EAE Business School

Introducción a SQL

Máster en Business Intelligence e Innovación Tecnológica

Índice de contenidos

- La información dentro de una base de datos se estructura en tablas.

Id	Nombre	Edad
1	José	24
2	Celia	27
3	Fran	25

- Dentro de una tabla, cada instancia o elemento básico de datos es una tupla o registro

Id	Nombre	Edad
1	José	24
2	Celia	27
3	Fran	25

- Las tablas contienen columnas, que son todos los atributos de cada uno de los registros

Id	Nombre	Edad
1	José	24
2	Celia	27
3	Fran	25

- Algunos atributos son especiales e identifican a las tuplas de forma única: las claves primarias

Id	Nombre	Edad
1	José	24
2	Celia	27
3	Fran	25

- Podemos crear las tablas con las sentencias CREATE

```
CREATE TABLE PERSONAS(
    id INT PRIMARY KEY ,
    nombre VARCHAR( 255 ) ,
    edad INT
)
```

Id	Nombre	Edad

- Y las podemos eliminar con las sentencias DROP

```
DROP TABLE EMPLEADOS
```

```
CREATE TABLE PERSONAS (
```

sentencia — nombre tabla

nombre — tipo de datos — identificador clave

atributos:

```
    id   INT  PRIMARY KEY ,
    nombre VARCHAR( 255 ) ,
    edad INT
```

```
)
```

- Para añadir información a una tabla, utilizamos las sentencias INSERT

```
INSERT INTO PERSONAS( id, nombre, edad )
    VALUES ( 1, 'José', 24 ) ;
INSERT INTO PERSONAS( id, nombre, edad )
    VALUES ( 2, 'Celia', 27 ) ;
INSERT INTO PERSONAS( id, nombre, edad )
    VALUES ( 3, 'Fran', 25 ) ;
```

```
INSERT INTO PERSONAS( id, nombre, edad )
```

sentencia — nombre tabla — columnas tabla

```
VALUES ( 1, 'José', 24 ) ;
```

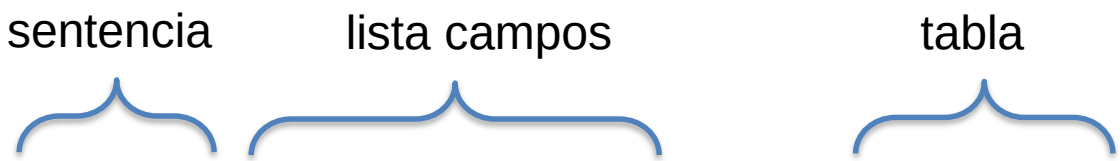
valores del registro

- Para obtener registros de una tabla, utilizamos la sentencia SELECT

```
SELECT *
FROM PERSONAS
```

```
SELECT id, nombre, edad
FROM PERSONAS
```

Id	Nombre	Edad
1	José	24
2	Celia	27
3	Fran	25




```
SELECT id, nombre, edad FROM PERSONAS
```

- Podemos filtrar los resultados de una query con la cláusula WHERE

```
SELECT id, nombre, edad
FROM PERSONAS
WHERE id =2
```

condiciones

Id	Nombre	Edad
2	Celia	27

- También podemos ordenar los resultados con la cláusula ORDER BY

```
SELECT id, nombre, edad
FROM PERSONAS
WHERE id =2
ORDER BY edad  ASC
```

Orden ascendiente

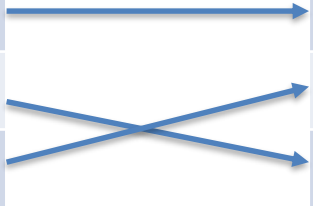
```
SELECT id, nombre, edad
FROM PERSONAS
WHERE id =2
ORDER BY edad DESC
```

Orden descendiente

- Las claves foráneas de una tabla vinculan sus registros con las claves primarias de otra tabla

Empleados

Id	Fecha alta	Departamento	IdPersona
1	01/10/2015	PROD	1
2	22/07/2012	FINA	3
3	15/02/2013	RRHH	2



Personas

Id	Nombre	Edad
1	José	24
2	Celia	27
3	Fran	25

- Podemos declarar las claves foráneas al crear la tabla…

```
CREATE TABLE EMPLEADOS(
      id INT PRIMARY KEY ,
      fecha_alta DATE,
      departamento VARCHAR( 4 ) ,
      id_persona INT,
      FOREIGN KEY ( id_persona )
            REFERENCES PERSONAS( id )
)
```

```
FOREIGN KEY ( id_empleado ) REFERENCES PERSONAS( id )
```

