SISTEMAS DE BASES DE DATOS - INGENIERÍA INFORMÁTICA

IMPLEMENTACIÓN Y PROGRAMACIÓN DE BASES DE DATOS RELACIONALES.

Los módulos previos le han permitido profundizar el análisis de los diferentes modelos de bases de datos y, en particular, del modelo relacional.
Ahora bien, ¿este modelo conceptual teórico se puede aplicar con la tecnología actual? El modelo es fruto de muchos años de investigación, y recién a finales del siglo pasado fue posible su implementación tecnológica. Esta implementación no ha sido completa, ni aún hoy, pero ha logrado soluciones que justifican totalmente su utilización. Esa misma pregunta puede hacerse para cada modelo, pero aquí nos interesa en particular el modelo relacional.

Así, en este módulo vamos a analizar cada uno de sus aspectos y cómo se realiza su implementación tecnológica.
Veremos que el modelo conceptual ha ido evolucionando y que esta evolución, a veces, ha surgido como consecuencia de una solución tecnológica anterior; es decir que los modelos conceptuales y los modelos tecnológicos son al mismo tiempo causa y efecto unos de otros, y van entrelazando ideas y soluciones que permiten mejoras continuas en el software que utilizamos para la gestión de datos.

Finalmente, las actividades planteadas le permitirán adquirir un conjunto de habilidades en la aplicación de los conocimientos adquiridos.
Es decir, este módulo es fundamentalmente práctico, y la idea es que Ud. pueda implementar una base de datos relacional que satisfaga los requerimientos de una aplicación “realista”, con el objeto de que adquiera una experiencia útil para cuando se encuentre en una situación laboral real.
Es importante justificar por qué la aplicación debe ser realista y no “ideal” o “real”. En el caso del desarrollo de sistemas o aplicaciones, las situaciones “ideales” pueden servir como ejemplos limitados para aprender conceptos muy particulares, pero es fundamental realizar experiencias aplicadas a situaciones reales ya que éstas contienen los detalles que hacen complejo el diseño (fundamentalmente) y la construcción de los sistemas.
Al no poder plantear situaciones reales por problemas de implementación, ya que se debería contar con alguna organización que les permita realizar esta experiencia, y además se debería encontrar el sistema o aplicación que se adapte en cuando al tipo y complejidad requeridos, entonces plantearemos una situación a partir de un conjunto de supuestos y requerimientos realistas, pero para una organización ideal.

El tema central de este módulo será la programación de bases de datos con SQL, resultado de la implementación tecnológica de los conceptos de álgebra relacional y cálculo relacional.

El dominio de este lenguaje es fundamental en el procesamiento de datos almacenados en bases de datos relacionales y solo se logra si se entiende claramente la implementación tecnológica de cada concepto del AR y el CR.
Para esto, le proporcionamos aquí un material en el que tratamos de explicar la sintaxis y aplicación de este lenguaje en base a los conceptos sobre los cuales éste se basa.

Trataremos de utilizar una sintaxis lo más estándar posible, de tal manera que Ud. pueda aplicar los diferentes ejemplos al RDBMS con el cual realizará la práctica.
El desarrollo de este material se ha basado en los conceptos vertidos en unidades anteriores de la materia con el fin de llevar a la práctica toda la base conceptual adquirida. Por lo tanto sería conveniente que antes de avanzar en el mismo, revise los conceptos correspondientes al modelo relacional, fundamentalmente los temas referidos al álgebra y cálculo relacional.

No incluiremos aquí, la extensión de SQL correspondientes a instrucciones de lógica de control procedural, ni tratamiento por registros o filas a través de cursores, dejando esto para la próxima asignatura.
El dominio de este lenguaje es fundamental para el procesamiento de datos almacenados en bases de datos relacionales y solo se logra si se entiende claramente la implementación tecnológica de cada concepto del AR y el CR.
Para esto, le proporcionamos aquí un material en el que tratamos de explicar la sintaxis y aplicación de este lenguaje en base a los conceptos sobre los cuales éste se basa.

Trataremos de utilizar una sintaxis lo más estándar posible, de tal manera que Ud. pueda aplicar los diferentes ejemplos al RDBMS con el cual realizará la práctica.

No incluiremos aquí, la extensión de SQL correspondientes a instrucciones de lógica de control procedural, ni tratamiento por registros o filas a través de cursores.

SQL (Structure Query Language)

Habiendo revisado distintos aspectos del tratamiento de datos a través del álgebra relacional y del cálculo relacional, iniciaremos ahora el camino de las técnicas de programación con SQL.

¿Qué es SQL? Es un lenguaje de alto nivel para el tratamiento de datos almacenados en bases de datos relacionales.
Es un lenguaje no procedural que permite trabajar con datos a nivel de conjunto, a diferencia de los lenguajes tradicionales que trabajan a nivel de registro individual y que requieren de lógica procedural para el tratamiento de los datos.
Está basado en álgebra relacional y cálculo relacional orientado a tuplas.
Sus instrucciones básicas se pueden dividir en dos grupos:

- DDL (Data Definition Language): conjunto de instrucciones que permiten definir tablas, índices, etc.
- DML (Data Manipulation Language): se utilizan para actualizar información en la base de datos (insertar, eliminar o modificar filas en tablas) y para extraer información de la misma.

Además de estos grupos existen otros, cuyas instrucciones permiten realizar otras operaciones aplicadas al DBMS y/o a sus bases de datos.

Estándar: Toda la definición de la sintaxis y uso de SQL está incluida en estándares que van actualizándose periódicamente. Actualmente el último estándar aceptado es SQL:2003. Muchos de los RDBMSs del mercado satisfacen gran parte de estándares anteriores (SQL/92, SQL/99) y otros este último estándar. Pero, además, varios de los RDBMSs líderes en el mercado tales como DB2 de IBM, ORACLE, SQL Server de Microsoft y otros, tienen funcionalidades que van más allá de los estándares y, en base a estas funcionalidades, muchas veces se elaboran nuevos estándares.

Este material tiene como premisa acercarse lo máximo posible a los estándares definidos, pero sin hacer mención de alguno de ellos en particular.
La idea es explicar el sentido de cada instrucción y las técnicas para su utilización adecuada. Luego, analizando los manuales del RDBMS con el cual haya decidido trabajar, podrán encontrar y utilizar la instrucción específica del mismo que le de soporte a la misma funcionalidad.

DDL (Data Definition Language):

Solo nos interesa definir algunos aspectos relacionados con la creación de tablas, índices, vistas y snapshots.

- **CREACIÓN DE TABLAS:**

La creación de tablas se realiza con la instrucción CREATE TABLE. A través de ellas se especifican:

- Identificación (nombre), ubicación (base de datos), propietario
- Esquema de la tabla (tipo de registro → descripción de sus columnas)
- Reglas de integridad declarativas tales como: identificadores (claves primarias y alternativas), asociaciones con otras tablas (claves foráneas) y reglas de integridad específicas de la tabla (checks)
- Características del almacenamiento físico

La instrucción es bastante simple y lo que deseamos destacar aquí son simplemente algunas cuestiones que son importantes a la hora de crear una tabla.
Estas cuestiones están relacionadas con las reglas de integridad:

- Regla de integridad de dominio:

  Cuando hablamos de dominio, estamos hablando de un tipo de datos semántico. Su implementación tecnológica, en la mayoría de los casos es incompleta y solo se lo logra implementar como tipo de dato, ya sea, utilizando los tipos de datos básicos provistos por el DBMS (integer, char, date, etc.) a los cuales se les agrega una regla de integridad (denominada integridad de dominio) a través de una cláusula CHECK o como tipo de dato definido por el usuario, el cual es definido previamente y reutilizado para la especificación de columnas en diferentes tablas. En estos casos el dominio es estático y solo puede modificarse alterando la definición de la tabla (ver ALTER TABLE) o modificando el tipo de dato definido por el usuario, tarea que le compete al DBA (Database Administrator).
  Un dominio dinámico y ajustado por los usuarios se puede implementar a través de una tabla de dominio, en cuyas filas se almacenan los valores posibles de un atributo determinado. Los valores válidos para el atributo (columna) son verificados a través de otra regla de integridad (regla de integridad de referencia) implementada a través de una clave foránea.
  La ventaja de esta implementación es que el usuario responsable de ese dominio puede ampliar y restringir los valores del mismo a través de la misma aplicación, sin depender del DBA.

