Afi Escuela

Bases de datos relacionales

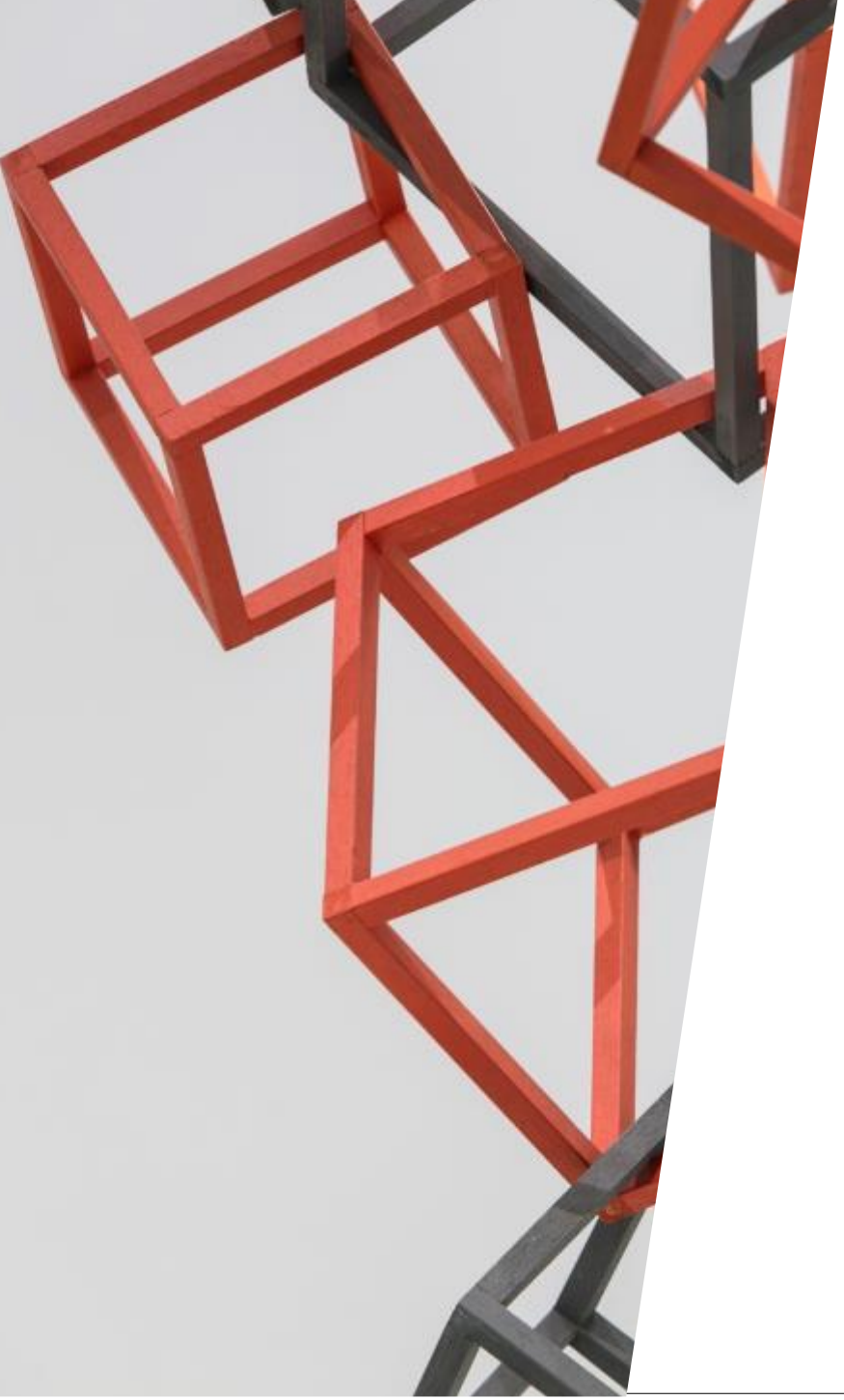
Máster en Data Science y Big Data en Finanzas (MDS_F)
Máster en Data Science y Big Data (MDS)

José Manuel Rodríguez Madrid
jmrodriguez@afi.es

Afi Escuela

Índice

Introducción



¿Qué es una base de datos?

- Almacén de información organizada/estructurada.

- Normalmente los datos de este almacén tienen algún tipo de **relación o vínculo** entre ellos.