- O las podemos añadir a una tabla existente con una sentencia ALTER…

- La sentencia ALTER TABLE nos permite modificar tablas existentes. Podemos modificar su nombre, el de sus atributos, los tipos de los mismos, añadir nuevos o eliminar los existentes y en general cualquier elemento que se ha definido.

```
ALTER TABLE EMPLEADOS RENAME TRABAJADORES
```

```
ALTER TABLE PERSONAS ADD COLUMN fecha_nacimiento DATE
```

```
ALTER TABLE PERSONAS DROP fecha_nacimiento
```

```
ALTER TABLE TRABAJADORES DROP FOREIGN KEY id_empleado
```

Truco:

```
SHOW CREATE TABLE trabajadores
```

```
ALTER TABLE  TRABAJADORES  DROP FOREIGN KEY trabajadores_ibfk_1
```

Alter table

```
ALTER TABLE TRABAJADORES ADD FOREIGN KEY ( id_empleado )
REFERENCES PERSONAS( id )
```

```
ALTER TABLE  TRABAJADORES  ADD UNIQUE KEY ( id_empleado )
```

```
ALTER TABLE TRABAJADORES ADD COLUMN email VARCHAR( 200 )
```

```
ALTER TABLE TRABAJADORES MODIFY id BIGINT AUTO_INCREMENT
```

Los alter table aceptan montones de combinaciones!!

Tipos de datos

CHAR	String (0 - 255)
VARCHAR	String (0 - 255)
TINYTEXT	String (0 - 255)
TEXT	String (0 - 65535)
BLOB	String (0 - 65535)
MEDIUMTEXT	String (0 - 16777215)
MEDIUMBLOB	String (0 - 16777215)
LONGTEXT	String (0 - 4294967295)
LONGBLOB	String (0 - 4294967295)
TINYINT	Integer (-128 to 127)
SMALLINT	Integer (-32768 to 32767)
MEDIUMINT	Integer (-8388608 to 8388607)
INT	Integer (-2147483648 to 2147483647)
BIGINT	Integer (-9223372036854775808 to 922337-2036854775807)
FLOAT	Decimal (precise to 23 digits)
DOUBLE	Decimal (24 to 53 digits)
DECIMAL	"DOUBLE" stored as string
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS

Una vez hemos creado una tabla, insertado registros y los hemos seleccionado, es necesario que necesitemos modificar contenido de las pimeras. Para ello nos quedan por ver dos tipos de sentencias: UPDATE Y DELETE

UPDATE permite actualizar registros existentes. Podemos hacerlo de forma masiva:

```
UPDATE TRABAJADORES SET fecha_alta = current_date
```

O podemos delimitar los registros afectados, al igual que un SELECT, con la cláusula where.

```
UPDATE PERSONAS SET nombre = 'Pepe' WHERE id = 1
```

```
UPDATE PERSONAS SET nombre = 'Pepe' WHERE id = 1
```

sentencia — cambios — condiciones

DELETE trabaja de una forma similar, borrando todos los registros de una tabla o únicamente los que cumplan ciertas condiciones.

```
DELETE FROM TRABAJADORES
```

```
DELETE FROM PERSONAS WHERE id =3
```

```
DELETE FROM PERSONAS WHERE nombre = 'Celia'
```

Atributos y restricciones

Cuando definimos una columna (en un créate o alter table), podemos definir algunos atributos adicionales sobre ese campo, que aplicarán un comportamiento adicional del mismo en la tabla:

- NOT NULL: especifica que el campo no puede tener un valor nulo
- DEFAULT N: el campo toma por defecto el valor “N” cuando no lo informamos en el insert/update.
- AUTO_INCREMENT: por defecto asignará un valor auto incremental (valor anterior + 1).

Otros:

- PIMARY KEY: Define un campo como clave primaria.
- UNIQUE: Define un campo como clave única.