- Reglas de integridad de entidad:

  Tanto la clave primaria como las alternativas son claves candidatas. ¿Recuerda las propiedades de una clave candidata? Si no las recuerda, vuelva sobre estos conceptos antes de continuar.

En forma declarativa se pueden definir ambas. Hay dos formas de definirlas y éstas dependen de la cantidad de columnas que la componen (si son simples -una columna- o compuestas -varias columnas-).
Una clave candidata compuesta por una única columna se puede definir como parte de la definición de la columna.
Si la clave candidata es compuesta, entonces se debe declarar, luego de la definición de todas las columnas de la tabla.

¿Recuerda si el orden de las columnas en la definición de una clave candidata compuesta es relevante? Si no lo recuerda, vuelva a revisar los conceptos de diseño.

¿Recuerda si es conveniente agregar una columna identificadora única para definir una clave primaria de una tabla o será mejor declarar como clave primaria alguna de las claves candidatas de la misma? Si no lo recuerda, vuelva a revisar los conceptos de diseño.

- Reglas de integridad de referencia:

Una clave foránea permite asociar información que está distribuida en dos tablas a través de una referencia establecida de una a la otra (o a ella misma) por medio de un conjunto de columnas cuyo dominio es común, permitiendo que instancias de la primera tabla se asocien a instancias de la segunda a través de los valores de dichas columnas.

Las claves foráneas se producen en los procesos de normalización y se utilizan para la reconstrucción de la relación original a través de la operación JOIN.

Una clave foránea referencia a la clave primaria de otra relación (o de ella misma), pero también se puede referenciar a través de una clave foránea a cualquier otra clave candidata (alternativa) de otra relación (o de ella misma).

Como en el caso de las claves candidatas, las claves foráneas pueden ser simples o compuestas. También en este caso, la declaración de una clave foránea simple puede ser definida como parte de la definición de una columna y si es compuesta debe declararse al final de la especificación de todas las columnas.

¿Puede responder a la siguiente pregunta?: ¿Cuáles de las siguientes afirmaciones son correctas?

Una clave foránea debe corresponderse con la clave (primaria o alternativa) referenciada a través de:

- nombre de las columnas (las columnas de la clave foránea deben tener el mismo nombre que las columnas de la clave referenciada)

- orden de las columnas (las columnas de la clave foránea deben estar especificadas en el mismo orden que las columnas correspondientes de la clave referenciada sin importar su nombre)

- tipo de dato de las columnas (el tipo de dato de cada columna de la clave foránea debe ser igual al de la columna correspondiente de la clave referenciada)

Si no pudo responderla, vuelva a revisar los conceptos de diseño.

¿Puede responder a la siguiente pregunta?: ¿Cuáles de las siguientes afirmaciones son correctas y por qué?

1. Un sub-conjunto propio de una clave primaria puede ser una clave alternativa en la misma tabla.
2. En una tabla dos claves candidatas distintas pueden estar solapadas (comparte uno o varios atributos).
3. Un sub-conjunto propio de una clave primaria puede ser una clave foránea.
4. Un sub-conjunto propio de una clave foránea puede ser otra clave foránea.
5. En una tabla dos claves foráneas distintas pueden estar solapadas (comparte uno o varios atributos).
6. Una clave foránea puede referenciar a un sub-conjunto de una clave primaria

Si no pudo responderla, vuelva a revisar los conceptos de diseño.

¿Puede responder a la siguiente pregunta?: Si una clave foránea es no obligatoria (cada fila de la tabla que contiene la clave foránea está asociada a cero o una fila correspondiente a la tabla referenciada, ¿cómo son sus atributos?

Si no pudo responderla, vuelva a revisar los conceptos de diseño.

- Reglas de integridad específicas:

Las reglas de integridad específicas no corresponden a reglas de integridad del modelo (no se aplican todas las tablas de cualquier base de datos relacional) sino que se aplican a determinadas tablas de una base de datos en particular.

Estas reglas de integridad son programadas de dos formas:

- Declarativa: solo se utiliza esta forma si la regla se aplica, o debe ser evaluada para una o varias columnas de cada fila correspondiente a una tabla.

  Se especifican a través de una expresión booleana que describe las distintas alternativas válidas. Es decir, las distintas combinaciones de valores de las columnas involucradas que verifican la expresión (el resultado es verdadero). Para reducir la expresión booleana se utiliza álgebra de Boole.

- Procedural: si la regla se debe verificar evaluando varias filas ya sea en una única tabla o correspondientes a varias tablas, la forma es crear una regla de integridad procedural a través de triggers o procedimientos almacenados.

Ejemplos de creación de tablas utilizando Microsoft SQL Server 2019:

```
CREATE TABLE deptos
(
 nro_depto          integer        not null      primary key,
 nom_depto          varchar(30)    not null      unique
)

CREATE TABLE cargos
(
 cod_cargo          varchar(3)     not null,
 nom_cargo          varchar(40)    not null,
 directivo          char(1)        not null      check (directivo in ('S','N')),
 primary key (nro_cargo),
 unique (nom_cargo)
)

CREATE TABLE emp
(
 nro_depto          integer              not null      references deptos,
 nro_emp            integer              not null,
 nom_emp            varchar(30)          not null,
```

```
    fecha_ing           date                not null,
    cod_cargo           varchar(3)          not null,
    salario             decimal(10,2)       not null        check (salario > 0.00),
    porc_comision       decimal(4,2)        null,
    primary key (nro_depto, nro_emp),
    foreign key (cod_cargo) references cargos,
    check (salario >= 2000.00 and porc_comision is null or
            salario <    2000.00 and porc_comision > 0.00)
  )
```

En este momento le aconsejamos conceptos de estructuras de almacenamiento y analizar la instrucción CREATE TABLE en su DBMS con el fin de determinar cuáles de las estructuras de almacenamiento son aceptadas y manejadas por el mismo y como ellas son especificadas en el momento de la creación de la misma.

También será conveniente que elabore un diagrama de entidades y relaciones (DER) a partir de las tablas indicadas, lo que le permitirá entender la implementación de ese diseño.

NOTA: Los detalles de las estructuras de almacenamiento y su utilidad son tratadas en profundidad en la unidad dedicada a optimización.

¿Como eliminar una tabla?

Usar la instrucción DROP TABLE nom_tabla

¿En qué orden se deberán eliminar las tablas?

Las tablas solo pueden ser creadas si existen previamente las tablas a las cuales referencia y solo podrán ser eliminadas si no tienen referencias de otras tablas, por lo tanto, el orden de eliminación será el inverso al de creación.
NOTA: Hay una alternativa que permite la creación y eliminación de tablas en cualquier orden. Buscar en el manual de su DBMS cuál es esa alternativa.

Le aconsejamos en este momento buscar en los manuales de su DBMS otras alternativas de creación de tablas.

- **CREACIÓN DE ÍNDICES:**

Una de las estructuras de almacenamiento más usadas por los RDBMSs es aquella que utiliza índices (en su gran mayoría alguna variante de árbol B).
Los índices permiten un acceso más rápido a datos ordenados, fundamentalmente cuando los requerimientos son muy selectivos, pero tienen un costo a la hora de las actualizaciones, ya que muchas veces, la actualización de la tabla requiere la actualización de uno o varios índices.

Los índices se crean utilizando la instrucción CREATE INDEX.

Le aconsejamos en este momento revisar conceptos sobre índices y analizar la instrucción CREATE INDEX en su DBMS con el fin de determinar las variantes que éste provee.

NOTA: Los tipos y usos de índices son tratadas en profundidad en la unidad dedicada a optimización.

DML (Data Manipulation Language):

En este bloque se incluyen las cuatro instrucciones básicas de SQL para el tratamiento de datos almacenados en tablas.