- El **gestor de base de datos** (DBMS, *Data Base Management System*) es un software que permite almacenar la información de manera estructurada y acceder a ella de forma rápida. Este software permite realizar 4 operaciones básicas:
  - **Definición** de la estructura y organización de los datos.
  - **Actualización** (inserción, modificación y borrado) de los datos.
  - **Recuperación** (query) de los datos.
  - **Administración** del sistema (permisos, monitorización del rendimiento, …)

Introducción

¿Qué es una base de datos?

- El modelo de una base de datos determina la estructura interna de la misma. Es decir, la manera en que los datos serán almacenados. Algunos ejemplos:
  - Modelo relacional (basado en tablas).
  - Modelo dimensional (evolución del modelo relacional).
  - Modelo orientado a objetos.
  - Modelo de grafos.

- Para poder acceder a la información de una base de datos emplearemos un lenguaje de consultas (SQL es el más extendido).

- Algunos de los datos que se pueden almacenar/encontrar en una base de datos están protegidos por la Ley Orgánica de Protección de Datos de Carácter Personal (LOPD). **Información sensible y de carácter personal**.

- Existen diferentes tipos de bases de datos. Algunos ejemplos:
  - Bases de datos **relacionales** (sistemas OLTP)
  - Bases de datos **multidimensionales** (sistemas OLAP)
  - Bases de datos **NoSQL**

Introducción

¿Qué es una base de datos relacional? (I)

- Es un tipo de base de datos que cumple con el **modelo relacional** (basado en la lógica de primer orden/lógica de predicados/cálculo de predicados).

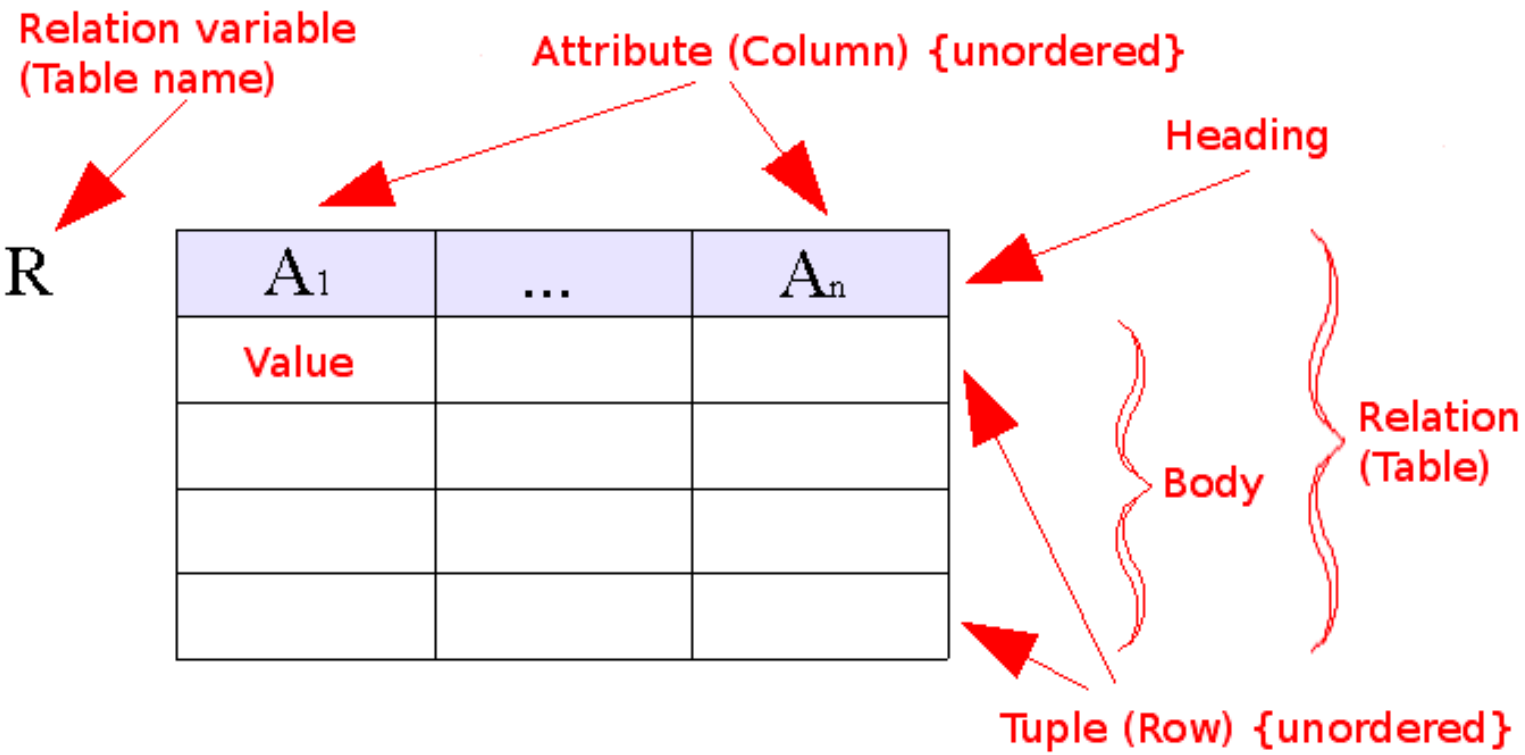
- Es el modelo de base de datos más extendido.