Las claves únicas son aquellos campos que deben tener valores únicos y que pueden actuar como claves secundarias de la tabla

```
CREATE TABLE NOMINAS(
      id INT AUTO_INCREMENT PRIMARY KEY ,
      fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      id_trabajador INT,
      salario DOUBLE NOT NULL ,
      FOREIGN KEY ( id_trabajador ) REFERENCES
            TRABAJADORES( id )
)
```

```
ALTER TABLE NOMINAS MODIFY
      salario DOUBLE NOT NULL DEFAULT 1000
```

De hecho, las claves (PK, FK, UK) son elementos considerados restricciones (constraints). Éstos son elementos declarativos pertenecientes a la tabla que restringen los valores que pueden tomar sus campos.
Las declaramos en la creación:

Bloque declaración de datos

Bloque declaración de restricciones

```
CREATE TABLE EMPLEADOS(
    id INT PRIMARY KEY ,
    fecha_alta DATE,
    departamento VARCHAR( 4 ) ,
    id_persona INT,
    FOREIGN KEY ( id_persona )
        REFERENCES PERSONAS( id )
)
```

O las declaramos con un ALTER:

```
ALTER TABLE TRABAJADORES ADD UNIQUE KEY ( email )
```

Además de claves, las restricciones pueden ser validaciones sobre valores concretos de los campos utilizando la cláusula CHECK.

Por ejemplo:

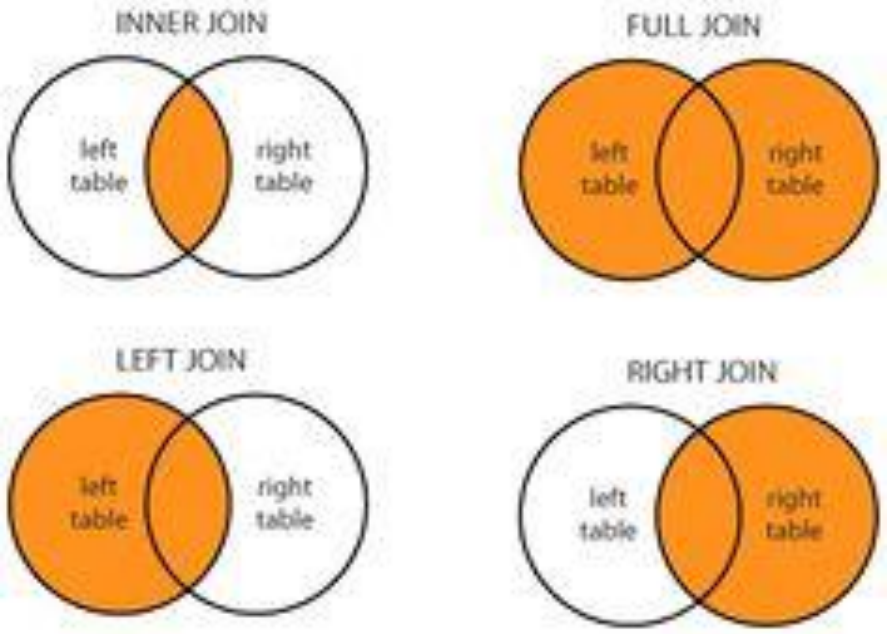
```
ALTER TABLE NOMINAS ADD CONSTRAINT chk_salario_minimo
    CHECK (salario >= 655.20)
```

NOTA: los checks no están al 100% suportados en MySQL

En algún momento posiblemente necesitaremos explotar información contenida en varias tablas. Para ello aparece el concepto de JOIN, que simplemente consisten en la unión de tablas en una consulta.

Existen dos tipos:

- INNER JOIN: la join “interna”, devuelve los valores que resultan de cruzar una tabla con otra.
- OUTER JOIN: en lugar devolver los valores comunes, hace una proyección de los resultados de una respecto a la otra, existan correspondencias en ésta o no.



INNER JOINS

Especificación de la join

```
SELECT * FROM PERSONAS INNER JOIN TRABAJADORES
ON PERSONAS.id = TRABAJADORES.id_persona
```

Campos sobre los que se aplica la Join

Además, podemos combinar cualquier JOIN con un where como en el resto de querys:

```
SELECT *
FROM PERSONAS INNER JOIN TRABAJADORES
ON PERSONAS.id = TRABAJADORES.id_persona
WHERE PERSONAS.id =1
```

OUTER JOINS

```
SELECT *
FROM PERSONAS LEFT JOIN TRABAJADORES
ON PERSONAS.id = TRABAJADORES.id_persona
```

id	nombre	edad
1	Pepe	24
2	Rosario	25

+

id	fecha_alta	departamento	id_persona
1	2016-11-17	RRHH	1

=

id	nombre	edad	id	fecha_alta	departamento	id_persona
1	Pepe	24	1	2016-11-17	RRHH	1
2	Rosario	25	NULL	NULL	NULL	NULL

