuela ADD calle_escuela CHAR(30), altura_escuela INT;
```

5. Agregar domicilio_guia en tabla Guia.

```
ALTER TABLE Guia ADD domicilio_guia CHAR(50);
```

6. Agregar tabla Email_Escuela (sin Clave primaria)

```
CREATE TABLE Email_Escuela (
      email_escuela CHAR(30) NOT NULL,
      codigo_escuela CHAR(10) NOT NULL CONSTRAINT cfEE
      FOREIGN KEY (codigo_escuela) REFERENCES escuela);
```

7. Establecer clave primaria para Email_Escuela

```
ALTER TABLE Email_Escuela
ADD CONSTRAINT cpEE PRIMARY KEY (codigo_escuela, telefono_escuela);
```

8. Crear un índice que impida ingresar dos guías con igual nombre y apellido.

```
CREATE unique INDEX idx_nom_guia
ON guia ( apellido_guia, nombre_guia) ;
```

9. Actualice el teléfono de una de las escuelas por el número 5445-3223.

```
UPDATE telefono_escuela SET telefono_escuela = '5445-3223'
WHERE codigo_escuela = 5 ;
```

10. Actualice la fecha de una reserva que usted seleccione por 23/12/2004.

```
UPDATE reserva SET fecha_visita_reservada = '20041223'
WHERE numero_reserva = 2 ;
```

11. Debe realizarse un descuento en el arancel por alumno de $2 para todas las reservas de más de 10 alumnos.

```
UPDATE reserva_tipo_visita
SET arancel_por_alumno = arancel_por_alumno - 2
WHERE cantidad_alumnos_reservados > 10 ;
```

12. Actualice el código de guía de las reservas que tengan asignado al guía 1 por el guía 2.

```
UPDATE reserva_tipo_visita SET codigo_guia = 2
WHERE codigo_guia = 1 ;
```

13. Borrar todas las reservas con menos de 10 Alumnos.

```
DELETE FROM reserva_tipo_visita
WHERE cantidad_alumnos_reservados < 10 ;
```

14. Elimine a todos los guías que no tengan cargado su nombre.

```
DELETE FROM guia WHERE nombre_guia = ' '
OR nombre_guia IS NULL ;
```

15. Obtenga un listado de todos los guías de nombre Bernardo.

```
SELECT *
FROM guia
WHERE nombre_guia LIKE '%Bernardo%' ;
```

16. Se desea obtener la cantidad de reservas con fecha mayor a 03/01/2004.

```
SELECT COUNT(* )
FROM reserva
WHERE fecha_visita_reservada > '20040103' ;
```

17. Se necesita conocer la cantidad total de alumnos reservados para cada reserva (agrupadas por reservas).

```
SELECT numero_reserva,
       SUM(cantidad_alumnos_reservados) AS total_reserva
FROM reserva_tipo_visita
GROUP BY numero_reserva ;
```

18. Liste todas las reservas que posee una cantidad total de alumnos reservados mayor a 100.

```
SELECT numero_reserva,
       SUM(cantidad_alumnos_reservados) AS total_reserva
FROM reserva_tipo_visita
GROUP BY numero_reserva
HAVING SUM(cantidad_alumnos_reservados) > 100 ;
```

19. Muestre las reservas realizadas en las cuales la inasistencia a las visitas sea mayor a 5.

```
SELECT numero_reserva,
     cantidad_alumnos_reservados , cantidad_alumnos_reales
FROM reserva_tipo_visita
WHERE (cantidad_alumnos_reservados - cantidad_alumnos_reales) > 5 ;
```

20. Obtenga la cantidad de escuelas que visitarán el parque después del '3/6/2004'.

```
SELECT COUNT (distinct codigo_escuela)
FROM reserva
WHERE fecha_visita_reservada > '20040603' ;
```

21. Insertar un nuevo guía con el número inmediato consecutivo al máximo existente.

```
INSERT INTO guia
SELECT MAX(codigo_guia) + 1, 'Gonzalez', 'Roberto'
FROM guia;
```

22. Insertar un nuevo tipo de visita con el número inmediato consecutivo al máximo existente sin utilizar subconsultas.

```
INSERT INTO tipo_visita
SELECT TOP 1 codigo_tipo_visita + 1, 'NUEVA VISITA'
FROM tipo_visita ORDER BY codigo_tipo_visita DESC ;
```

23. Insertar los datos de la tabla escuela en una nueva tabla, borre los datos de la tabla escuela. En la nueva tabla realice una actualización de los códigos de escuela incrementándose en uno. Posteriormente reinsértelos en la tabla escuela y vuelva a la normalidad los códigos.

```
SELECT * INTO escuela2 FROM ESCUELA ;
```

```
DELETE FROM ESCUELA ;
UPDATE escuela2 SET codigo_escuela = codigo_escuela + 1 ;

INSERT INTO escuela SELECT * FROM escuela2 ;

UPDATE ESCUELA SET codigo_escuela = codigo_escuela - 1 ;

DROP TABLE escuela2 ;
```

24. . Debido a un feriado inesperado, las fechas de las visitas deben posponerse por un día.

```
UPDATE reserva
SET fecha_visita_reservada = fecha_visita_reservada + 1 ;
```

25. Obtener los datos de la última reserva existente.

```
SELECT TOP 1 *
FROM reserva
  ORDER BY fecha_visita_reservada DESC, hora_visita_reservada ASC ;
```

26. Obtener los apellidos de los guías que se encuentren repetidos.

```
SELECT COUNT(apellido_guia) AS repetidos, apellido_guia
FROM guia
GROUP BY apellido_guia
HAVING COUNT(apellido_guia)> 1 ;
```

27. Obtener un listado con la cantidad de reservas por fecha.

```
SELECT fecha_visita_reservada, COUNT(*) AS cantidad_de_reservas
FROM reserva
GROUP BY fecha_visita_reservada ;
```

28. Obtener el promedio de alumnos asistentes, reservados y la diferencia entre estos promedios.

```
SELECT AVG(cantidad_alumnos_reales),
       AVG(cantidad_alumnos_reservados),
       AVG(cantidad_alumnos_reservados) - AVG(cantidad_alumnos_reales)
FROM reserva_tipo_visita ;
```

29. Obtener los guías que tengan más de 3 visitas.

```
SELECT codigo_guia, COUNT(codigo_tipo_visita)
FROM reserva_tipo_visita
GROUP BY codigo_guia HAVING COUNT(codigo_tipo_visita) > 3 ;
```

30. Listar las escuelas cuyos domicilios sean de calles que empiecen con S, indicando nombre, domicilio y teléfono. Hacer una versión en la que aparezcan sólo las que tienen teléfono, y hacer otra en la que aparezca solo las escuelas con domicilio en calles que empiecen con S y que no tienen ningún teléfono.

V1:

```
SELECT ESC.Nombre_Escuela, ESC.Domicilio_Escuela, TEL.Telefono_Escuela
FROM escuela ESC INNER JOIN telefono_escuela TEL
ON ESC.Codigo_Escuela = TEL.Codigo_Escuela
WHERE ESC.Domicilio_Escuela LIKE 'S%';
```

V2:

```
SELECT ESC.Nombre_Escuela, ESC.Domicilio_Escuela, TEL.Telefono_Escuela
FROM escuela ESC LEFT JOIN telefono_escuela TEL
ON ESC.Codigo_Escuela = TEL.Codigo_Escuela
WHERE ESC.Nombre_Escuela LIKE 'S%'
AND TEL.Codigo_Escuela IS NULL;
```

31. Listar las reservas mostrando día, nombre de escuela, cantidad de alumnos de reserva y el nombre del guía.

```
SELECT RES.Numero_Reserva,
     RES.Fecha_Visita_Reservada,
     ESC.Nombre_Escuela,
     REV.Cantidad_Alumnos_Reservados,
     GUI.Nombre_Guia, GUI.Apellido_Guia
FROM guia GUI,
     escuela ESC,
     reserva RES,
     Reserva_Tipo_Visita REV
WHERE ESC.Codigo_Escuela = RES.Codigo_Escuela
AND RES.Numero_Reserva = REV.Numero_Reserva
AND GUI.Codigo_Guia = REV.Codigo_Guia;
```

32. Listar las reservas, la cantidad total real de alumnos y el valor total (cantidad x arancel con iva incluido).

```
SELECT REV.Numero_Reserva,
SUM(REV.Cantidad_Alumnos_Reales) AS TOTAL_Alumn_Asistentes,
SUM(REV.Cantidad_Alumnos_Reales * TVI.Arancel_por_Alumno * 1.21)
AS VALOR_TOTAL
FROM tipo_visita TVI INNER JOIN Reserva_Tipo_Visita REV
ON TVI.Codigo_Tipo_Visita = REV.Codigo_Tipo_Visita
GROUP BY REV.Numero_Reserva;
```

33. Listar las reservas y los nombres de escuelas con valor total mayor a $1.000.