- Características básicas del modelo relacional:
  - Los datos son representados como **registros/tuplas (filas)** y agrupados dentro de **relaciones (tablas)**.
  - La tabla tiene un conjunto de **campos/atributos (columnas)**.
  - Las conexiones entre tablas se establecen entre las **claves primarias y claves foráneas**.
  - A través de las claves primarias y foráneas garantizamos la integridad de información.

- Las bases del modelo relacional fueron postuladas por Edgar Frank Codd en el año 1970 cuando trabajaba en IBM.

Introducción

¿Qué es una base de datos relacional? (II)



Introducción

Elementos de una base de datos relacional (I)

- **Relaciones:** las relaciones permiten almacenar y consultar los datos. Existen dos tipos:
  - **Relación base o tabla:** Almacena datos.
  - **Relación derivada o consulta o vista:** No almacena datos. Se calculan al aplicar operaciones relacionales. Expresan información de varias tablas como si fuera una única tabla.

- **Restricciones:** son limitaciones sobre los campos de las tablas de base de datos. No son fundamentales pero permiten mejorar la organización de los datos.

- **Dominios:** tipos de datos que serán almacenados.

- **Claves:**
  - **Clave primaria:** clave única que permite identificar un registro de la tabla.
  - **Clave foránea:** referencia a una clave de otra tabla. No tienen porque ser claves únicas en la tabla donde están, pero sí donde están referenciadas. **Integridad de la información.**
  - **Clave índice:** permiten un acceso más rápido a los datos. Se pueden crear como combinación de los campos de una tabla.

Introducción

Elementos de una base de datos relacional (II)

- **Procedimientos almacenados:** código ejecutable que se guarda en la base de datos. Cada motor de base de datos tiene su propio lenguaje de programación (por ejemplo: SQL Server tiene Transact-SQL, Oracle tiene PL/SQL…).

- **Estructura:**
  - **Esquema:** define la estructura de base de datos (tablas, columna, tipos de datos, claves primarias, relaciones, índices…)
  - **Datos:** registros de información. Filas de una tabla (tuplas). El número de filas de una tabla es denominado cardinalidad. El número de columnas se denomina aridad o grado.

Introducción

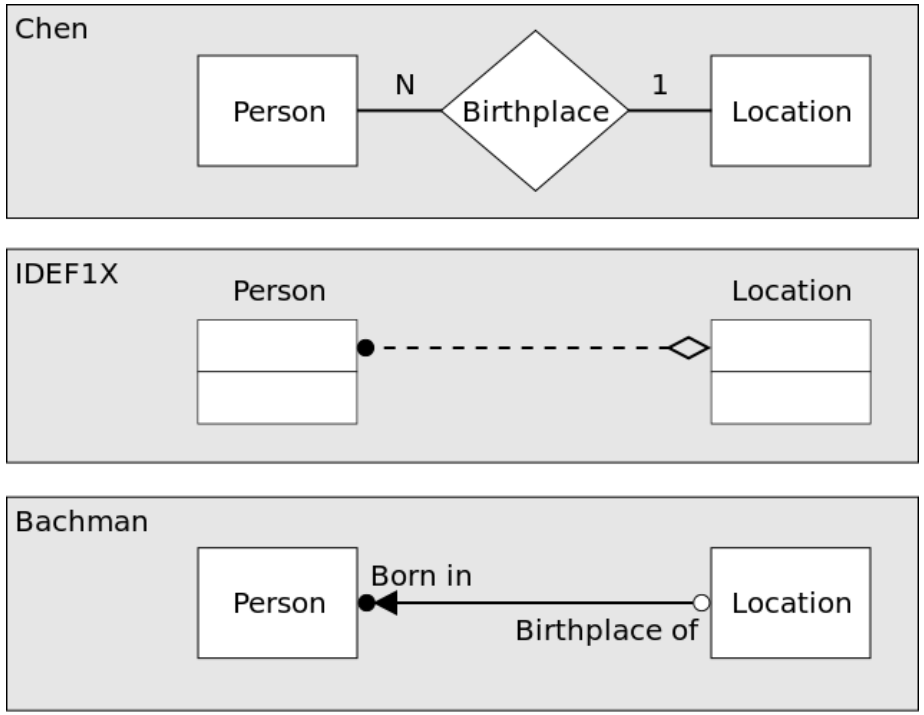
Modelo entidad-relación (I)