Ellas son: INSERT, SELECT, DELETE, UPDATE.

Usaremos la siguiente especificación de tablas para el análisis de estas instrucciones:

DEPTOS (nro_depto, nom_depto)
Primary Key: nro_depto

EMP (nro_emp, nom_emp, fecha_ing, cargo, sueldo, nro_depto)
Primary Key: nro_emp
Foreign Key: nro_depto REFERENCIA a DEPTOS

DEPTOS

nro_depto	**nom_depto**
10	CONTABILIDAD
20	SISTEMAS
30	VENTAS
40	COMPRAS

EMP

nro_emp	nom_emp	fecha_ing	cargo	sueldo	nro_depto
1	PEREZ	01/11/1999	GERENTE	1700.00	10
2	GOMEZ	10/01/1998	ADMINISTRATIVO	1200.00	10
3	RODRIGUEZ	10/02/1999	VENDEDOR	1000.00	30
4	JUAREZ	10/03/1999	PROGRAMADOR	1100.00	20
5	URRUTI	01/02/2000	VENDEDOR	900.00	30
6	CASTRO	01/02/2000	VENDEDOR	900.00	30
7	GIMENEZ	30/10/2001	ANALISTA	1200.00	20
8	LINARES	10/05/2001	ADMINISTRATIVO	700.00	10
9	TORRES	01/06/2002	RESP. VENTAS	1500.00	30
10	VILCHEZ	01/01/2000	RESP. SISTEMAS	1600.00	20

Le recomendamos que programe la creación de estas tablas, de tal manera de poder realizar la programación y prueba de los ejemplos mencionados en este material. Elegir los tipos de datos adecuados en cada caso, según los requerimientos de los datos a insertar.

- INSERT:

  Hay varias sintaxis disponibles para la instrucción INSERT. La más común es aquella que permite insertar una fila a través de valores constantes.

  INSERT INTO tabla (lista de columnas)
  VALUES (lista de valores)

  Lo importante para destacar en esta variante es que la lista de columnas debe incluir a todas aquellas columnas obligatorias definidas sin valores por defecto.
  Esta lista es opcional, si no se especifica, se supone que se insertarán valores para cada una de las columnas que componen la tabla, en el orden en el cual ellas fueron especificadas en la creación de la misma.
  La lista de valores debe corresponder en número y orden con la lista de columnas (o con las columnas de la tabla en el orden en el cual ellas fueron creadas, si no se especifica la lista de columnas).

  Le aconsejamos en este momento revisar en los manuales de su DBMS, otras alternativas de inserción.

Veamos cómo se programan las instrucciones para insertar los datos de cada una de las tablas:

```
INSERT INTO deptos (nro_depto, nom_depto)
VALUES (10, 'CONTABILIDAD')
....
```

continúe Ud. con el resto de las inserciones en la tabla DEPTOS.

```
INSERT INTO emp (nro_emp, nom_emp, fecha_ing, cargo, sueldo, nro_depto)
VALUES (1, 'PEREZ', '1999-11-01', 'GERENTE', 1700.00, 10)
...
```

continúe Ud. con el resto de las inserciones en la tabla EMP.

Aclaraciones:

- La instrucción:

  ```
  INSERT INTO deptos (nom_depto)
  VALUES ('CONTABILIDAD')
  ```

  es incorrecta, ya que no se informa ningún valor para el número de departamento, dato que es obligatorio.

- La instrucción:

  ```
  INSERT INTO deptos
  VALUES (10, 'CONTABILIDAD')
  ```

  es correcta

- La instrucción:

  ```
  INSERT INTO deptos
  VALUES ('CONTABILIDAD', 10)
  ```

  es incorrecta. El DBMS, emitirá un error indicando que no se podrá insertar un valor no numérico (CONTABILIDAD) en una columna con tipo de dato numérico (nro_depto).

- La instrucción:

INSERT INTO deptos (nom_depto, nro_depto)
VALUES ('CONTABILIDAD', 10)

es correcta.

- La instrucción:

  INSERT INTO emp (nro_emp, **cargo**, fecha_ing, **nom_emp**, sueldo, nro_depto)
  VALUES (1, 'PEREZ', '1999-11-01', 'GERENTE', 1700.00, 10)

  es correcta sintácticamente y el DBMS la insertará sin problemas, ya que a pesar de que las columnas están invertidas, ambas están definidas con un tipo de dato que acepta la información que se intenta insertar.
  Esta instrucción es incorrecta, en cuanto a que los datos insertados no lo están en las columnas correspondientes.
  NOTA: Esto mismo podría ocurrir con nro_emp y nro_depto, incluso habiendo una clave foránea, ya que si el valor existe como nro_depto en la tabla DEPTOS, esta regla de integridad no dará error.
  NOTA: Verifique que el empleado no haya sido insertado ya, porque en ese caso, el DBMS dará un error por intentar insertar una fila con una clave primaria que ya existe en alguna fila ya insertada en la tabla. Si el DBMS da un error: "clave primaria duplicada", intente ejecutar la misma instrucción, pero con otro número de empleado.
  NOTA: Asegúrese de que el nro. de departamento 10 exista en la tabla DEPTOS, ya que, si no existe, el DBMS devolverá un error relacionado con la clave foránea.

- El formato de la fecha ingresada puede no ser válido en el DBMS sobre el cual intenta ejecutar la instrucción. Verifique previamente cual es el o los formatos admitidos y modifique el valor a insertar.

- Si alguna de las columnas fuera no obligatoria, y no se deseara insertar información en ella, entonces podría eliminarse de la lista de columnas.

- <u>SELECT</u>:

  La sintaxis básica de la instrucción SELECT es la siguiente:

  SELECT lista de expresiones escalares de resultado

FROM lista de expresiones tabulares fuente
[WHERE expresión booleana relacional]
[GROUP BY lista de expresiones escalares]
[HAVING expresión booleana relacional]
[ORDER BY lista de expresiones escalares de ordenamiento]

[] indica que la cláusula es opcional

Vamos a analizar las características de esta instrucción basándonos en lo aprendido acerca de AR y CR y aplicaremos estos conocimientos a las tablas del ejemplo incluido arriba. Para eso iremos planteando diferentes requerimientos, resolviéndolos en AR y CR y luego en SQL y encontrando distintas variantes para ello.

Comencemos con una primera consulta básica:

1. Mostrar todos los atributos de todos los empleados.

HERRAMIENTA	SOLUCIÓN
AR	1. emp [nro_emp, nom_emp, fecha_ing, cargo, sueldo, nro_depto] 2. emp [*]
CR	1. e.nro_emp, e.nom_emp, e.fecha_ing, e.cargo, e.sueldo, e.nro_depto 2. e 3. e.*
SQL	1. SELECT nro_emp, nom_emp, fecha_ing, cargo, sueldo, nro_depto FROM emp 2. SELECT emp.nro_emp, emp.nom_emp, emp.fecha_ing, emp.cargo, emp.sueldo, emp.nro_depto

	FROM emp 3. SELECT e.nro_emp, e.nom_emp, e.fecha_ing, e.cargo, e.sueldo, e.nro_depto FROM emp e 4. SELECT emp.* FROM emp 5. SELECT e.* FROM emp e 6. SELECT * FROM emp

Álgebra relacional:

- La operación utilizada es una proyección.
- La variante con * no es válida en el AR, pero la agregamos para asimilarla a la solución en SQL

Cálculo relacional:

- La variable "e" es una variable de tupla basada en la tabla emp
- La variante con * no es válida en el CR, pero la agregamos para asimilarla a la solución en SQL

SQL:

- Como se ve, la cláusula SELECT actúa de forma similar a la operación PROYECCIÓN del AR. Luego veremos que también actúa como la operación EXTENSIÓN, ya que permite agregar, además de columnas, constantes y expresiones escalares.
- La cláusula FROM permite especificar la o las tablas desde donde se extraerán los datos solicitados o que se utilizarán en la selección de los mismos.
- La letra "e" seguida al nombre de la tabla "emp" es un alias y permite simplificar el nombre para su utilización en la calificación de las columnas y para evitar ambigüedades cuando se utiliza más de una vez una misma tabla

en la misma instrucción. Si a una tabla se le da un alias, luego se la debe indicar a través de ese alias.
- El asterisco permite indicar que los datos que se desean mostrar son todas aquellas columnas pertenecientes a todas las tablas incluidas en la cláusula FROM.