```
SELECT ESC.Nombre_Escuela, REV.Numero_Reserva,
       SUM(REV.Cantidad_Alumnos_Reales) AS TOTAL_CantAlumn_Asistentes,
       SUM(REV.Cantidad_Alumnos_Reales * TVI.Arancel_por_Alumno) AS
           VALOR_TOTAL
FROM escuela ESC,
     Reserva_Tipo_Visita AS REV,
     reserva, tipo_visita TVI
WHERE REV.Numero_Reserva = reserva.Numero_Reserva
AND ESC.Codigo_Escuela = reserva.Codigo_Escuela
AND TVI.Codigo_Tipo_Visita = REV.Codigo_Tipo_Visita
GROUP BY ESC.Nombre_Escuela, REV.Numero_Reserva
HAVING
SUM(REV.Cantidad_Alumnos_Reales * TVI.Arancel_por_Alumno) > 1000;
```

34. Listar las escuelas que fueron atendidas alguna vez por el guía “Eleonora Fernandez”.

```
SELECT ESC.Nombre_Escuela
FROM escuela ESC,
     reserva RES,
     guia GUI,
     Reserva_Tipo_Visita REV
WHERE ESC.Codigo_Escuela = RES.Codigo_Escuela
AND GUI.Codigo_Guia = REV.Codigo_Guia
AND RES.Numero_Reserva = REV.Numero_Reserva
AND GUI.Nombre_Guia = 'Eleonora'
AND GUI.Apellido_Guia = 'Fernandez'
GROUP BY ESC.Nombre_Escuela;
```

35. Listar las escuelas que realizaron más de una reserva para el mismo día.

```
SELECT ESC.Nombre_Escuela, RES.Numero_Reserva,
     RES.Fecha_Visita_Reservada
FROM escuela ESC INNER JOIN reserva RES
       ON ESC.Codigo_Escuela = RES.Codigo_Escuela
INNER JOIN Reserva_Tipo_Visita REV
ON RES.Numero_Reserva = REV.Numero_Reserva
```

```
GROUP BY ESC.Nombre_Escuela,
     RES.Numero_Reserva,
     RES.Fecha_Visita_Reservada
HAVING COUNT(RES.Fecha_Visita_Reservada) > 1;
```

36. Listar los nombres de los guías y la cantidad de visitas para aquellos con más de 3 visitas de 30 personas.

```
SELECT GUI.Nombre_Guia, GUI.Apellido_Guia,
       COUNT(REV.Numero_Reserva) AS CANT_TOTAL_RESERVAS ,
       SUM(REV.Cantidad_Alumnos_Reales) AS
            Cantidad_TOTAL_Alumnos_Asistentes
FROM escuela ESC INNER JOIN reserva RES
       ON ESC.Codigo_Escuela = RES.Codigo_Escuela
       INNER JOIN Reserva_Tipo_Visita REV
       ON RES.Numero_Reserva = REV.Numero_Reserva
       INNER JOIN guia GUI
       ON REV.Codigo_Guia = GUI.Codigo_Guia
WHERE REV.Cantidad_Alumnos_Reales > 30
GROUP BY GUI.Nombre_Guia, GUI.Apellido_Guia
HAVING COUNT(REV.Numero_Reserva) > 3 ;
```

37. Listar las reservas, el día, la cantidad total de alumnos por tipo de visita, el nombre de la escuela y el nombre del guía.

```
SELECT ESC.Nombre_Escuela, GUI.Apellido_Guia, GUI.Nombre_Guia,
      RES.Numero_Reserva, TV.Descripcion_Tipo_Visita,
     SUM(REV.Cantidad_Alumnos_Reales) AS TOTAL_Cantidad_Alumnos_Reales
FROM tipo_visita TV, escuela ESC,
     guia GUI,reserva RES,
     Reserva_Tipo_Visita REV
WHERE RES.Numero_Reserva = REV.Numero_Reserva
AND GUI.Codigo_Guia = REV.Codigo_Guia
AND ESC.Codigo_Escuela = RES.Codigo_Escuela)
AND TV.Codigo_Tipo_Visita = REV.Codigo_Tipo_Visita
GROUP BY ESC.Nombre_Escuela, GUI.Apellido_Guia,
     GUI.Nombre_Guia, RES.Numero_Reserva, TV.Descripcion_Tipo_Visita;
```

38. Listar los tipos de visita (Código y Descripción) que han sido asignadas más de 5 veces.

```
SELECT TV.Codigo_Tipo_Visita, TV.Descripcion_Tipo_Visita
FROM Reserva_Tipo_Visita REV, tipo_visita TV
WHERE REV.Codigo_Tipo_Visita = TV.Codigo_Tipo_Visita
GROUP BY TV.Codigo_Tipo_Visita, TV.Descripcion_Tipo_Visita
HAVING COUNT(REV.Codigo_Tipo_Visita) > 5;
```

39. . Listar los Guías (Apellido y Nombre) que no han sido nunca asignados a una reserva.

```
SELECT GUI.Apellido_Guia, GUI.Nombre_Guia
FROM guia GUI LEFT JOIN Reserva_Tipo_Visita REV
ON GUI.Codigo_Guia = REV.Codigo_Guia
WHERE REV.Codigo_Guia IS NULL
GROUP BY GUI.Apellido_Guia, GUI.Nombre_Guia;
```

40. Listar los Guías (Apellido y Nombre) con visitas de escuelas cuyo nombre no comience con “E”.

```
SELECT GUI.Apellido_Guia, GUI.Nombre_Guia
FROM escuela ESC,reserva RES, guia GUI, Reserva_Tipo_Visita REV
WHERE ESC.Codigo_Escuela = RES.Codigo_Escuela
AND GUI.Codigo_Guia = REV.Codigo_Guia
AND RES.Numero_Reserva = REV.Numero_Reserva
AND ESC.Nombre_Escuela NOT LIKE 'E%'
GROUP BY GUI.Apellido_Guia, GUI.Nombre_Guia;
```

41. Listar las escuelas (Nombre) que tengan visitas en donde la cantidad de alumnos reservada sea igual a la cantidad de alumnos reales asistentes.

```
SELECT ESC.Nombre_Escuela
FROM escuela ESC, reserva RES, Reserva_Tipo_Visita REV
WHERE ESC.Codigo_Escuela = RES.Codigo_Escuela
AND REV.Cantidad_Alumnos_Reservados = REV.Cantidad_Alumnos_Reales
AND RES.Numero_Reserva = REV.Numero_Reserva
GROUP BY ESC.Nombre_Escuela;
```

42. Realizar la Unión de las consultas: Nombres de las escuelas con reservas antes de las 9:00 hs Apellidos de los guías cuyos nombres comiencen con “v”. Ordenar el resultado en forma descendente

```
SELECT ESC.Nombre_Escuela
FROM escuela AS ESC ,reserva AS RES
WHERE ESC.Codigo_Escuela = RES.Codigo_Escuela
AND RES.Hora_Visita_Reservada < '9:00:00 AM'
UNION
SELECT GUI.Apellido_Guia
FROM guia GUI
WHERE GUI.Nombre_Guia LIKE 'V%'
ORDER BY 1 DESC;
```

43. Listar los Nombres y Teléfonos de las Escuelas que concurrieron a la visita: “Los Mamuts en Familia”.

```
SELECT TV.Descripcion_Tipo_Visita, ESC.Nombre_Escuela,
     TEL.Telefono_Escuela
FROM escuela ESC , telefono_escuela TEL,
     reserva RES, tipo_visita TV, Reserva_Tipo_Visita REV
WHERE ESC.Codigo_Escuela = TEL.Codigo_Escuela
AND ESC.Codigo_Escuela = RES.Codigo_Escuela
AND TV.Codigo_Tipo_Visita = REV.Codigo_Tipo_Visita
AND RES.Numero_Reserva = REV.Numero_Reserva
AND TV.Descripcion_Tipo_Visita = 'Los Mamuts en Familia'
GROUP BY TV.Descripcion_Tipo_Visita, ESC.Nombre_Escuela,
     TEL.Telefono_Escuela;
```

44. Listar para cada grado (sin importar de cual escuela) cuál fue la última vez que hicieron una visita.

```
SELECT RVG.Numero_Grado,
       MAX(RES.Fecha_Visita_Reservada) AS Ultima_Reserva
FROM reserva RES, Reserva_Por_Grado RVG
WHERE RES.Numero_Reserva = RVG.Numero_Reserva
GROUP BY RVG.Numero_Grado;
```

45. Insertar en una nueva tabla llamada Guia_Performance los datos del Guía y las sumas de las cantidades reservadas y reales del año 2003.

```
SELECT GUI.Codigo_Guia, GUI.Apellido_Guia, GUI.Nombre_Guia,
       SUM(REV.Cantidad_Alumnos_Reservados) AS
            Cantidad_Alumnos_Reservados,
       SUM(REV.Cantidad_Alumnos_Reales) AS Cantidad_Alumnos_Reales INTO
            Guia_Performance
FROM guia AS GUI, Reserva_Tipo_Visita AS REV, reserva
WHERE GUI.Codigo_Guia = REV.Codigo_Guia
AND REV.Numero_Reserva = reserva.Numero_Reserva
AND Fecha_Visita_Reservada BETWEEN '01-01-2003' AND '31-12-2003'
GROUP BY GUI.Codigo_Guia, GUI.Apellido_Guia, GUI.Nombre_Guia;
```

46. Listar los tipos de visita que fueron guiadas alguna vez por Cristina Zaluzi.