- Herramienta para el **modelado de datos**, permite representar las entidades relevantes de un sistema de información, sus relaciones y propiedades (atributos).

- Elementos de un modelo entidad-relación:
  - **Entidad:** representa una cosa u objeto del mundo real. Pueden ser objetos con existencia física (por ejemplo: persona…) o con existencia conceptual (por ejemplo: cuenta bancaria…).
  - **Atributo:** son las características que describen o definen las entidades/relaciones.
  - **Relación:** establece dependencias entre entidades. La **cardinalidad** de una relación indica el número de entidades con las que puede estar relacionada otra entidad.
    - **Uno a uno (1:1):** un registro de una entidad A se relaciona con un registro de una entidad B.
    - **Uno a varios (1:n):** un registro de una entidad A se relaciona con cero o varios registros de una entidad B.
    - **Varios a uno (n:1):** un registro de una entidad B se puede relacionar con cero o varios registros de una entidad A.
    - **Varios a varios (n:m):** una entidad A se relaciona con varios registros de la entidad B y viceversa.

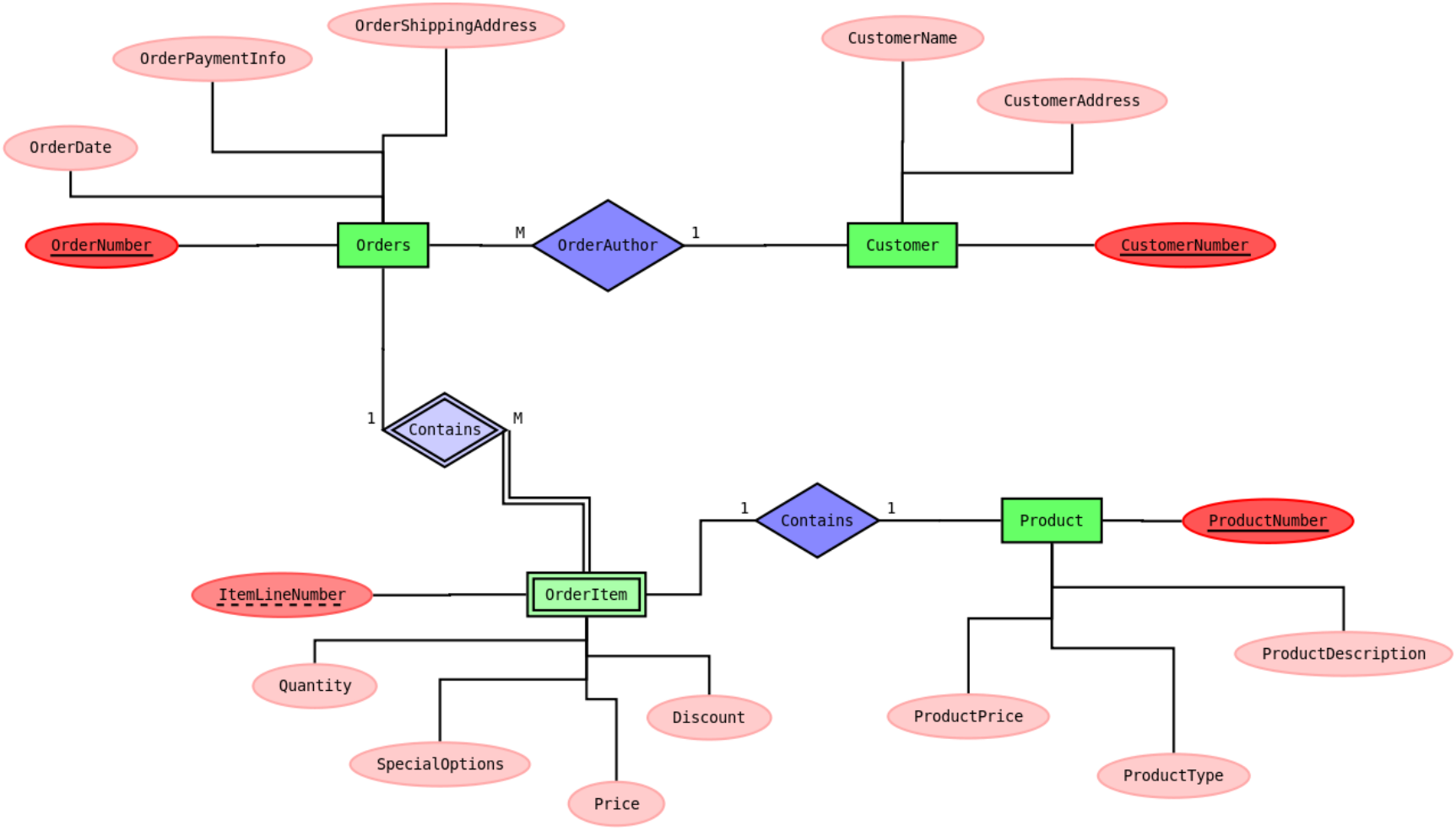
Introducción

Modelo entidad-relación (II)