2. Mostrar nom_emp y nro_depto de todos los empleados.

HERRAMIENTA	SOLUCIÓN
AR	1. emp [nom_emp, nro_depto]
CR	1. e.nom_emp, e.nro_depto
SQL	1. SELECT nom_emp, nro_depto FROM emp 2. SELECT emp.nom_emp, emp.nro_depto FROM emp 3. SELECT e.nom_emp, e.nro_depto FROM emp e

Álgebra relacional:

- La operación utilizada nuevamente es una proyección.

SQL:

- Aquí se ve más claramente como la cláusula SELECT actúa de forma similar a la operación PROYECCIÓN del AR.
- Notar también que la utilización del nombre de la tabla o del alias para calificar a las columnas no es necesario. Solo lo será cuando hubiera ambigüedad.

3. Mostrar nom_emp y nro_depto de los empleados con sueldo entre 1000 y 2000.

HERRAMIENTA	SOLUCIÓN
AR	1. (emp WHERE sueldo >= 1000 AND sueldo <= 2000) [nom_emp, nro_depto] 2. (emp WHERE sueldo BETWEEN 1000 and 2000) [nom_emp, nro_depto]
CR	1. e.nom_emp, e.nro_depto WHERE e.sueldo >= 1000 AND e.sueldo <= 2000 2. e.nom_emp, e.nro_depto WHERE e.sueldo BETWEEN 1000 AND 2000
SQL	1. SELECT nom_emp, nro_depto FROM emp WHERE sueldo >= 1000 AND sueldo <= 2000 2. SELECT nom_emp, nro_depto FROM emp WHERE sueldo BETWEEN 1000 AND 2000 3. SELECT e.nom_emp, e.nro_depto FROM emp e WHERE e.sueldo BETWEEN 1000 AND 2000

Álgebra relacional:

- Además de la operación proyección para obtener los datos requeridos, se ha utilizado una operación restricción a través de la cláusula WHERE.

SQL:

- Encontramos aquí que el uso de la cláusula WHERE cumple la misma función que dicha cláusula tanto en el AR como en el CR.
- Notar también la similitud de la alternativa 3 con la expresión del CR, y como en éste y en los casos anteriores el alias de tabla en SQL se puede asociar a la variable de tupla en el CR.
- A través del comparador BETWEEN se puede programar una comparación de valores en un rango determinado incluidos sus límites.

4. Mostrar nom_emp, nom_emp y sueldo de los empleados con cargo 'PROGRAMADOR' o 'ANALISTA'.

HERRAMIENTA	SOLUCIÓN
AR	1. (emp WHERE cargo = 'PROGRAMADOR' or cargo = 'ANALISTA') [nro_emp, nom_emp, sueldo] 2. (emp WHERE cargo IN ('PROGRAMADOR','ANALISTA')) [nro_emp, nom_emp, sueldo]
CR	1. e.nro_emp, e.nom_emp, e.sueldo WHERE e.cargo = 'PROGRAMADOR' or e.cargo = 'ANALISTA' 2. e.nro_emp, e.nom_emp, e.sueldo WHERE e.cargo IN ('PROGRAMADOR','ANALISTA')
SQL	1. SELECT nro_emp, nom_emp, sueldo FROM emp WHERE (cargo = 'PROGRAMADOR' OR cargo = 'ANALISTA') 2. SELECT nro_emp, nom_emp, sueldo FROM emp WHERE cargo IN ('PROGRAMADOR','ANALISTA') 3. SELECT e.nro_emp, e.nom_emp, e.sueldo FROM emp e WHERE e.cargo IN ('PROGRAMADOR','ANALISTA')

SQL:

- A través del comparador IN se puede programar una comparación por igualdad contra una lista de valores.

5. Mostrar nom_depto de los departamentos que tengan empleados.

Comenzaremos a aplicar con este ejemplo una metodología que nos permitirá acercarnos a la solución a través de resolver cuestiones parciales. Los consejos que podemos dar aquí son los siguientes:

a. Trate de reescribir el requerimiento en términos un poco más formales y más cercanos al lenguaje que estamos utilizando para programar.
b. Intente dividir el problema en soluciones parciales, que Ud. podrá ir programando y verificando individualmente. Finalmente componga la solución total.
c. Descarte inicialmente datos que tendrá que mostrar recurriendo al acceso a otras tablas y reemplácelos por datos equivalentes almacenados en las tablas principales a utilizar en la consulta. Por ejemplo: nro_depto y nom_depto son equivalentes. Una vez encontrada la solución, podrá agregar los JOINs necesarios para obtener los datos solicitados.
d. Trate de imaginar tablas que harían que la programación de la consulta se simplifique. Por ejemplo, imaginando que la tabla tiene una columna con la información que Ud. necesita evaluar. Esto simplificará la programación de la consulta. Luego, vea como obtener esa columna, completando así la consulta.

NOTA: Estos son diferentes caminos que Ud. puede tomar si la consulta a programar es compleja. No todos ellos se pueden aplicar ni todos ellos producen beneficios en todos los casos, Ud. puede intentar con alguno de ellos o con algún otro camino que se le ocurra más conveniente para el caso.
De todas maneras, no puede olvidar nunca que la única forma de programar una consulta correcta es conocer a fondo el modelo de datos con el cual está trabajando y la información contenida en la base de datos a la cual Ud. debe acceder para obtener los datos solicitados.

Luego de esta introducción, analicemos el requerimiento:

- ¿Cuáles son los departamentos que tienen empleados? Son aquellos que están asociados a empleados en la tabla emp.
- Olvidémonos por ahora del nombre del departamento y concentrémonos en el número, ya que éste está almacenado en la tabla emp.
- Expresando la consulta de otra manera: "Mostrar el nro_depto de los empleados".

    En AR: emp [nro_depto]

- ¿Qué devuelve esta expresión del AR? Recuerde que el AR es cerrado.

  Esta expresión devuelve los DISTINTOS números de departamentos que tienen empleados.

  Ahora solo quedará obtener el nombre. ¿Cómo?:

  En AR: ((emp [nro_depto]) JOIN deptos) [nom_depto]

Veamos entonces las soluciones:

HERRAMIENTA	SOLUCIÓN
AR	1 ((emp [nro_depto]) JOIN deptos) [nom_depto]
CR	1. d.nom_depto WHERE EXISTS e (d.nro_depto = e.nro_depto)
SQL	1. SELECT DISTINCT d.nom_depto FROM emp e, deptos d WHERE e.nro_depto = d.nro_depto 2. SELECT DISTINCT d.nom_depto FROM emp e JOIN deptos d ON e.nro_depto = d.nro_depto 3. SELECT d.nom_depto

	FROM deptos d WHERE d.nro_depto IN (SELECT e.nro_depto FROM emp e) 4. SELECT d.nom_depto FROM deptos d WHERE EXISTS (SELECT * FROM emp e WHERE d.nro_depto = e.nro_depto)

AR:

- Notar que la proyección elimina automáticamente las tuplas duplicadas, con lo cual se cumple una de las propiedades de las relaciones y permite obtener entonces los diferentes números de departamentos con empleados.

CR:

- Aquí se utiliza un operador de condición muy usado en CR y en SQL. La expresión del cálculo relacional se puede leer de la siguiente manera:

  "Mostrar el nombre del departamento obtenido de las tuplas de la relación DEPTOS para las cuales EXISTE alguna tupla en la relación EMP que tenga el mismo nro. de departamento"

SQL:

- Las dos primeras variantes están basadas en el AR, la tercera es una variante propia de SQL y la cuarta está basada en el CR.