```
SELECT DISTINCT tv.descripcion_tipo_visita
FROM Reserva_Tipo_Visita rev, tipo_visita tv
WHERE rev.codigo_tipo_visita = tv.codigo_tipo_visita
AND rev.codigo_guia IN ( SELECT gui.Codigo_Guia
                         FROM guia gui
                         WHERE gui.Apellido_Guia = 'Zaluzi'
                         AND gui.Nombre_Guia = 'Cristina' );
```

47. Listar los nombres de escuela que visitaron en 2002 pero no lo hicieron en el 2001.

```
SELECT Nombre_Escuela
FROM escuela esc
WHERE Codigo_Escuela NOT IN (SELECT res1.codigo_escuela
                             FROM reserva res1
                             WHERE res1.Fecha_Visita_Reservada)
                             NOT BETWEEN '01-01-2001' AND '31-12-2001')
AND esc.Codigo_Escuela IN (SELECT res2.codigo_escuela
                             FROM reserva res2
                             WHERE YEAR(res2.Fecha_Visita_Reservada)
                             BETWEEN '01-01-2002' AND '31-12-2002');
```

48. Listar los guías (codigo, nombre y apellido) que fueron asignados a más de 2 tipos de visita distintos y con una cantidad total real de alumnos que guiaron, mayor a 200.

```
SELECT gui.codigo_guia, gui.nombre_guia, gui.apellido_guia
FROM guia gui
WHERE gui.codigo_guia IN ( SELECT rev.codigo_guia
                           FROM Reserva_Tipo_Visita rev
                           GROUP BY rev.codigo_guia
                           HAVING
                           COUNT (DISTINCT rev.codigo_tipo_visita)) > 2
                           AND SUM(rev.Cantidad_Alumnos_Reales) > 200);
```

49. Listar las escuelas que poseen más de 1 reservas con más de 2 tipos de visitas para cada reserva.

```
SELECT esc.codigo_escuela, nombre_escuela
FROM escuela esc WHERE esc.codigo_escuela
IN ( SELECT res.codigo_escuela
     FROM reserva res
     WHERE res.Numero_Reserva IN (SELECT rev.Numero_Reserva
                                  FROM Reserva_Tipo_Visita rev
                                  GROUP BY rev.Numero_Reserva
                                  HAVING
                                  COUNT (rev.codigo_tipo_visita)> 2 )
     GROUP BY res.codigo_escuela
     HAVING COUNT(res.Numero_Reserva) > 1 );
```

50. Listar el nombre, apellido y código de aquellos guías que, en alguna visita de una reserva en particular, hayan atendido por lo menos al 40% de los alumnos totales guiados en todas sus visitas.

```
SELECT gui.codigo_guia, gui.apellido_guia, gui.nombre_guia
FROM guia gui
WHERE gui.codigo_guia IN (SELECT rev1.codigo_guia
                          FROM Reserva_Tipo_Visita rev1
                          WHERE rev1.Cantidad_Alumnos_Reales >
                                (SELECT
                                SUM(Cantidad_Alumnos_Reales)*0.40
                                FROM Reserva_Tipo_Visita rev2
                                WHERE rev2.codigo_guia =
                                rev1.codigo_guia ));
```

51. Listar el nombre y el código de aquellas escuelas que hayan asistido el día en que se registró la mayor cantidad de alumnos reales.

```
SELECT esc.codigo_escuela, esc.nombre_escuela
FROM escuela esc
WHERE esc.codigo_escuela IN ( SELECT res.codigo_escuela
                               FROM reserva res
                              WHERE res.fecha_visita_reservada =
                              (SELECT TOP 1 res2.fecha_visita_reservada
                              FROM reserva res2
                              INNER JOIN Reserva_Tipo_Visita rev
                              ON  res2.Numero_Reserva  =  rev.Numero_Reserva
                              GROUP BY res2.fecha_visita_reservada
                              ORDER BY SUM(rev.Cantidad_Alumnos_Reales) DESC
                              ) ) ;
```

52. Listar el código y nombre de las escuelas cuya fecha de reserva sea igual a la primera fecha de reserva realizada.

```
SELECT esc.codigo_escuela, esc.nombre_escuela
FROM escuela esc
INNER JOIN reserva res
ON esc.codigo_escuela = res.codigo_escuela
WHERE res.Fecha_Visita_Reservada = ( SELECT MIN(Fecha_Visita_Reservada)
                                     FROM reserva );
```

53. Listar las escuelas que visitaron entre los años 2001 y en el 2002.

```
SELECT esc.Nombre_Escuela
FROM escuela esc
WHERE esc.Codigo_Escuela IN ( SELECT res1.codigo_escuela
                              FROM reserva res1
                              WHERE YEAR(res1.Fecha_Visita_Reservada)
                              BETWEEN 2001 AND 2002 );
```

54. Listar los guías que tuvieron más de 3 escuelas diferentes y una cantidad total real de alumnos mayor a 200.

```
SELECT gui.codigo_guia, gui.nombre_guia, gui.apellido_guia
FROM guia gui
WHERE gui.codigo_guia IN ( SELECT rev.codigo_guia
                           FROM Reserva_Tipo_Visita rev
                           INNER JOIN reserva res
                           ON rev.numero_reserva=res.numero_reserva
                           GROUP BY rev.codigo_guia
                           HAVING
                           COUNT (DISTINCT (res.codigo_escuela)) > 3
                           AND SUM(rev.cantidad_alumnos_reales) > 100 );
```

55. Listar los nombres y códigos de escuelas con gasto total de todas las visitas mayor a $1700.

```
SELECT esc.Nombre_Escuela
FROM escuela esc
WHERE esc.Codigo_Escuela IN ( SELECT res.codigo_escuela
                              FROM reserva res
                              INNER JOIN reserva_tipo_visita rev
                              ON res.numero_reserva = rev.numero_reserva
                              INNER JOIN tipo_visita tv
                              ON tv.codigo_tipo_visita =
                              rev.codigo_tipo_visita
                              GROUP BY res.codigo_escuela
                              HAVING
                              (SUM((tv.Arancel_por_alumno)*
                         rev.cantidad_alumnos_reales) > 1700) );
```

56. Listar los guías que hayan tenido en solo un tipo de visita de una reserva en particular por lo menos el 45% del total de alumnos totales que esa persona atendió.

```
SELECT gui.codigo_guia, gui.apellido_guia, gui.nombre_guia
FROM guia gui
WHERE gui.codigo_guia IN ( SELECT rev1.codigo_guia
                           FROM Reserva_Tipo_Visita rev1
                           WHERE rev1.cantidad_alumnos_reales >
                                 (SELECT SUM(cantidad_alumnos_reales) * 0.45
                                 FROM Reserva_Tipo_Visita rev2
                                 WHERE rev2.codigo_guia = rev1.codigo_guia ) );
```

Ejercicio 25 - Vuelta Ciclista [21]

Dado el siguiente modelo relacional, se pide:

EQUIPO(codigoequipo, nombreequipo, director)
CICLISTA(codigociclista, nombreciclista, fechanacimiento, peso, ***Codigoequipo***)
ETAPA(numetapa, km, salida, llegada, ***codigociclista***)
TRAMO(codigotramo, nombretramo, altura, categoria, ***numetapa***,***codigociclista***)
CAMISETA(codigocamiseta, tipo,premio, color)
LLEVA(***numetapa, codigocamiseta,codigociclista***)

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita.

Equipo: contiene los datos de los distintos equipos: nombre (nomequipo) y nombre de su director (director).
Ciclista: contiene los datos de los ciclistas que componen los distintos equipos: número del dorsal (dorsal), nombre del ciclista (nombre), edad del ciclista (edad) y nombre del equipo al que pertenece (nomequipo).
Etapa: contiene los datos de las etapas que componen la vuelta ciclista: número de la etapa(numetapa) (las etapas se numeran consecutivamente: 1, 2, ...), kilómetros que tiene la etapa (kms), nombre de la población de donde sale la etapa (salida), nombre de la población donde está la meta de la etapa (llegada) y número del dorsal del ciclista que ha ganado la etapa (dorsal). Los atributos salida y llegada están definidos sobre el mismo dominio.
Tramo: contiene los datos de los tramos de montaña que visita la vuelta ciclista: nombre del tramo(nomtramo), altura máxima del tramo(altura), categoría del tramo: primera, especial, etc. (categoría), porcentaje que indica la pendiente media del tramo(pendiente), número de la etapa donde se sube el tramo (numetapa) y número del dorsal que ha ganado el tramo al pasar en primera posición (dorsal).
Camiseta: contiene los datos de los premios que se otorgan mediante las distintas camisetas: código de la camiseta (código), clasificación que premia esa camiseta : general, montaña, etc. (tipo), color de la camiseta asociada (color) e importe del premio que corresponde al ciclista que termine la vuelta llevando la camiseta (premio).
Lleva: contiene la información sobre qué ciclistas (dorsal) han llevado cada camiseta (código) en cada una de las etapas (numetapa)

1. Nombre y el equipo de aquellos corredores menores de 30 años que han ganado alguna etapa.

   Solución 1:

```
SELECT DISTINCT nombre, nomequipo
FROM Ciclista C, Etapa E
WHERE edad < 30
AND C.dorsal = E.dorsal;
```

   Solución 2:

```
SELECT DISTINCT nombre, nomequipo
FROM Ciclista
WHERE edad < 30
AND dorsal IN ( SELECT DISTINCT dorsal FROM Etapa);
```

2. Nombre y equipo de los corredores mayores de 35 años que han ganado algún tramo.

   Solución 1:

```
SELECT DISTINCT nombre, nomequipo
FROM Ciclista C, Tramo P
WHERE C.dorsal = P.dorsal
AND edad > 35;
```

   Solución 2:

```
SELECT DISTINCT nombre, nomequipo
FROM Ciclista
WHERE edad > 35
AND dorsal IN (SELECT DISTINCT dorsal FROM Tramo);
```

3. Datos de las etapas que pasan por algún tramo de montaña y que tienen salida y llegada en la misma población.

   Solución 1:

```
SELECT DISTINCT E.numetapa, E.kms, E.salida, E.llegada, E.dorsal
FROM Etapa E, Tramo P
WHERE E.numetapa = P.numetapa
AND E.salida = E.llegada;
```

   Solución 2:

```
SELECT *
FROM Etapa
WHERE salida = llegada
AND numetapa IN ( SELECT numetapa FROM Tramo);
```

4. Poblaciones que tienen la meta de alguna etapa, pero desde las que no se realiza ninguna salida

   Solución 1:

```
SELECT llegada FROM Etapa
MINUS
SELECT salida FROM Etapa;
```

   Solución 2:

```
SELECT DISTINCT llegada
FROM Etapa
WHERE llegada NOT IN (SELECT salida FROM Etapa);
```

5. Nombre y el equipo de los ciclistas que han ganado alguna etapa llevando la camiseta de color amarillo, mostrando también el número de etapa.

```
SELECT C.nombre, C.nomequipo, E.numetapa
FROM Ciclista C, Etapa E, Lleva LL, Camiseta M
WHERE E.dorsal = LL.dorsal
AND E.numetapa = LL.numetapa
AND LL.código = M.código
AND M.color =  'amarillo'
AND E.dorsal = C.dorsal;
```

6. Poblaciones de salida y de llegada de las etapas donde se encuentran tramos con altura superior a 1300 metros.

```
SELECT DISTINCT salida, llegada
FROM Etapa E, Tramo P
WHERE E.numetapa = P.numetapa
AND altura > 1300;
```

7. Número de las etapas que tienen algún tramos de montaña, indicando cuántos tiene cada una de ellas.

```
SELECT numetapa, COUNT(*) AS numero_tramos
FROM Tramo
GROUP BY numetapa;
```

8. Nombre y edad de los ciclistas que han llevado dos o más camisetas en una misma etapa

```
SELECT DISTINCT LL.dorsal, C.nombre, C.edad
FROM Ciclista C, Lleva LL
WHERE C.dorsal = LL.dorsal
GROUP BY LL.numetapa, LL.dorsal, C.nombre, C.edad
HAVING COUNT(*) >= 2;
```

9. Obtener el nombre y el equipo de los ciclistas que han llevado alguna camiseta o que han ganado algún tramo.

Solución 1:

```
SELECT C.nombre, C.nomequipo
FROM Ciclista C, Lleva LL
WHERE C.dorsal = LL.dorsal
UNION
SELECT C.nombre,C.nomequipo
FROM Ciclista C, Tramo P
WHERE C.dorsal = P.dorsal;
```

Solución 2:

```
SELECT nombre, nomequipo
FROM Ciclista
WHERE dorsal IN (SELECT dorsal FROM Lleva)
OR dorsal IN (SELECT dorsal FROM Tramo);
```

10. Obtener los números de las etapas que no tienen tramos de montaña.

Solución 1:

```
SELECT numetapa FROM Etapa
MINUS
SELECT numetapa FROM Tramo;
```

Solución 2:

```
SELECT numetapa
FROM Etapa
WHERE numetapa NOT IN (SELECT numetapa FROM Tramo);
```

11. Obtener la edad media de los ciclistas que han ganado alguna etapa.

```
SELECT AVG(C.edad) AS media_edad_ganadores
FROM Ciclista C, Etapa E
WHERE C.dorsal = E.dorsal ;
```

12. Obtener el nombre de los tramos de montaña que tienen una altura superior a la altura media de todos los tramos.

```
SELECT nompuerto
FROM Tramo
WHERE altura > ( SELECT AVG(altura) FROM Tramo) ;
```

13. Obtener las poblaciones de salida y de llegada de las etapas donde se encuentran los puertos con mayor pendiente.

Solución 1:

```
SELECT DISTINCT salida, llegada
FROM Etapa E, Tramo P
WHERE E.numetapa = P.numetapa
AND P.pendiente = (SELECT MAX(pendiente) FROM Tramo) ;
```

Solución 2:

```
SELECT DISTINCT E.salida, E.llegada
FROM Etapa E, Tramo P
WHERE E.numetapa = P.numetapa
AND P.pendiente >= ALL(SELECT pendiente FROM Tramo) ;
```

14. Obtener el dorsal y el nombre de los ciclistas que han ganado los puertos de mayor altura.

Solución 1:

```
SELECT DISTINCT C.dorsal, C.nombre
FROM Ciclista C, tramo P
WHERE C.dorsal = P.dorsal
AND P.altura = (SELECT MAX(altura)FROM tramo) ;
```

Solución 2:

```
SELECT DISTINCT C.dorsal,C.nombre
FROM tramo P, Ciclista C
WHERE P.dorsal = C.dorsal
AND P.altura >= ALL(SELECT altura FROM tramo) ;
```

15. Obtener los datos de las etapas cuyos puertos (todos) superan los 1300 metros de altura.

Solución 1:

```
SELECT E.*
FROM Tramo P, Etapa E
WHERE P.numetapa = E.numetapa
AND P.altura > 1300
MINUS
SELECT E.*
FROM Tramo P, Etapa E
WHERE P.numetapa = E.numetapa
AND P.altura <= 1300 ;
```

Solución 2:

```
SELECT DISTINCT E.*
FROM Tramo P, Etapa E
WHERE P.numetapa = E.numetapa
AND P.altura > 1300
AND E.numetapa NOT IN (SELECT numetapa
                       FROM Tramo
                       WHERE altura <= 1300) ;
```

16. Obtener la edad del ciclista más joven, la del más veterano y la media de edad de los ciclistas que han participado en la vuelta.

```
SELECT MIN(edad) AS menor_edad,
       MAX(edad) AS mayor_edad,
       AVG(edad) AS media_edad
FROM Ciclista;
```

17. Obtener el nombre del equipo y el director del ciclista que ha ganado la etapa más larga.

Solución 1:

```
SELECT director, Q.nomequipo
FROM Etapa E, Ciclista C, Equipo Q
WHERE E.dorsal = C.dorsal
AND C.nomequipo = Q.nomequipo
AND kms = (SELECT MAX(kms) FROM Etapa);
```

Solución 2:

```
SELECT TOP(1) director, Q.nomequipo
FROM Etapa E, Ciclista C, Equipo Q
WHERE E.dorsal = C.dorsal
AND C.nomequipo = Q.nomequipo
ORDER BY kms DESC;
```

18. Obtener el dorsal y el nombre de los ciclistas que hayan ganado alguna etapa, mostrando también el número de etapas que han ganado.

```
SELECT E.dorsal, C.nombre,
       COUNT(E.dorsal) AS etapas_ganadas
FROM Etapa E, Ciclista C
WHERE C.dorsal = E.dorsal
GROUP BY  E.dorsal, C.nombre;
```

19. Obtener el nombre de los ciclistas que pertenecen a un equipo de más de cinco ciclistas y que han ganado alguna etapa, indicando también cuántas etapas han ganado.

```
SELECT C.dorsal , C.nombre,
       COUNT(*) AS num_etapas_ganadas
FROM Ciclista C, Etapa E
WHERE C.dorsal = E.dorsal
AND C.nomequipo IN (SELECT nomequipo FROM Ciclista
                    GROUP BY nomequipo
                    HAVING COUNT(*) > 5)
GROUP BY C.dorsal, C.nombre ;
```

20. Nombre y equipo de los ciclistas que han llevado alguna vez camiseta de color amarillo, indicando durante cuántas etapas lo han llevado.

```
SELECT DISTINCT C.nombre, C.nomequipo, COUNT(*) AS Num_veces
FROM Ciclista C, Lleva L, Camiseta M
WHERE C.dorsal = L.dorsal
AND L.código = M.código
AND color = 'amarillo'
GROUP BY C.nombre, C.nomequipo;
```

21. Por cada equipo, color de las camisetas que han llevado sus ciclistas.

```
SELECT nomequipo, color
FROM camiseta M, Lleva L, Ciclista C
WHERE M.código = L.código
AND L.dorsal = C.dorsal
GROUP BY nomequipo,color;
```

22. Nombre y equipo del ganador de la vuelta (es decir, el que ha lucido la camiseta amarilla en la última etapa).