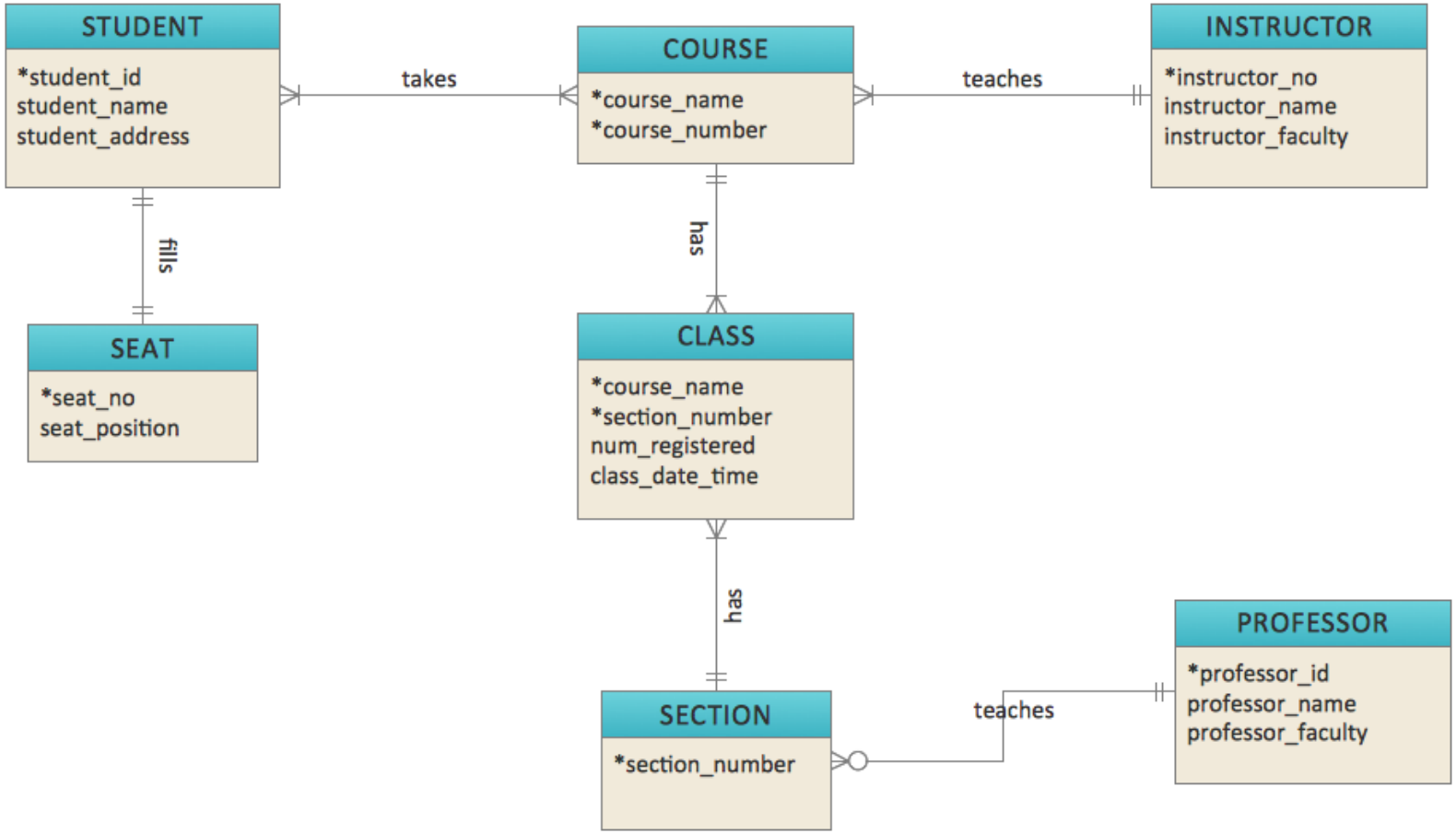
Introducción

Diagrama entidad-relación (I)



Introducción

Diagrama entidad-relación (II)



Introducción

Del modelo entidad relación al modelo de tablas

- **Entidades:** las entidades se convierten en tablas. Los atributos de la entidad en columnas de la tabla.

- **Relaciones:** en general las relaciones se transforman también en tablas. En el caso de que la relación contenga atributos siempre genera tablas.
  - **Relaciones 1 a 1:** cada entidad de la relación se convierte en una tabla. La relación aparece como clave primaria en una de las tablas y como clave foránea en la otra tabla.
  - **Relaciones 1 a N:** cada entidad de la relación se convierte en una tabla. Se pasa la clave primaria de la entidad con cardinalidad 1 a la tabla de la otra entidad como clave foránea.
  - **Relaciones N a M:** tanto las entidades como la relación se convierten en tablas. Las claves primarias de las entidades aparecen como clave primaria en la relación.

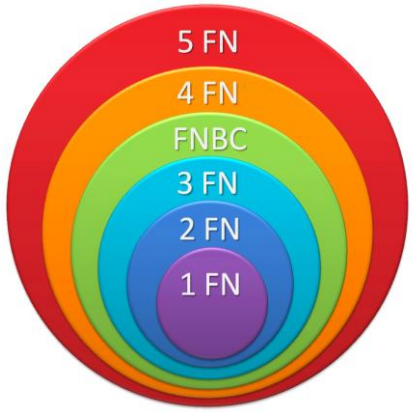
Introducción

Normalización de bases de datos

- A través de una serie de reglas o normas aplicadas en el momento del diseño de una base de datos relacional podremos:
  - Evitar redundancia de datos.
  - Disminuir problemas derivados de la actualización de datos en tablas.
  - Proteger la integridad de datos.

- Estas reglas se denominan Formas Normales. Las tres primeras formas normales son suficientes para cumplir las necesidades de la mayoría de base de datos.

- En la práctica es **complicado** hacer cumplir todas las reglas de las Formas Normales.