- Debemos imaginar que las operaciones asociadas a las cláusulas de la instrucción SELECT se ejecutan en el siguiente orden:

  1. FROM
  2. WHERE
  3. SELECT

- Analicemos las variantes:

Variante 1:

- La cláusula FROM produce un producto cartesiano entre las tablas indicadas en dicha cláusula asociando cada fila de la tabla emp con cada fila de la tabla deptos.

- Luego, la cláusula WHERE selecciona las filas de ese producto cartesiano para las cuales se cumple que el nro. de departamento del empleado es igual al nro. de departamento de la fila del departamento asociada.

- Finalmente, la cláusula SELECT selecciona las columnas a mostrar.

Es decir, que la variante 1 en SQL se podría definir como una proyección de una restricción de un producto cartesiano.

Vale la pena aclarar las diferencias del SQL con el AR:

a. En SQL, se puede hacer un producto cartesiano entre tablas con encabezamiento que tienen alguna columna con el mismo nombre. En este caso: nro_depto.

b. ¿Cómo se distingue una columna de otra en el resultado del producto cartesiano?

   A través de una calificación que se hace de la columna. Esta calificación se puede realizar agregando como prefijo el nombre de la tabla a la cual pertenece la columna o su alias:

   ```
   emp.nro_depto,
   e.nro_depto,
   deptos.nro_depto,
   d.nro_depto
   ```

   Entonces en esta consulta, sí hace falta utilizar o el nombre de la tabla o un alias para calificar a las columnas debido a que si no fuera así, el DBMS nos devolvería un error indicando que no puede saber si la columna a la que se está refiriendo es la que pertenece a una tabla o a la otra.
   NOTA: pruebe la misma consulta eliminando los alias de tablas en la cláusula WHERE.

c. ¿Por qué se debe agregar la cláusula DISTINCT?

   Otra diferencia con el AR, es que SQL no elimina las filas duplicadas, por lo tanto, si se desea eliminarlas se debe agregar antes de la lista de expresiones escalares la cláusula DISTINCT.

Variante 2:

- La cláusula FROM tiene una única expresión tabular (el resultado de un JOIN, en este caso un INNER JOIN).
  En SQL, la operación JOIN requiere la cláusula ON donde se indica la comparación por igualdad que se debe satisfacer. Este operador JOIN sería equivalente a una restricción de un producto cartesiano.

- La cláusula WHERE desaparece, ya que la restricción está planteada en la operación JOIN.

Variante 3:

- Se podría leer como: "Muestre los departamentos para los cuales su número está en la lista de números de departamentos asociados a empleados".

- Notar que tenemos dos sentencias SELECT. Una principal, desde donde se extrae la información solicitada y una secundaria, denominada sub-consulta, la cual (siempre) está encerrada entre paréntesis, desde donde se obtienen los números de departamentos a utilizar en la comparación.

- En este caso, la sub-consulta es independiente (su resultado no depende) de la consulta principal. Notar que, en el contexto de la sub-consulta, no se usa ninguna columna de la tabla utilizada en la consulta principal. También se le denomina no correlacionada. Esta sub-consulta se puede ejecutar independientemente de la consulta principal y con ella obtener todos los números de departamentos de la tabla emp. Esta sub-consulta devuelve una tabla de una columna y n filas.

- La ejecución de esta sub-consulta se realiza al comienzo. Con ella se extraen los números de departamento de los empleados y se la reemplaza

en la consulta principal para poder realizar la comparación. A partir de allí, la consulta principal se ejecuta normalmente.

- Es muy importante notar que los valores de la lista devuelta por la sub-consulta deben pertenecer al mismo dominio (tipo de dato) que el de la columna con la cual se compara, o un tipo de dato convertible al de la columna con la cual se compara.

Variante 4:

- Notar la similitud entre la expresión del CR y del SQL.

- En este caso la sub-consulta no es independiente y se denomina correlacionada. Esta dependencia se puede observar en la comparación indicada en la misma, donde se utiliza una columna de la tabla utilizada en la consulta principal.
  Se debe interpretar que, para cada fila resultado de la ejecución de la operación asociada a la cláusula FROM, se dispara la sub-consulta. Por lo tanto, el resultado en cada ejecución de esta sub-consulta dependerá de la fila que se esté analizando en la consulta principal.
  El resultado de la sub-consulta es un valor booleano relacional (tres valores): Verdadero, Falso o Nulo. Solo se seleccionará la fila en la consulta principal si el resultado de la sub-consulta es Verdadero.
  Es por esa razón que no se "selecciona" ninguna columna en particular, sino que se utiliza el "*", ya que lo único que se pretende saber es si devuelve filas o no y no el contenido de esas filas y así evitar una operación de proyección adicional.

6. Mostrar los nombres de empleados (nom_emp) de los empleados del departamento 'CONTABILIDAD'.

Rápidamente, Ud. podrá notar que la operación principal de esta consulta es la restricción, pero ¿cuál es la limitación que exige que previamente se realicen otras operaciones?.
Analicemos una expresión preliminar del AR:

(emp WHERE nom_depto = 'CONTABILIDAD') [nom_emp]

- ¿Cuál es el problema?

El problema es que nom_depto no es una columna de la tabla emp. ¡Pero ojalá lo fuera!, entonces:

- ¿Qué podemos hacer para que lo sea?

  Reunimos la tabla emp con la tabla deptos, y así:

  ((emp JOIN deptos) WHERE nom_depto = 'CONTABILIDAD') [nom_emp]

Veamos entonces las soluciones:

HERRAMIENTA	SOLUCIÓN
AR	1. ((emp JOIN deptos) WHERE nom_depto = 'CONTABILIDAD') [nom_emp] 2. ((emp JOIN (deptos WHERE nom_depto = 'CONTABILIDAD')) [nom_emp]
CR	1. e.nom_emp WHERE EXISTS d (d.nro_depto = e.nro_depto AND d.nom_depto = 'CONTABILIDAD')
SQL	1. SELECT e.nom_emp FROM emp e, deptos d WHERE e.nro_depto = d.nro_depto AND d.nom_depto = 'CONTABILIDAD' 2. SELECT e.nom_emp FROM emp e JOIN deptos d ON e.nro_depto = d.nro_depto WHERE d.nom_depto = 'CONTABILIDAD' 3. SELECT e.nom_emp FROM emp e JOIN deptos d ON e.nro_depto = d.nro_depto

	AND d.nom_depto = 'CONTABILIDAD' 4. SELECT e.nom_emp FROM emp e WHERE e.nro_depto IN (SELECT d.nro_depto FROM deptos d WHERE d.nom_depto = 'CONTABILIDAD') 5. SELECT e.nom_emp FROM emp e WHERE e.nro_depto = (SELECT d.nro_depto FROM deptos d WHERE d.nom_depto = 'CONTABILIDAD') 6. SELECT e.nom_emp FROM emp e WHERE EXISTS (SELECT * FROM deptos d WHERE e.nro_depto = d.nro_depto AND d.nom_depto = 'CONTABILIDAD')

AR:

- Tenemos dos variantes. La diferencia entre una y otra es que la segunda es más eficiente, ya que la operación JOIN se aplica a una tabla deptos con una única fila. Este tipo de variante se utiliza en los procesos de optimización que Ud. estudiará en la materia siguiente.

SQL:

- Las tres primeras variantes son expresiones equivalentes a las variantes del AR.

- Entre la segunda y la tercera variante la diferencia está en que la condición "d.nom_depto = 'CONTABILIDAD'", se ejecuta luego del JOIN (en la segunda variante) o antes del JOIN (en la tercera) como una restricción previa aplicada a la tabla deptos.

Es decir, la segunda variante de SQL es equivalente a la primera variante del AR y la tercera de SQL a la segunda del AR.
En este caso, ambas expresiones son equivalentes, pero sea cuidadoso, ya que en otros casos (donde es necesario utilizar OUTER JOINs) este orden puede ser relevante, por lo tanto, Ud. debe pensar cuáles son realmente las operaciones que se deben realizar y en qué orden para obtener la solución correcta.

- Entre la cuarta y la quinta variante, la diferencia está en el comparador. En uno se utiliza IN y en el otro =.