```
SELECT C.nombre, C.nomequipo
FROM Ciclista AS C, Lleva AS L, CAmiseta AS M
WHERE C.dorsal = L.dorsal
AND M.código = L.código
AND color = 'amarillo'
AND L.numetapa = (SELECT MAX(numetapa)FROM Etapa);
```

23. Nombre de los equipos que no han llevado la camiseta roja.

```
SELECT DISTINCT nomequipo
FROM Equipo
WHERE nomequipo NOT IN (SELECT DISTINCT nomequipo
                        FROM Ciclista C, Lleva L, Camiseta M
                        WHERE C.dorsal = L.dorsal
                         AND L.código = M.código
                         AND color = 'rojo');
```

24. Nombre y dorsal de los ciclistas mayores de 30 años que han ganado algún tramo, junto con el número de tramos que han ganado.

```
SELECT C.dorsal, nombre, edad, COUNT(*) AS numeroDeTramos
FROM Ciclista AS C, Tramo AS P
WHERE C.dorsal = P.dorsal
AND edad > 30
GROUP BY C.dorsal, nombre, edad;
```

25. Nombre y director de los equipos que, en alguna etapa, sus ciclistas han llevado tres o más camisetas.

```
SELECT DISTINCT E.nomequipo, E.director
FROM Ciclista AS C, Equipo AS E, Lleva AS L
WHERE C.dorsal = L.dorsal
AND C.nomequipo = E.nomequipo
GROUP BY L.numetapa, E.nomequipo, E.director
HAVING COUNT(*) >= 3;
```

26. Nombre de los equipos que sólo tienen ciclistas menores de 28 años.

```
SELECT nomequipo
FROM Equipo
WHERE nomequipo IN (SELECT nomequipo
                    FROM Ciclista
                    WHERE edad < 28)
AND nomequipo NOT IN (SELECT nomequipo
                      FROM Ciclista
                      WHERE edad >= 28);
```

27. Dorsal y nombre del ciclista que ha llevado la camiseta verde durante más etapas.

Solución 1:

```
SELECT C.dorsal, C.nombre
FROM Ciclista AS C, Lleva AS L, Camiseta AS M
WHERE C.dorsal = L.dorsal
AND M.código = L.código
AND M.color = 'verde'
GROUP BY C.dorsal, C.nombre
HAVING COUNT(*) >=  ALL ( SELECT COUNT(*) AS veces
                          FROM Lleva LL, Camiseta MA
                          WHERE LL.código = MA.código
                          AND MA.color = 'verde'
                          GROUP BY LL.dorsal );
```

Solución 2:

```
SELECT TOP(1) dorsal, nombre
FROM Ciclista AS C, Lleva AS L, Maillot AS M
WHERE C.dorsal = L.dorsal
AND M.código = L.código
AND M.color = 'amarillo'
GROUP BY C.dorsal, C.nombre
ORDER BY COUNT(*) DESC;
```

28. Obtener todos los datos de aquellos ciclistas de los que se desconozca su edad.

```
SELECT *
FROM Ciclista
WHERE FechaNacimiento IS NULL;
```

29. Obtener de las camisetas, el tipo y el premio en dólares (está en bolívares) de aquellas cuyo premio supere los $5000.

```
SELECT tipo, premio/80000 premioEnDolares
FROM Camiseta
WHERE premio / 80000 > 5000;
```

30. Obtener el número de las etapas donde el nombre de la ciudad de llegada tenga por segunda letra una B o donde el nombre de la ciudad de salida lleve dos o más E.

```
SELECT NumEtapa
FROM Etapa
WHERE llegada LIKE '_B%'
OR salida LIKE '%E%E%';
```

31. Obtener el nombre y la fecha de nacimiento de los ciclistas que pertenezcan a equipos cuyo nombre contenga la cadena 100%.

```
SELECT nombreciclista, fechaNacimiento
FROM Ciclista, Equipo
WHERE nombreequipo LIKE '%100\%%'
AND equipo.codigoequipo = ciclista.codigoequipo;
```

Ejercicio 26 - Proyectos de Construcción

Dado el siguiente modelo relacional, se pide:

PROYECTOS(codP, descripcion, localidad, ***codCl***)
CLIENTE(CodCl, nombre, telefono, cedula)
CONDUCTOR(codC, nombre, localidad, categoria)
MAQUINA(CodM, nombreExcavadora, preciohora)
TRABAJO(CodT,***CodC, CodM, CodP***, fecha, tiempo)

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita.

1. Obtener el nombre de los conductores con categoría 15

```
SELECT nombre
FROM conductor
WHERE categoria = 15;
```

2. Obtener la descripción de sus proyectos que se hayan realizado durante los días 11 al 15 de septiembre de 2019.

Solución 1:

```
SELECT descripcion
FROM proyectos
WHERE CodP IN (SELECT CodP
               FROM trabajos
               WHERE fecha BETWEEN '11-09-2019' AND '15-09-2019');
```

Solución 2:

```
SELECT p.descripcion
FROM proyectos p, trabajos t
WHERE p.CodP = t.CodP
AND t.fecha BETWEEN '11-09-2019' AND '15-09-2019';
```

3. Obtener el nombre de los conductores que hayan trabajado con una Hormigonera, ordenados descendentemente.

Solución 1:

```
SELECT DISTINCT c.nombre
FROM conductor c, trabajo t, maquina m
WHERE c.codC = t.codC
AND t.codM = m.codM
AND m.nombre = 'Hormigonera'
ORDER BY nombre DESC;
```

Solución 2:

```
SELECT nombre
FROM conductor
WHERE CodC IN (SELECT codC
               FROM trabajos
               WHERE CodM  IN (SELECT CodM
                               FROM maquinas
                               WHERE nombre = 'HORMIGONERA'))
ORDER BY nombre DESC;
```

4. Obtener el nombre de los conductores que hayan trabajado con una Hormigonera en proyectos de la localidad de Caracas.

Solución 1:

```
SELECT c.nombre
FROM conductor c, trabajos t,maquina m, proyectos p
WHERE c.codC = t.codC
AND t.codM = m.codM
AND t.codP = p.codP
AND m.nombre = 'Hormigonera'
AND p.localidad = 'Caracas';
```

Solución 2:

```
SELECT nombre
FROM conductor
WHERE CodC IN (SELECT codC
               FROM trabajos
               WHERE CodM  IN (SELECT CodM
                               FROM maquinas
                               WHERE nombre = 'HORMIGONERA')
              AND CodP IN (SELECT CodP
                           FROM proyectos
                           WHERE localidad = 'Caracas'));
```

5. Obtener el nombre de los conductores y descripción del proyecto, para aquellos conductores que hayan trabajado con una Hormigonera en proyectos de Caracas durante los días 12 al 17 de Octubre del 2019.

```
SELECT c.nombre, p.descrip
FROM conductor c, trabajos t, maquina m, proyectos p
WHERE c.codC = t.codC
AND t.codM = m.codM
AND t.codP = p.codP
AND m.nombre = 'Hormigonera'
AND p.localidad = 'Caracas'
AND t.fecha BETWEEN '12-10-2019' AND '17-10-2019';
```

6. Obtener los conductores que trabajan en los proyectos de José Pérez.

```
SELECT DISTINCT c.nombre
FROM conductor c, trabajos t, proyectos p, cliente cl
WHERE c.codC = t.codC
AND p.codCI = cl.CodCI
AND cl.nombre = 'José Pérez';
```

7. Obtener el nombre y localidad de los conductores que NO trabajan en los proyectos de José Pérez.

```
SELECT DISTINCT c.nombre, c.localidad
FROM conductor c
WHERE c.codC NOT IN (SELECT codC
                     FROM trabajos
                     WHERE codP IN (SELECT codP
                                    FROM proyectos, cliente
                                    WHERE cliente.nombre = 'José Pérez'
                                    AND proyectos.CodCI = cliente.CodCI));
```

8. Obtener todos los datos de los proyectos realizados en Vargas o que sean de un cliente llamado José.

```
SELECT p.codP, p.descrip, p.localidad
FROM proyectos p, cliente c
WHERE p.CodCI = c.CodCI
AND (p.localidad = 'Vargas' OR c.nombre LIKE '%José%';
```

9. Obtener los conductores que figuren con más de 200 horas trabajadas.

```
SELECT c.nombre, SUM(t.tiempo) AS horasTrabajadas
FROM conductor c, trabajos t
WHERE c.codC = t.codC
GROUP BY c.nombre
HAVING SUM(t.tiempo) > 200;
```

10. Obtener los empleados que tengan como apellido Pérez y hayan trabajado en proyectos de localidades diferentes a las suyas.

Solución 1:

```
SELECT c.nombre
FROM conductor c, trabajos t, proyectos p
WHERE c.codC = t.codC
AND p.codP = t.codP
AND c.nombre LIKE '%Pérez'
AND p.localidad NOT IN (SELECT localidad
                        FROM conductor
                        WHERE nombre LIKE '%Pérez');
```

Solución 2:

```
SELECT c.nombre
FROM conductor c, trabajos t, proyectos p
WHERE c.nombre LIKE '%Pérez%'
AND p.localidad != c.localidad
AND p.codP = t.codP
AND t.codC = c.codC;
```

11. Obtener el nombre de los conductores y la localidad del proyecto, para aquellos conductores que hayan trabajado con máquinas con precio hora comprendido entre 10000 y 15000 Bs.