OUTER JOINS

```
SELECT *
FROM PERSONAS RIGHT JOIN TRABAJADORES
ON PERSONAS.id = TRABAJADORES.id_persona
```

id	nombre	edad
1	Pepe	24
2	Rosario	25

+

id	fecha_alta	departamento	id_persona
1	2016-11-17	RRHH	1

=

id	nombre	edad	id	fecha_alta	departamento	id_persona
1	Pepe	24	1	2016-11-17	RRHH	1

Una forma de utilizar la INNER JOIN (y la más común) es la forma implícita, que modifica levemente su sintaxis:

```
SELECT *
FROM PERSONAS, TRABAJADORES
WHERE PERSONAS.id = TRABAJADORES.id_persona
```

```
SELECT *
FROM PERSONAS, TRABAJADORES
WHERE PERSONAS.id = TRABAJADORES.id_persona
AND PERSONAS.id = 1
```

Separamos todas las tablas que incluyamos mediante comas (“,”) y añadimos los campos del JOIN como condiciones del where, que adicionalmente puede contener otras no específicas del JOIN para filtrar separadas por AND.

Algo que no es específico de las JOINS pero se suele encontrar en ellas es el uso de ALIAS:

```
SELECT *
FROM PERSONAS PER, TRABAJADORES TRAB
WHERE PER.id = TRAB.id_persona
```

Con ellos podemos definir “motes” para las tablas, que utilizaremos para referirnos a ellas más tarde.

Como podemos ver, podemos incluir cuantas tablas sea necesario:

```
SELECT *
FROM PERSONAS, TRABAJADORES, NOMINAS
WHERE PERSONAS.id = TRABAJADORES.id_persona
    AND TRABAJADORES.id = NOMINAS.id_trabajador
```

Como dice el modelo relacional, el resultado de una consulta siempre es tratado una tabla. Por lo tanto, podemos utilizar consultas a modo de tablas:

```
SELECT id, departamento, (
SELECT salario FROM NOMINAS WHERE NOMINAS.id_trabajador = T.id
)
FROM TRABAJADORES T
```

```
SELECT id, departamento
FROM TRABAJADORES T
WHERE id IN (
    SELECT id
    FROM PERSONAS
    WHERE nombre LIKE 'Pepe'
)
```

Para poder comparar bien los valores de una o varias tablas, necesitamos conocer bien los operadores disponibles:

BETWEEN ... AND ...	Compara en un rango de valores
=	Operador “igual”
>	Operador “mayor”
>=	Operador “mayor o igual”
IN()	Compara en una lista de valores
IS	Compara un valor con un booleano
IS NOT	Negación de la comparación con booleano
IS NOT NULL	Compara que no tenga el valor NULL
IS NULL	Compara que sea NULL
<	Operador “menor”
<=	Operador “menor o igual”
LIKE	Comparador de cadenas de texto
NOT BETWEEN ... AND ...	Compara que un valor no esté en un rango
!=, <>	Operador “no igual” o distinto
NOT IN()	Chequea que el valor no esté en una lista
NOT LIKE	Negación del comparador LIKE

El operador LIKE tiene una casuística especial, puesto que no compara únicamente valores si no expresiones. Son muy frecuentes las que utilizan el "%", que a ojos de la consulta actua como un comodín.

```
SELECT *
FROM PERSONAS
WHERE nombre LIKE '%epe'
```

Acaba por…

Empieza por…

```
SELECT *
FROM PERSONAS
WHERE nombre LIKE '%epe'
```

```
SELECT *
FROM PERSONAS
WHERE nombre LIKE '%ep%'
```

Contiene…

Son pequeños procedimientos que añaden una lógica de operaciones para manipular o formatear datos. Su uso más común es aplicados sobre campos, por ejemplo en un where o en los resultados de una query.

```
SELECT AVG( salario ) FROM NOMINAS
```

```
SELECT *
FROM PERSONAS
WHERE LOWER( nombre ) LIKE 'pepe'
```