El comparador = solo se puede utilizar en el caso que la sub-consulta devuelva solo una fila.
De esta manera el DBMS considera a ese resultado como un escalar y permite su comparación con el valor de la columna.
Es importante entonces, asegurarse que la sub-consulta devuelve una única fila para poder utilizar ese comparador.

- La última variante, simplemente es una implementación directa de la expresión en CR en SQL.

7. Mostrar el nombre de los departamentos (nom_depto) que tengan empleados con cargo 'ANALISTA'.

La solución es similar a la del ejercicio anterior. Solamente agregamos este ejemplo como base para el ejemplo siguiente.

Veamos entonces las soluciones:

HERRAMIENTA	SOLUCIÓN
AR	1. ((emp JOIN deptos) WHERE cargo = 'ANALISTA') [nom_depto] 2. ((emp WHERE cargo = 'ANALISTA') JOIN deptos) [nom_depto]
CR	1. d.nom_depto WHERE EXISTS e (d.nro_depto = e.nro_depto AND e.cargo = 'ANALISTA')

SQL	1. SELECT DISTINCT d.nom_depto FROM emp e JOIN deptos d ON e.nro_depto = d.nro_depto WHERE e.cargo = 'ANALISTA' 2. SELECT d.nom_depto FROM deptos d WHERE d.nro_depto IN (SELECT e.nro_depto FROM emp e WHERE e.cargo = 'ANALISTA') 3. SELECT d.nom_depto FROM deptos d WHERE EXISTS (SELECT * FROM emp e WHERE e.nro_depto = d.nro_depto AND e.cargo = 'ANALISTA')

SQL:

- La primera variante es una implementación del AR.

  La segunda, es una variante SQL, tal como la explicábamos en ejemplos anteriores.

- La tercera variante, es una implementación del CR.

8. Mostrar el nombre de los departamentos (nom_depto) que NO tengan empleados con cargo 'ANALISTA'.

Este ejemplo tiene sus variantes y además debe ser interpretado correctamente.

Primer análisis:

Si no se analiza en detalle, lo primero que surge como solución es:

((emp JOIN deptos) WHERE cargo **<>** 'ANALISTA') [nom_depto]

Sin embargo, el requerimiento para esa respuesta es esa sería: "Mostrar el nombre de los departamentos (nom_depto) **que tengan** algún empleado con **cargo distinto a 'ANALISTA'**".

Pero, lo que se está pidiendo es: **"**Mostrar el nombre de los departamentos (nom_depto) **que NO tengan** empleados **con cargo igual a 'ANALISTA'"**

Para acercarnos a la expresión del AR: podemos transformar el requerimiento en uno similar: De todos los departamentos eliminar los que tienen empleados con cargo ANALISTA y mostrar su nombre.
Notar que las filas que se eliminan son aquellas extraídas en la consulta anterior. Es decir, será necesario utilizar la operación diferencia.

Veamos entonces las soluciones:

HERRAMIENTA	SOLUCIÓN
AR	1. (((deptos [nro_depto]) – (emp WHERE cargo = 'ANALISTA') [nro_depto]) JOIN deptos) [nom_depto]
CR	1. d.nom_depto WHERE **NOT** EXISTS e (d.nro_depto = e.nro_depto AND e.cargo = 'ANALISTA')
SQL	1. SELECT d.nom_depto FROM deptos d WHERE d.nro_depto **NOT** IN (SELECT e.nro_depto FROM emp e WHERE e.cargo = 'ANALISTA') 2. SELECT d.nom_depto FROM deptos d WHERE **NOT** EXISTS (SELECT * FROM emp e WHERE e.nro_depto =

	d.nro_depto AND e.cargo = 'ANALISTA')

SQL:

- Las dos variantes utilizan sub-consultas. Hay pocos DBMSs que admiten la operación diferencia del AR.
- La única diferencia entonces con el ejemplo anterior es el agregado del operador **NOT**.
- La fila de la consulta principal se seleccionará solo si no existe ninguna fila como resultado de la sub-consulta.

Segundo análisis:

Lo que se está mostrando son los departamentos que NO tienen empleados con cargo ANALISTA, incluyendo en la lista a los departamentos que NO tienen empleados.

Pero si se pide que muestre solo aquellos departamentos que SI tienen empleados y que NO tienen empleados con cargo ANALISTA.

¿Cómo serían entonces las expresiones? Así:

HERRAMIENTA	SOLUCIÓN
AR	1. (((emp [nro_depto]) – (emp WHERE cargo = 'ANALISTA') [nro_depto]) JOIN deptos) [nom_depto]
CR	1. d.nom_depto WHERE EXISTS e (d.nro_depto = e.nro_depto) AND **NOT** EXISTS e (d.nro_depto = e.nro_depto AND e.cargo = 'ANALISTA')
SQL	1. SELECT DISTINCT d.nom_depto FROM deptos d, emp e WHERE d.nro_depto = e.nro_depto AND d.nro_depto **NOT** IN (SELECT e.nro_depto FROM emp e

	WHERE e.cargo = 'ANALISTA') 2. SELECT d.nom_depto FROM deptos d, emp e WHERE d.nro_depto IN (SELECT e.nro_depto FROM emp e) AND d.nro_depto **NOT** IN (SELECT e.nro_depto FROM emp e WHERE e.cargo = 'ANALISTA') 3. SELECT d.nom_depto FROM deptos d WHERE EXISTS (SELECT * FROM emp e WHERE e.nro_depto = d.nro_depto) AND **NOT** EXISTS (SELECT * FROM emp e WHERE e.nro_depto = d.nro_depto AND e.cargo = 'ANALISTA')

- Seguramente Ud. ya podrá interpretar estas variantes, sin ninguna explicación extra.

9. Mostrar el número de empleado de aquellos empleados que trabajan en el mismo departamento que el empleado 'CASTRO'.

Podríamos imaginarnos una tabla emp, en la cual además del número de departamento donde trabaja el empleado, tengamos una columna nro_depto_castro, cuyo valor en todas las filas sería el número de departamento donde trabaja CASTRO. En ese caso, la consulta en AR sería simple:

(emp WHERE nro_depto = nro_depto_castro) [nro_emp]

Esta columna realmente no existe en la tabla emp, pero se la puede obtener con la siguiente expresión:

(emp WHERE nom_emp = 'CASTRO') [nro_depto AS nro_depto_castro]

Como esta expresión devuelve una tabla con una única fila y un único, podríamos utilizarla en un producto cartesiano con la tabla emp, lo que permitiría combinar esta fila con todas las de emp y obtener entonces la tabla que buscábamos:

((emp x ((emp WHERE nom_emp = 'CASTRO') [nro_depto AS nro_depto_castro])) WHERE nro_depto = nro_depto_castro)) [nro_emp]

El único inconveniente con el resultado es que también aparece CASTRO. Para eliminarlo del resultado restringimos:

(((emp x ((emp WHERE nom_emp = 'CASTRO') [nro_depto AS nro_depto_castro])) WHERE nro_depto = nro_depto_castro)) WHERE nom_emp <> 'CASTRO) [nro_emp]

Veamos entonces las soluciones:

HERRAMIENTA	SOLUCIÓN
AR	1. (((emp x ((emp WHERE nom_emp = 'CASTRO') [nro_depto AS nro_depto_castro])) WHERE nro_depto = nro_depto_castro)) WHERE nom_emp <> 'CASTRO) [nro_emp]
CR	1. e1.nro_emp WHERE e1.nom_emp <> 'CASTRO' AND EXISTS e2 (e1.nro_depto = e2.nro_depto AND e2.nom_emp = 'CASTRO')
SQL	1. SELECT e1.nro_emp FROM emp e1, emp e2 WHERE e1.nro_depto = e2.nro_depto AND e1.nom_emp <> 'CASTRO' AND e2.nom_emp = 'CASTRO' 2. SELECT e1.nro_emp FROM emp e1

	JOIN emp e2 ON e1.nro_depto = e2.nro_depto WHERE e1.nom_emp <> 'CASTRO' AND e2.nom_emp = 'CASTRO' 4. SELECT e1.nro_emp FROM emp e1 WHERE e1.nom_emp <> 'CASTRO' AND e1.nro_depto = (SELECT e2.nro_depto FROM emp e2 WHERE e2.nom_emp = 'CASTRO') 5. SELECT e1.nro_emp FROM emp e1 WHERE e1.nom_emp <> 'CASTRO' AND EXISTS (SELECT * FROM emp e2 WHERE e1.nro_depto = e2.nro_depto AND e2.nom_emp = 'CASTRO')

- Es importante destacar en estas soluciones la utilización de dos variables de tupla (en CR) y dos alias (en SQL), ya que, al utilizar dos veces la misma tabla, la única forma de salvar la ambigüedad es a través del uso de alias diferentes para cada una de ellas.