```
SELECT c.nombre, p.localidad
FROM conductor c, proyectos p, trabajos t, maquina m
WHERE c.codC = t.codC
AND p.codP = t.codP
AND m.codM = t.codM
AND m.precioHora BETWEEN '10000' AND '15000';
```

12. Obtener el nombre y localidad de los conductores, y la localidad del proyecto para aquellos proyectos que sean de Vargas y en los que no se haya utilizado una máquina de tipo Excavadora o una máquina de tipo Hormigonera.

```
SELECT c.nombre, c.localidad, p.localidad AS localidadProyecto
FROM conductor c, proyectos p, trabajos t, maquina m
WHERE c.codC = t.codC
AND p.codP = t.codP
AND m.codM = t.codM
AND p.localidad = 'Vargas'
AND m.nombre NOT IN ('Excavadora', 'Hormigonera');
```

13. Obtener el nombre de los conductores, el nombre del cliente y localidad del proyecto, en los que se haya utilizado la máquina con el menor precio hora.

```
SELECT c.nombre, cc.nombre, p.localidad AS localidadProyecto
FROM conductor c, proyectos p, trabajos t, maquina m, cliente cc
WHERE c.codC = t.codC
AND p.codP = t.codP
AND m.codM = t.codM
AND p.codCI = cc.codCI
AND m.precioHora = (SELECT MIN(precioHora)
                    FROM maquina);
```

14. Obtener todos los datos de los proyectos que siempre han utilizado una máquina de precio por sobre el promedio de éstas.

```
SELECT p.codP, p.descrip, p.localidad, p.cliente, p.telefono
FROM proyectos p, trabajos t, maquina m
WHERE p.codP = t.codP
AND m.codM = t.codM
AND m.precioHora > (SELECT AVG(precioHora)
                    FROM maquina);
```

15. Obtener los proyectos en los que haya trabajado el conductor de categoría más alta menos dos puntos, con la máquina de precio hora más bajo.

```
SELECT p.descrip AS Descripcion, P.Localidad, P.Cliente, P.Telefono
FROM conductor c, proyectos p, trabajos t, maquina m
WHERE c.codC = t.codC
AND p.codP = t.codP
AND m.codM = t.codM
AND c.categ = (SELECT MAX(categ)- 2
               FROM Conductor)
AND m.precioHora = (SELECT MIN(precioHora)
                    FROM Maquina);
```

16. Obtener por cada uno de los clientes el tiempo total empleado en sus proyectos.

```
SELECT p.codCI, SUM(t.tiempo) AS tiempoTotal
FROM proyectos p, trabajos t
WHERE p.codP = t.codP
GROUP BY p.codCI;
```

Ejercicio 27 - Empresas de envíos [21]

Dado el siguiente modelo relacional, se pide:

S (sn, snombre, estado, ciudad)
P (pn, pnombre, color, peso, ciudad)
SP (***sn***, ***pn***, cant)

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita.

La relación S almacena los datos de los proveedores: código (sn), nombre (snombre), estado (estado) y ciudad donde se ubica cada proveedor (ciudad).

La relación P almacena la información referente a las piezas: código (pn), nombre (pnombre), color (color), peso (peso) y ciudad donde se almacena cada pieza (ciudad).

Los atributos S.ciudad y P.ciudad están definidos sobre el mismo dominio (nombres de ciudades).

La relación SP almacena los datos sobre los envíos mensuales que cada proveedor realiza (sn)de cada pieza que suministra (pn). Cada uno de estos envíos contiene un número de unidades determinado (cant)

1. Piezas rojas con peso mayor que 15.

```sql
SELECT *
FROM P
WHERE color = 'rojo'
AND peso > 15;
```

2. Ciudades en las que hay proveedores y se almacena alguna pieza.

```sql
SELECT DISTINCT ciudad
FROM S
WHERE ciudad IN (SELECT ciudad FROM P);
```

3. Ciudades en las que hay proveedores y no se almacenan piezas.

```sql
SELECT DISTINCT ciudad
FROM S
WHERE ciudad NOT IN (SELECT ciudad FROM P);
```

4. Datos de los proveedores que envían la pieza P1 y cantidad que envían.

```
SELECT S.*, cant
FROM S,SP
WHERE S.sn = SP.sn
AND pn = 'P1';
```

5. Nombre de los proveedores que suministran la pieza con código P2, ordenados alfabéticamente.

```
SELECT DISTINCT snombre
FROM SP , S
WHERE SP.sn = S.sn
AND pn = 'P2'
ORDER BY snombre;
```

6. Nombre de los proveedores que suministran por lo menos una pieza roja.

```
SELECT DISTINCT snombre
FROM SP , S , P
WHERE SP.sn = S.sn
AND SP.pn = P.pn
AND P.color = 'rojo';
```

7. Nombre de todas las piezas que empiecen por “P”.

```
SELECT DISTINCT pnombre
FROM P
WHERE pnombre LIKE 'P%';
```

8. Nombre y cantidad de todas las piezas enviadas por el proveedor con código S3.

```
SELECT pnombre, cant
FROM SP, P
WHERE SP.pn = P.pn
AND SP.sn = 'S3';
```

9. Nombre de las piezas de color rojo suministradas por los proveedores de la ciudad de Caracas.

```
SELECT DISTINCT pnombre
FROM SP , S , P
WHERE SP.sn = S.sn
AND SP.pn = P.pn
AND P.color = 'rojo'
AND S.ciudad = 'Caracas';
```

10. Número total de unidades de piezas suministradas por cada proveedor.

```
SELECT sn AS Proveedor, SUM(cant) AS Total
FROM SP
GROUP BY sn;
```

11. Número medio de unidades suministrados.

```
SELECT AVG(cant) AS Media
FROM SP ;
```

12. Listado de las piezas cuyo peso supera las 12 unidades y cuyo nombre termina por la letra o ó por la letra a. De cada pieza hay que mostrar su número, su nombre y su color. El listado debe aparecer ordenado por el color. Las que sean del mismo color deben salir ordenadas por peso descendentemente.

```
SELECT pn, pnombre, color
FROM P
WHERE peso > 12
AND (pnombre LIKE '%a' OR pnombre LIKE '%o')
ORDER BY color ASC, peso DESC;
```

13. ¿Cuántos tipos de piezas suministra el proveedor S2?

```
SELECT COUNT(pn) AS numpiezas
FROM SP
WHERE sn = 'S2';
```

14. ¿Cuántos proveedores suministran la pieza P2?

```
SELECT COUNT(sn) AS numproveedores
FROM SP
WHERE pn = 'P2';
```

15. ¿En cuántas ciudades se almacenan piezas?

```
SELECT COUNT(DISTINCT(ciudad)) AS 'Num. ciudades'
FROM P;
```

16. ¿De cuántos colores tenemos piezas?

```
SELECT COUNT(DISTINCT(color)) AS 'Num. colores'
FROM P;
```

17. ¿Cuántas piezas son rojas?

```
SELECT COUNT(pn) AS piezas_rojas
FROM P
WHERE color = 'rojo';
```

18. ¿Cuántos proveedores hay en la base de datos?

```
SELECT COUNT(*) AS numproveedores
FROM S;
```

19. ¿Cuántos proveedores realizan suministros de piezas?

```
SELECT COUNT(DISTINCT(sn)) AS 'Num. proveedores'
FROM SP;
```

20. Muestra el peso de la pieza más pesada y el de la más ligera.

```
SELECT MAX(peso) AS peso_max, MIN(peso) AS peso_min
FROM P;
```

21. ¿Cuál es la cantidad media de los envíos del proveedor S1, sin tener en cuenta los de las piezas P1 y P2?

```
SELECT AVG(cant) AS media_envios
FROM SP WHERE sn = 'S1'
AND pn NOT IN ('P1', 'P2');
```

22. ¿Cuál es el nombre de la pieza más pesada?

Solución 1:

```
SELECT pnombre AS mas_pesada
FROM P
WHERE peso = (SELECT MAX(peso) FROM P);
```

Solución 2:

```
SELECT pnombre AS mas_pesada
FROM P WHERE peso >= ALL (SELECT peso FROM P);
```

23. Muestra los datos de los envíos de menor cantidad.

```
SELECT *
FROM SP
WHERE cant = (SELECT MIN(cant)FROM SP);
```

24. ¿Cómo se llaman los proveedores que tienen un estado superior al valor medio de éste?

```
SELECT snombre
FROM S
WHERE estado > (SELECT AVG(estado) FROM S);
```

25. Muestra la cantidad media por envío de cada pieza.

```
SELECT pn AS pieza, AVG(cant) AS media_envio
FROM SP
GROUP BY pn;
```

26. Muestra la cantidad media por envío realizada por cada proveedor.

```
SELECT sn AS proveedor, AVG(cant) AS media_envio
FROM SP
GROUP BY sn;
```

27. Muestra la cantidad media por envío de aquellos proveedores que realizan más de dos envíos.