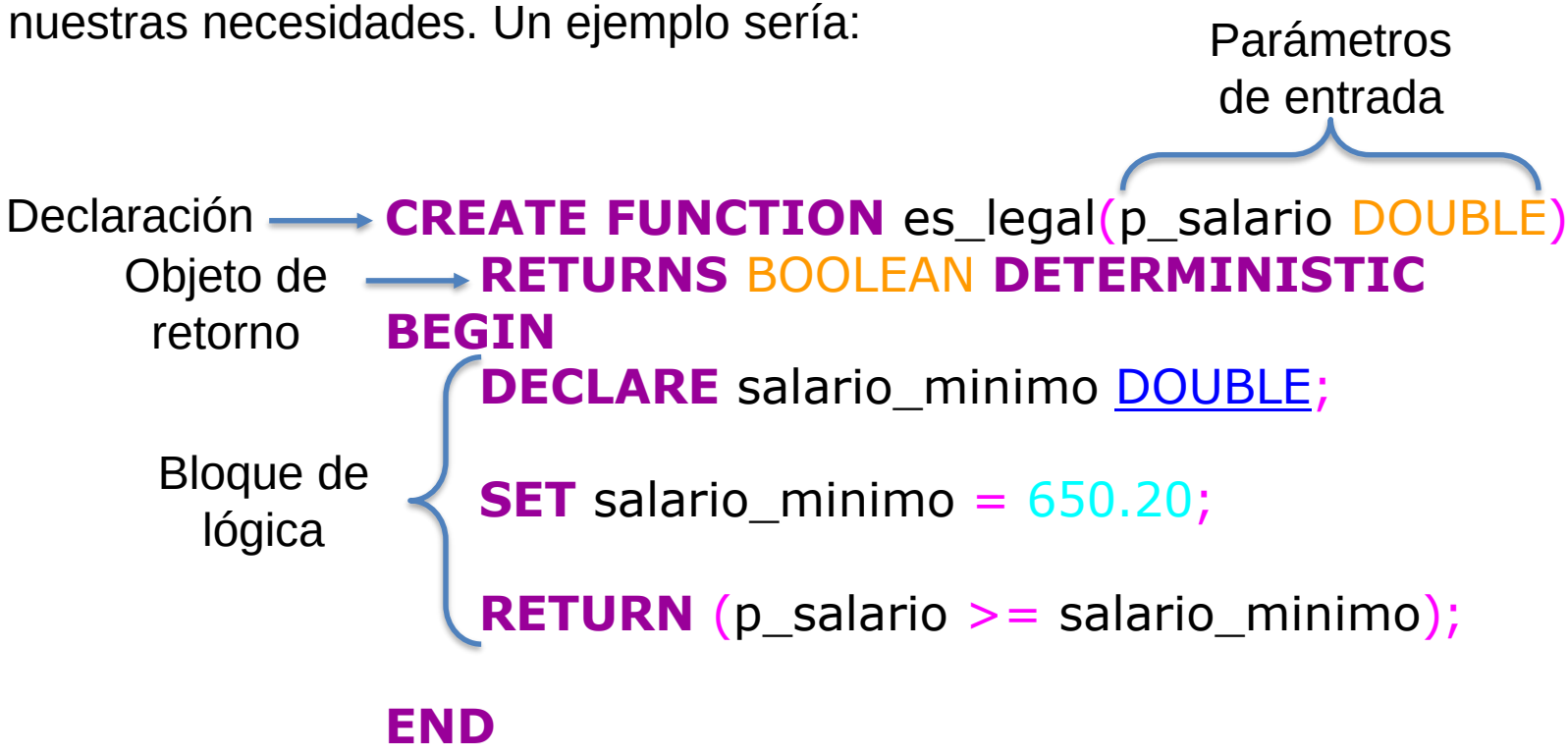
```
SELECT *
FROM TRABAJADORES
WHERE fecha_alta > STR_TO_DATE( '2001-12-21', '%Y-%m-%d' )
```

```
SELECT *
FROM TRABAJADORES
WHERE TRIM( departamento ) = 'RRHH'
```

Función	Utilidad	Ejemplo
LOWER o LCASE	convierte una cadena a minúsculas	SELECT LOWER('Hola'); ⇒ hola
UPPER o UCASE	convierte una cadena a mayúsculas	SELECT UPPER('Hola'); ⇒ HOLA
LEFT(cadena, longitud)	extrae varios caracteres del comienzo (la parte izquierda) de la cadena	SELECT LEFT('Hola',2); ⇒ Ho
RIGHT(cadena, longitud)	extrae varios caracteres del final (la parte derecha) de la cadena	SELECT RIGHT('Hola',2); ⇒ la
SUBSTR(cadena, posición, longitud)	extrae varios caracteres de cualquier posición de una cadena, tantos como se indique en "longitud"	SELECT SUBSTRING('Hola',2,3); ⇒ ola
CONCAT	une (concatena) varias cadenas para formar una nueva	SELECT CONCAT('Ho', 'la'); ⇒ Hola
LTRIM	devuelve la cadena sin los espacios en blanco que pudiera contener al principio (en su parte izquierda)	SELECT LTRIM(' Hola'); ⇒ Hola
RTRIM	devuelve la cadena sin los espacios en blanco que pudiera contener al final (en su parte derecha)	SELECT RTRIM('Hola '); ⇒ Hola
TRIM	devuelve la cadena sin los espacios en blanco que pudiera contener al principio ni al final	SELECT TRIM(' Hola '); ⇒ Hola
LENGTH	devuelve la longitud de la cadena en bytes	SELECT LENGTH(' Hola '); ⇒ 4
STR_TO_DATE	convierte un texto a un objeto de tipo tiempo (DATE, TIMESTAMP,…) a partir de un patrón	SELECT STR_TO_DATE('19-11-2016','%d-%m-%Y'); ⇒ Fecha