10. Mostrar el número de empleado (nro_emp) de los empleados más antiguos que el GERENTE.

Aplicando el mismo método que en el ejemplo anterior resolveremos éste, que nos servirá de base para el próximo ejemplo.

Veamos entonces las soluciones:

HERRAMIENTA	SOLUCIÓN
AR	1. ((emp x ((emp WHERE cargo = 'GERENTE') [fecha_ing AS fecha_ing_gerente])) WHERE fecha_ing <

	fecha_ing_gerente)) [nro_emp]
CR	1. e1.nro_emp WHERE EXISTS e2 (e1.fecha_ing < e2.fecha_ing AND e2.cargo = 'GERENTE')
SQL	1. SELECT e1.nro_emp FROM emp e1, emp e2 WHERE e1.fecha_ing < e2.fecha_ing AND e2.cargo = 'GERENTE' 2. SELECT e1.nro_emp FROM emp e1 WHERE e1.fecha_ing < (SELECT e2.fecha_ing FROM emp e2 WHERE e2.cargo = 'GERENTE') 3. SELECT e1.nro_emp FROM emp e1 WHERE e1.fecha_ing < (SELECT e2.fecha_ing FROM emp e2 WHERE e2.cargo = 'GERENTE') 4. SELECT e1.nro_emp FROM emp e1 WHERE EXISTS (SELECT * FROM emp e2 WHERE e1.fecha_ing < e2.fecha_ing AND e2.cargo = 'GERENTE')

11. Mostrar el número de empleado (nro_emp) de los empleados más antiguos.

Nuevamente aplicamos el mismo método.

Veamos entonces las soluciones:

HERRAMIENTA	SOLUCIÓN

AR	1. ((emp x (emp ADD MIN(fecha_ing) AS min_fecha_ing)) WHERE fecha_ing <= min_fecha_ing)) [nro_emp]
SQL	1. SELECT e.nro_emp FROM emp e WHERE e.fecha_ing <= (SELECT MIN(fecha_ing) FROM emp) 2. SELECT e.nro_emp FROM emp e WHERE e.fecha_ing = (SELECT TOP 1 fecha_ing FROM emp ORDER BY fecha_ing ASC) 3. SELECT e.nro_emp FROM emp e WHERE e.fecha_ing <= ALL (SELECT fecha_ing FROM emp)

CR:

- No se analizaron expresiones del CR para dar solución a este problema, por lo tanto, dejamos de lado esta herramienta.

SQL:

- Variante 1:

  Se usa aquí una operación o función de grupo, la cual actúa sobre todas las filas de la tabla emp (tomándolas como si pertenecieran a un único grupo) obteniendo la mínima fecha de ingreso entre todas ellas.

- Variante 2:

  Se usa aquí una cláusula SQL que permite seleccionar del resultado la primera fila (si fuera TOP 3 seleccionaría las primeras tres filas).
  Notar, que incorporamos una nueva cláusula: ORDER BY. La función de esta cláusula es ordenar el resultado final, por lo tanto, se aplica al resultado

obtenido luego de aplicar la cláusula SELECT.
En esta cláusula ORDER BY, se debe indicar cuales son las expresiones resultado por las cuales se ordena el mismo y para cada una de ellas, si se ordenan en forma ascendente (ASC) o descendente (DESC).
En este caso ordenamos las fechas en forma ascendente y seleccionamos la primera (la menor). Esto devuelve un único valor y es por eso que se puede utilizar el comparador =.

- Variante 3:

  Se usa aquí un operador que permite seleccionar los empleados cuya fecha de ingreso sea menor o igual a TODAS las fechas de ingreso de todos los empleados.

12. Mostrar el nombre de los departamentos (nom_depto) que tienen empleados en todos los cargos.

Suponemos acá que "todos los cargos" son todos aquellos que encontramos en la tabla emp.

Veamos entonces las soluciones:

HERR.	SOLUCIÓN
AR	1. ((emp [nro_depto, cargo] / emp [cargo]) JOIN deptos) [nom_depto]
CR	1. d.nom_depto WHERE FORALL c (EXISTS e (e.nro_depto = d.nro_depto AND e.cargo = c.cargo))
SQL	1. SELECT d.nom_depto FROM deptos d WHERE NOT EXISTS (SELECT * FROM emp c WHERE NOT EXISTS (SELECT * FROM emp e WHERE e.nro_depto = d.nro_depto

	AND e.cargo = c.cargo))

AR:

- Se utiliza la división como la operación central.

CR:

- Se puede expresar como: “mostrar el nombre del departamento donde, para todos los cargos existe algún empleado en ese departamento y en ese cargo”.

SQL:

- SQL no tiene el comparador FORALL, por lo tanto, debemos encontrar algo equivalente:

  FORALL (condición) es equivalente a NOT EXISTS (NOT condición)

  De esa manera, basándonos en la expresión del CR, se encuentra la solución en SQL, la cual se puede traducir como: “mostrar el nombre del departamento donde, no existe ningún cargo para el cual el departamento no tiene ningún empleado con ese cargo”.

13. Mostrar el número, nombre y sueldo anual (nro_emp, nom_emp, sueldo_anual) de los empleados con cargo VENDEDOR, donde sueldo_anual es el sueldo * 12.

Veamos las soluciones:

HERRAMIENTA	SOLUCIÓN
AR	1. (emp WHERE cargo = ‘VENDEDOR’) [nro_emp, nom_emp, sueldo * 12 AS sueldo_anual]
CR	1. e.nro_emp, e.nom_emp, e.sueldo*12 AS sueldo_anual WHERE e.cargo = ‘VENDEDOR’

SQL	1. SELECT e.nro_emp, e.nom_emp, e.sueldo * 12 sueldo_anual FROM emp e WHERE e.cargo = 'VENDEDOR'

AR:

- Se utiliza la operación restricción y la operación extensión.

SQL:

- Se pueden observar dos elementos nuevos en la cláusula SELECT:

  a. Una expresión escalar
  b. Un alias para el resultado de esa expresión escalar. Esto permite mostrar esta columna con ese título.

14. Mostrar la cantidad de empleados por departamento (nom_depto, cantidad).

A partir de aquí solo trabajaremos con soluciones en SQL, asumiendo que Ud. ya ha podido apreciar y utilizar el AR y el CR para iniciar el análisis del requerimiento y luego aplicar el resultado del mismo a la implementación de la solución en SQL.

Veamos entonces diferentes variantes:

```
1. SELECT d.nom_depto, COUNT(*) cantidad
     FROM emp e JOIN deptos d ON e.nro_depto = d.nro_depto
   GROUP BY d.nom_depto

2. SELECT d.nom_depto, COUNT(nro_emp) cantidad
     FROM emp e JOIN deptos d ON e.nro_depto = d.nro_depto
   GROUP BY d.nom_depto
```

3.
```
SELECT d.nom_depto, COUNT(DISTINCT nro_emp) cantidad
  FROM emp e JOIN deptos d ON e.nro_depto = d.nro_depto
 GROUP BY d.nom_depto
```

4.
```
SELECT d.nom_depto, c.cantidad
  FROM deptos d, (SELECT e.nro_depto, COUNT(*) cantidad
                    FROM emp e
                   GROUP BY e.nro_depto) c
 WHERE d.nro_depto = c.nro_depto
```

5.
```
SELECT d.nom_depto, c.cantidad
  FROM deptos d
       JOIN (SELECT e.nro_depto, COUNT(*) cantidad
               FROM emp e
              GROUP BY e.nro_depto) c
         ON d.nro_depto = c.nro_depto
```

- Variantes 1, 2 y 3:

  La operación asociada a la cláusula GROUP BY se ejecuta luego de que se ejecuten las operaciones asociadas a las cláusulas FROM y WHERE (si está programada).