```
SELECT sn AS proveedor, AVG(cant) AS media_envio
FROM SP
GROUP BY sn
HAVING COUNT(sn) > 2;
```

28. Para cada ciudad en la que se almacenan piezas, obtener el número de piezas que se almacenan de cada color distinto.

```
SELECT ciudad, color, COUNT(*) AS numpiezas
FROM P
GROUP BY ciudad, color;
```

29. Listado de los proveedores que suministran todas las piezas (sólo el código del proveedor).

```
SELECT sn AS proveedores
FROM SP
GROUP BY sn
HAVING COUNT(*) = (SELECT COUNT(*) FROM P);
```

30. Obtener el código de los proveedores que suministran alguna de las piezas suministradas por el proveedor S2.

```
SELECT DISTINCT sn
FROM SP
WHERE pn IN ( SELECT pn FROM SP WHERE sn = 'S2');
```

31. Obtener el código de los proveedores que suministran, al menos, todas las piezas suministradas el proveedor S2.

```
SELECT sp1.sn AS proveedores
FROM SP AS sp1, SP AS sp2
WHERE sp2.sn = 'S2'
AND sp1.pn = sp2.pn
GROUP BY sp1.sn
HAVING COUNT(*) = (SELECT COUNT(*)
                   FROM SP
                   WHERE sn = 'S2');
```

32. Obtener el código de los proveedores que suministran, al menos, todas las piezas que se almacenan en la ciudad de Londres.

```
SELECT sn
FROM SP, P
WHERE P.pn = SP.pn
AND ciudad = 'Londres'
GROUP BY sn
HAVING COUNT(*) = (SELECT COUNT(*)
                   FROM P
                   WHERE ciudad = 'Londres') ;
```

33. Obtener el nombre y el estado de los proveedores que suministran, al menos, todas las piezas suministradas por el proveedor S2.

```
SELECT S.sn, S.snombre AS nombre, S.estado
FROM S, SP AS sp1, SP AS sp2
WHERE S.sn = sp1.sn
AND sp2.sn = 'S2'
AND sp1.pn = sp2.pn
GROUP BY S.sn, S.snombre, S.estado
HAVING COUNT(*) = (SELECT COUNT(*)
                   FROM SP
                   WHERE sn = 'S2');
```

34. Obtener el nombre y la ciudad de los proveedores que suministran, al menos, todas las piezas que se almacenan en la ciudad de Londres.

```
SELECT S.sn, S.snombre, S.ciudad
FROM S, SP, P
WHERE S.sn = SP.sn
AND P.pn = SP.pn
AND P.ciudad = 'Londres'
GROUP BY S.sn, S.snombre, S.ciudad
HAVING COUNT(*) = (SELECT COUNT(*)
                   FROM P
                   WHERE ciudad = 'Londres');
```

35. Obtener el nombre de los proveedores que suministran alguna pieza verde.

```
SELECT DISTINCT snombre
FROM S, P, SP
WHERE S.sn = SP.sn
AND P.pn = SP.pn
AND P.color = 'verde';
```

36. Obtener el nombre de los proveedores que no suministran ninguna pieza verde.

```
SELECT DISTINCT snombre
FROM S
WHERE sn NOT IN (SELECT sn
                 FROM SP, P
                 WHERE SP.pn = P.pn
                 AND color = 'verde' );
```

37. Obtener el nombre de las piezas de color rojo suministradas por los proveedores de la ciudad de Londres.

```
SELECT DISTINCT pnombre
FROM S, P, SP
WHERE S.sn = SP.sn
AND P.pn = SP.pn
AND P.color = 'rojo'
AND S.ciudad = 'Londres';
```

38. Nombre y código de los proveedores que no suministran la pieza con el código P1 .

```
SELECT sn, snombre
FROM S
WHERE sn NOT IN (SELECT sn
                 FROM SP WHERE SP.pn = 'P1' );
```

39. Datos de los proveedores que sólo suministran piezas de color rojo.

```
SELECT *
FROM S
WHERE sn IN (SELECT sn
             FROM SP, P
             WHERE SP.pn = P.pn
             AND P.color = 'rojo' )
AND sn NOT IN (SELECT sn
               FROM SP, P
               WHERE SP.pn = P.pn
               AND color <> 'rojo');
```

Ejercicio 28 - Titulaciones

Dado el siguiente modelo relacional, se pide:

Persona (CI, nombre, apellido, ciudad, direccioncalle, direccionnum, telefono, fechanac, sexo, tipo)
Asignatura (ID, nombre, créditos, cuatrimestre, costobasico, ***IdProfesor***, ***IdTitulacion***, fechacurso)
Titulacion (Id, nombre)
Alumno_Asignatura(Id, ***IdPersona, IdAsignatura***, numeromatricula)

El atributo tipo en la entidad persona puede ser profesor o alumno.

Nota: Las claves foráneas en los modelos relacionales presentes en este documento se representan en cursiva y negrita.

1. Obtener los créditos de las asignaturas.

```
SELECT Creditos
FROM ASIGNATURA;
```

2. Nombres y créditos de las asignaturas.

```
SELECT Nombre,Creditos
FROM ASIGNATURA;
```

3. Posibles distintos créditos de asignaturas.

```
SELECT DISTINCT Creditos
FROM ASIGNATURA;
```

4. Todos los datos de personas.

```
SELECT *
FROM PERSONA;
```

5. Mostrar la suma de todos los créditos de las asignaturas.

```
SELECT SUM(Creditos)
FROM ASIGNATURA;
```

6. Media de coste básico.

```
SELECT AVG(CosteBasico)
FROM ASIGNATURA;
```

7. Coste básico de la asignatura más cara.

```
SELECT MAX(CosteBasico)
FROM ASIGNATURA;
```

8. Coste básico de la asignatura más barata.

```
SELECT MIN(CosteBasico)
FROM ASIGNATURA;
```

9. Cuantos costes básicos hay.

```
SELECT COUNT(CosteBasico)
FROM ASIGNATURA;
```

10. Cuantos costes básicos distintos hay.

```
SELECT COUNT(DISTINCT(CosteBasico))
FROM ASIGNATURA;
```

11. Cuantas asignaturas hay.

```
SELECT COUNT(*)
FROM ASIGNATURA;
```

12. Suma de los créditos de las asignaturas llamando a la columna resultante "TotalCreditos".

```
SELECT SUM(Creditos) TotalCreditos
FROM ASIGNATURA;
```

13. Nombre y créditos de las asignaturas del primer cuatrimestre.

```
SELECT Nombre, Creditos
FROM ASIGNATURA
WHERE Cuatrimestre = '1';
```

14. Nombre y Apellido de las personas nacidas antes del 1 de enero de 1975.

```
SELECT Nombre Apellido
FROM PERSONA
WHERE FechaNacimiento < '1/01/1975';
```

15. Nombre y coste básico de las asignaturas de más de 4,5 créditos.

```
SELECT Nombre, CosteBasico
FROM ASIGNATURA WHERE Creditos > 4.5;
```

16. Nombre de las asignaturas cuyo coste básico esté entre 25 y 35 Bs.

Solución 1:

```
SELECT Nombre
FROM ASIGNATURA
WHERE CosteBasico >= 25 AND CosteBasico <= 35;
```

Solución 2:

```
SELECT Nombre
FROM ASIGNATURA
WHERE CosteBasico BETWEEN 25 AND 35;
```

17. Id de los alumnos matriculados en la asignatura 150212 ó en la 130113..

```
SELECT DISTINCT(IdAlumno)
FROM ALUMNO_ASIGNATURA
WHERE IdAsignatura = '150212'
OR IdAsignatura = '130113';
```

18. Nombres de las asignaturas del 2º cuatrimestre que no sean de 6 créditos.

```
SELECT Nombre
FROM ASIGNATURA
WHERE Cuatrimestre = '2'
AND Creditos != 6;
```

19. Nombre de las asignaturas cuyo coste básico esté entre 25 y 35 Bs.

```
SELECT Nombre
FROM ASIGNATURA
WHERE CosteBasico BETWEEN 25 AND 35;
```

20. Id de los alumnos matriculados en la asignatura 150212 ó en la 130113.

```
SELECT IdAlumno
FROM ALUMNO_ASIGNATURA
WHERE IdAsignatura IN ('150212', '130113');
```

21. Nombre y apellido de las personas cuyo apellido comience por 'G'.

```
SELECT Nombre, Apellido
FROM PERSONA
WHERE Apellido LIKE 'G%';
```

22. Nombre de las asignaturas que no tienen dato para el IdTitulacion.

```
SELECT Nombre
FROM ASIGNATURA WHERE IdTitulacion IS NULL;
```

23. Nombre de las asignaturas que tienen dato para el IdTitulacion.

```
SELECT Nombre
FROM ASIGNATURA
WHERE IdTitulacion IS NOT NULL;
```

24. Nombre de las asignaturas cuyo coste por crédito sea mayor de 8 Bs.

```
SELECT Nombre
FROM ASIGNATURA
WHERE (CosteBasico/Creditos) > 8;
```

25. Nombre y número de horas de todas las asignaturas. (1cred.=10 horas).