Función	Utilidad	Ejemplo
COUNT()	Devuelve la cantidad total de ítems	SELECT COUNT(*)
MAX	Devuelve el valor más alto	SELECT MAX(CAMPO)
MIN	Devuelve el valor más bajo	SELECT MIN (CAMPO)
AVG	Devuelve el valor medio	SELECT AVG(CAMPO)
DISTINCT	Devuelve todos los valores distintos de una lista de valores	SELECT DISTINCT(CAMPO);

MySQL permite la posibilidad de crear nuestras propias funciones adaptadas a nuestras necesidades. Un ejemplo sería:



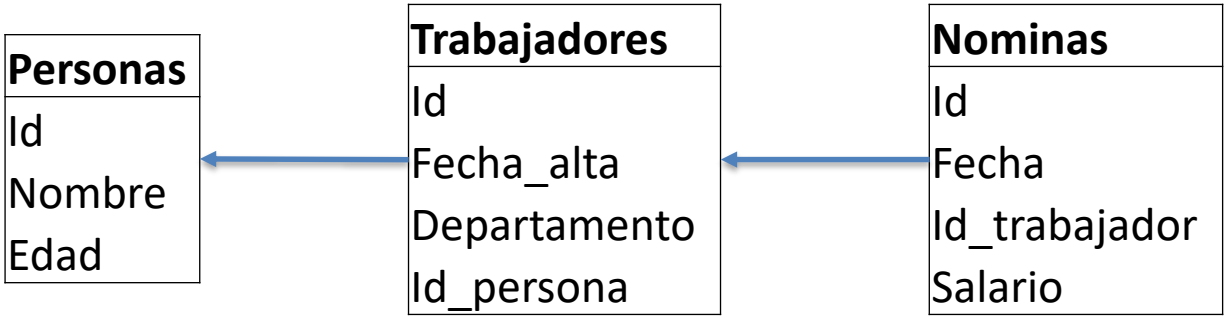
Podremos invocar nuestra función de una manera muy sencilla:

```
SELECT es_legal(salario)
FROM NOMINAS
```

```
UPDATE NOMINAS SET salario =550 WHERE id =2
```

```
SELECT es_legal(salario)
FROM NOMINAS
WHERE id =2
```

La BDD quedaría como…



Realizar inserts en tabla PERSONAS:

ID	Nombre	Edad
1	Fátima	45
2	Francisco	28
3	Carol	21
4	Javier	33
5	César	52
6	Marta	42
7	Alba	37

Realizar inserts en tabla TRABAJADORES:

Id	Fecha_alta	Departamento	Id_persona
1	23-02-2007	IT	2
2	03-09-2014	RRHH	3
3	18-12-2008	PROD	5
4	12-10-2012	FINA	6
5	08-04-2011	PROD	7

Realizar inserts en tabla NOMINAS:

Id	Fecha	Id_trabajador	Salario
1	30-09-2016	1	1200
2	30-09-2016	2	900
3	30-09-2016	3	1100
4	30-09-2016	4	1500
5	30-09-2016	5	1200
6	31-10-2016	1	1200
7	31-10-2016	2	900
8	31-10-2016	3	1100
9	31-10-2016	4	1500
10	31-10-2016	5	1200
11	HOY	1	1300

Realizar las siguientes consultas:

- Lista de personas mayores de 30 años
- Edad de la persona “Javier”
- Lista de aquellas personas cuyo nombre empieza por “C”
- Lista de personas que no son empleados
- Devuelve lista de departamentos sin repeticiones
- Nombre de los empleados del departamento ‘PROD’
- Nombre y edad de la persona con más antigüedad
- Nombre y departamento de la persona con el mayor salario
- Distintos salarios pagados a Francisco y Marta
- Nombre de la persona que más salarios ha cobrado