  La cláusula GROUP BY tiene la función de agrupar las filas del resultado obtenido hasta el momento (en este caso el resultado del JOIN). Las filas en cada grupo tienen el mismo valor para las columnas o expresiones escalares incluidas en la cláusula GROUP BY (en este caso d.nom_depto).
  En este caso, luego del JOIN, se agrupan las filas correspondientes a cada departamento.
  La función de grupo COUNT se aplica a cada grupo.
  La cláusula GROUP BY no es obligatoria, pero si además de una función de grupo, existen expresiones escalares en el resultado (cláusula SELECT), entonces esas expresiones escalares o las columnas utilizadas en ellas deben estar incluidas en una cláusula GROUP BY. En este caso, en la cláusula SELECT, además de la función COUNT, se incluye la columna d.nom_depto, por lo tanto, esta columna debe estar incluida en una cláusula GROUP BY.

  Variantes de la función COUNT:

- COUNT(*): cuenta la cantidad de filas del grupo, sin analizar valores de ninguna columna.
- COUNT(expresión_escalar): cuenta la cantidad de filas en cada grupo cuyo valor para la expresión_escalar es NO NULO.
- COUNT(DISTINCT expresión_escalar): cuenta la cantidad valores DIFERENTES y NO NULOS de la expresión_escalar en cada grupo.

En este caso, las tres variantes dan el mismo resultado, ya que la columna nro_emp es obligatoria y con valores únicos.

- Variantes 4 y 5:

En estas variantes, aparecen sub-consultas en la cláusula FROM. Esta cláusula permite incluir no solo tablas, sino cualquier expresión tabular (cuyo resultado tenga la estructura de una tabla). Así podremos usar vistas, funciones tabulares (aún no desarrolladas aquí) y también sub-consultas.
Las características de las sub-consultas son las siguientes:

- Están encerradas entre paréntesis (como siempre).
- Se le asigna un alias para poder referenciar a sus columnas sin ambigüedad (en este caso el alias que se le otorgó es "c").
- Todas las expresiones escalares de la cláusula SELECT que no sean columnas deberán tener un alias, para poder referenciarlas (ver en este caso el alias "cantidad").

Ahora bien, ¿cuál sería la respuesta si lo que se quiere es la cantidad de empleados de todos los departamentos, incluso aquellos que no tienen empleados?

Veamos diferentes soluciones:

```
1. SELECT d.nom_depto, COUNT(*) cantidad
       FROM emp e JOIN deptos d ON e.nro_depto = d.nro_depto
     GROUP BY d.nom_depto
      UNION
```

```
    SELECT d.nom_depto, 0 cantidad
      FROM deptos d
    WHERE NOT EXISTS (SELECT *
                        FROM emp e
                       WHERE e.nro_depto = d.nro_depto)

2. SELECT d.nom_depto, COUNT(nro_emp) cantidad
     FROM deptos d LEFT OUTER JOIN emp e ON d.nro_depto = e.nro_depto
    GROUP BY d.nom_depto

3. SELECT d.nom_depto, (SELECT COUNT(*)
                          FROM emp e
                         WHERE d.nro_depto = e.nro_depto) cantidad
     FROM deptos d
```

- Variante 1:

  Utiliza la operación UNION basada en el AR, para agregar al resultado las filas correspondientes a los departamentos que no tienen empleados.
  Notar que los encabezamientos de ambos SELECTs unidos, tienen la misma cantidad de expresiones escalares y en orden los mismos tipos de datos o tipos de datos equivalentes, para permitir la operación UNION.
  Aquí aparece una nueva expresión escalar de resultado que es una constante. En este caso 0 (cero), que indica la cantidad de empleados de esos departamentos.

- Variante 2:

  Se reemplaza la UNION por un LEFT OUTER JOIN.
  Ud. debe notar que se ha invertido el orden de las tablas en el JOIN. Si no se hubiera invertido el orden, se tendría que haber usado un RIGHT OUTER JOIN.
  Ahora, COUNT(nro_emp) solo contará aquellas filas donde nro_emp es no nulo, o sea aquellas filas de departamentos con empleados asociados. Por lo tanto, para aquellos departamentos que no tengan empleados asociados, la cuenta dará 0 (cero).
  Si hubiésemos utilzado COUNT(*), estas filas hubieran sido contadas y la cantidad de empleados hubiera dado 1 para todos los departamentos que no tengan empleados.

- Variante 3:

  En esta variante se utiliza una sub-consulta como expresión escalar en la cláusula SELECT.
  Como habíamos mencionado antes, la cláusula SELECT permite incluir cualquier expresión escalar, incluyendo sub-consultas. El alias "cantidad" agregado permite darle un título a esa expresión.

15. Nombre de los departamentos (nom_depto) que tengan más de 3 empleados.

La operación principal es una restricción.
En este ejemplo veremos que hay otra cláusula que permite realizar esa operación.

Veamos entonces diferentes variantes:

```
1. SELECT d.nom_depto
     FROM emp e JOIN deptos d ON e.nro_depto = d.nro_depto
   GROUP BY d.nom_depto
   HAVING COUNT(*) > 3

2. SELECT d.nom_depto
     FROM deptos d
   WHERE (SELECT COUNT(*)
            FROM emp e
          WHERE d.nro_depto = e.nro_depto) > 3
```

- Variante 1:

  Aparece aquí una nueva cláusula: HAVING.
  Esta cláusula permite restringir (seleccionar) los grupos que se mostrarán como resultado.
  Tal como la cláusula WHERE, utiliza expresiones booleanas que permiten seleccionar los grupos para los cuales el resultado de dicha expresión es verdadero.

Habitualmente se utiliza asociada a la cláusula GROUP BY y se ejecuta luego de ésta.
Otras veces puede utilizarse sola, con lo cual la condición se aplica a un único grupo formado por todas las filas del resultado hasta el momento (luego de las operaciones correspondientes a las cláusulas FROM y WHERE).
Las condiciones planteadas en esta cláusula siempre utilizan funciones de grupo o expresiones o columnas incluidas en la cláusula GROUP BY.
No tiene sentido aquí aplicar condiciones a columnas no incluidas en la cláusula GROUP BY, porque estas condiciones se podrían aplicar antes en la cláusula WHERE, lo que permitiría restringir las filas que luego se agruparán, haciendo más eficiente la ejecución.

- Variante 2:

  En este caso se utilizó una sub-consulta en la cláusula WHERE, la cual es una sub-consulta correlacionada y escalar, cuyo resultado se compara con una constante (3).

16. Número de empleado y número de departamento de los empleados con menor sueldo que el menor sueldo del departamento nro. 10.

La operación principal nuevamente es una restricción.
Aquí se puede utilizar el mismo método que aplicamos en otros ejemplos, donde imaginamos una tabla emp con un atributo nuevo "min_sueldo_depto_10", y a partir de ahí, comenzar todo el análisis.

Veamos entonces una solución:

```
1. SELECT nro_emp, nro_depto
      FROM emp
   WHERE sueldo < (SELECT MIN(sueldo)
                     FROM emp
                   WHERE nro_depto = 10)
```

La sub-consulta es independiente (no correlacionada)
En ese caso, no es necesario utilizar alias para la tabla emp en la sub-consulta, ni en la consulta principal.

Con esto hemos finalizado una introducción al lenguaje SQL.
Esperamos que este material le haya permitido consolidar algunos conceptos vertidos en unidades anteriores, dominar las características y facilidades principales del lenguaje y avanzar rápidamente en el estudio, profundización y aplicación de estos y otros recursos que brinda, con el fin de sacar el máximo provecho en la satisfacción de requerimientos.