```
SELECT Nombre, Creditos*10 AS NUM_HORAS
FROM ASIGNATURA;
```

26. Mostrar el nombre completo en una columna y la dirección completa en otra columna de las personas.

```
SELECT Nombre||' '||Apellido,
       DireccionCalle||','||DireccionNum||' - '||Ciudad
FROM PERSONA;
```

27. Datos de personas por orden alfabético de apellidos.

```
SELECT *
FROM PERSONA
ORDER BY Apellido;
```

28. Datos de las asignaturas del 2º cuatrimestre por orden alfabético de IdTitulacion. Para cada titulación ordenar por coste mostrando primero las asignaturas más caras y para las asignaturas del mismo coste mostrar por orden alfabético de nombre de asignatura.

```
SELECT *
FROM ASIGNATURA
WHERE Cuatrimestre = '2'
ORDER BY IdTitulacion, CosteBasico DESC, Nombre;
```

29. Nombre y apellidos de los profesores.

```
SELECT Nombre, Apellido
FROM PERSONA, PROFESOR
WHERE PERSONA.DNI = PROFESOR.DNI;
```

30. Nombre de asignaturas impartidas por profesores de Caracas.

```
SELECT Asignatura.Nombre
FROM ASIGNATURA, PROFESOR, PERSONA
WHERE ASIGNATURA.IdProfesor = PROFESOR.IdProfesor
AND PROFESOR.DNI = PERSONA.DNI
AND Ciudad = 'Caracas';
```

31. Nombre completo (nombre y apellido), en una sóla columna, de los alumnos.

```
SELECT Nombre||' '||Apellido
FROM PERSONA, ALUMNO
WHERE PERSONA.DNI = ALUMNO.DNI;
```

32. DNI, Nombre y Apellido de los alumnos que viven en la Gran Caracas. Como criterio usar el teléfono.

```
SELECT PERSONA.DNI AS DNI, Nombre, Apellido
FROM PERSONA, ALUMNO
WHERE PERSONA.DNI = ALUMNO.DNI
AND Telefono LIKE '0212%';
```

33. DNI, Nombre y Apellido de los alumnos matriculados en la asignatura "Seguridad Vial".

```
SELECT PERSONA.DNI AS DNI_, PERSONA.Nombre, Apellido
FROM PERSONA, ALUMNO, ALUMNO_ASIGNATURA, ASIGNATURA
WHERE PERSONA.DNI = ALUMNO.DNI
AND ALUMNO.IdAlumno = ALUMNO_ASIGNATURA.IdAlumno
AND ALUMNO_ASIGNATURA.IdAsignatura = ASIGNATURA.IdAsignatura
AND ASIGNATURA.NOMBRE = 'Seguridad Vial';
```

34. Id de las titulaciones en las que está matriculado el alumno con DNI 20202020A. Un alumno está matriculado en una titulación si está matriculado en una asignatura de la titulación.

```
SELECT DISTINCT(IdTitulacion)
FROM ASIGNATURA, ALUMNO_ASIGNATURA, ALUMNO
WHERE ASIGNATURA.IdAsignatura = ALUMNO_ASIGNATURA.IdAsignatura
AND ALUMNO_ASIGNATURA.IdAlumno = ALUMNO.IdAlumno
AND ALUMNO.DNI = '20202020A';
```

35. Nombre de las asignaturas en las que está matriculada Rosa García, mostrando también por cada asignatura, el número de matrícula.

```
SELECT ASIGNATURA.Nombre,ALUMNO_ASIGNATURA.IdAlumno
FROM PERSONA, ALUMNO, ALUMNO_ASIGNATURA, ASIGNATURA
WHERE PERSONA.DNI = ALUMNO.DNI
AND ALUMNO.IdAlumno = ALUMNO_ASIGNATURA.IdAlumno
```

```
AND ALUMNO_ASIGNATURA.IdAsignatura = ASIGNATURA.IdAsignatura
AND PERSONA.Nombre = 'Rosa'
AND PERSONA.Apellido = 'García';
```

36. DNI de los alumnos a los que imparte el profesor Jorge Ruiz.

Solución 1:

```
SELECT ALUMNO.DNI AS DNI_ALUMNO
FROM PERSONA, ALUMNO, ALUMNO_ASIGNATURA, ASIGNATURA, PROFESOR
WHERE PERSONA.DNI = PROFESOR.DNI
AND ALUMNO.IdAlumno = ALUMNO_ASIGNATURA.IdAlumno
AND ALUMNO_ASIGNATURA.IdAsignatura = ASIGNATURA.IdAsignatura
AND ASIGNATURA.IdProfesor = PROFESOR.IdProfesor
AND PERSONA.Nombre = 'Jorge'
AND PERSONA.Apellido = 'Ruiz';
```

Solución 2:

```
SELECT ALUMNO.DNI AS DNI_ALUMNO
FROM ALUMNO, ALUMNO_ASIGNATURA, ASIGNATURA
WHERE ALUMNO.IdAlumno = ALUMNO_ASIGNATURA.IdAlumno
AND ALUMNO_ASIGNATURA.IdAsignatura = ASIGNATURA.IdAsignatura
AND ASIGNATURA.IdProfesor IN (SELECT PROFESOR.IdProfesor
                              FROM PERSONA, PROFESOR
                              WHERE PERSONA.DNI = PROFESOR.DNI
                              AND PERSONA.Nombre = 'Jorge'
                              AND PERSONA.Apellido = 'Sáenz');
```

37. ¿Cuál sería el coste global de cursar la titulación de Bases de Datos si el coste de cada asignatura fuera incrementado en un 7%?

```
SELECT SUM(CosteBasico * 1.07)
FROM ASIGNATURA, TITULACION
WHERE ASIGNATURA.IdTitulacion = TITULACION.IdTitulacion
AND TITULACION.Nombre = 'Bases de Datos';
```

Referencias Bibliográficas

[1] R. Elmasri & S. Navathe. Fundamentos de Sistemas de Bases de Datos. Madrid: Addison Wesley. Quinta Edición. 2007.

[2] A. Oppel & R. Sheldon. Fundamentos de SQL. México: McGRAW-HILL. Tercera Edicion. 2010.

[3] A relational model of data for large shared data banks “A Relational Model Of Data For Large Shared Data Banks” (Un modelo relacional de datos para grandes bancos de datos compartidos), en el diario Communications of the ACM, volumen 13, número 6 (junio de 1970).

[4] A. Silberschatz, H. Korth & S. Sudarshan. Fundamentos de Bases de Datos. Madrid:McGRAW-HILL. Cuarta Edición. 2002.

[5] D.D Chamberlain, SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control, IBM Journal of Research and Development. , vol. 20, no. 6, noviembre 1976.

[6] C. Gonzalez. Sistemas de Base de Datos. Editorial Tecnológica de Costa Rica 1996.

[7] C. Coronel, S Morris & P. Rob. Bases de datos diseño, implementación y administración. México: Cengage Learning. Novena edición. 2011.

[8] M. Villapecellin. Desarrollo de aplicaciones en entornos de 4ª generación con herramientas Case. RA-MA S.A. Editorial y Publicaciones. 2004.

[9] B. Sotomayor. Ejercicios Resueltos de SQL. 2002

[10] C. Gutiérrez. Guía de ejercicios de SQL.2008.
Recuperado de: https://users.dcc.uchile.cl/~mnmonsal/BD/maestra/guiaSQL.pdf

[11] G. Quintana, M. Marqués, J. Aliaga & M. Aramburu. Aprende SQL. Castellón de la Plana. Publicacions de Universitat Jaume. 2010

[12] Guía para la instalación, y utilización del sistema gestionador de bases de datos (DBMS) postgreSQL. Recuperado de:
http://postgresql-dbms.blogspot.com/p/una-vez-terminada-la-instalacion-del_6441.html

[13] M. Fernández. Guía de ejercicios SQL. Cali: Universidad del Valle. 2012.

[14] Guía de ejercicios para resolver SQl. 2011. Recuperado de: https://kesquivel.files.wordpress.com/2011/03/clasepractica2_sql_2011_solucion1.pdf

[15] Ejercicios SQL. Recuperado de: https://es.slideshare.net/kyuubi92/ejercicios-resueltos-de-sql

[16] Algebra relacional y SQL. 2006. Recuperado de: https://www.academia.edu/7277162/Algebra_Relacional_ejercicios_Resueltos

[17] Y. Nava. Ejercicios SQL . Recuperado de: https://www.academia.edu/25941453/EJERCICIOS_DE_SQL_CON_RESPUESTAS

[18]Curso de SQL. Recuperado de: https://www.aulaclic.es/sql/t_1_2.htm

[19] Bases de Datos Ejercicios SQL. Recuperado de: https://www.mastergeoinformatica.es/wp-content/uploads/2016/06/FSI-BD-T7-EjerciciosSQLResueltos.pdf

[20] E. Chinkes, E. Alarcon, L. Coronel, D. Contreras & E. Golmand. Guía de SQL. 2008. Recuperado de: https://www.academia.edu/36354931/GUIA_DE_SQL_DDL_DATA_DEFINITION_LANGUAGE_DML_DATA_MANIPULATION_LANGUAGE

[21] Ejercicios SQL. Recuperado de: https://www.ces-vegamedia.es