Introducción

Primera forma normal: 1FN (I)

- Una tabla está en primera forma (**1FN**) normal sí y solo sí es isomorfa a alguna relación, es decir, si se cumplen las siguientes condiciones:
  - No existe orden en las filas (de arriba a abajo). El orden no es significativo en la vista.
  - No existe orden en las columnas (de izquierda a derecha).
  - No hay filas duplicadas. Tiene una clave primaria.
  - Cada intersección de fila y columna contiene exactamente **un valor** del dominio aplicable. Los campos son atómicos. No puede contener columnas con valores nulos o más de un valor.

Introducción

Primera forma normal: 1FN (II)

Customer Id	First Name	Surname	Telephone Number
123	Rachel	Ingram	555-861-2025
456	James	Wright	555-403-1659
789	Cesar	Dure	555-808-9633

- La tabla anterior guarda información de clientes.
- ¿Qué pasa si queremos guardar más de un número de teléfono?

Introducción

Primera forma normal: 1FN (III)

Customer Id	First Name	Surname	Telephone Number
123	Rachel	Ingram	555-861-2025
456	James	Wright	555-403-1659 555-776-4100
789	Cesar	Dure	555-808-9633

- No cumple la 1FN porque hay más de un valor de dominio en la columna "`Telephone Number`".

Introducción

Primera forma normal: 1FN (IV)

Customer Id	First Name	Surname	Telephone Number 1	Telephone Number 2	Telephone Number 3
123	Rachel	Ingram	555-861-2025	NULL	NULL
456	James	Wright	555-403-1659	555-776-4100	NULL
789	Cesar	Dure	555-808-9633	NULL	NULL

- No cumple la 1FN porque las columnas "`Telephone Number 2`" y "`Telephone Number 3`" contienen valores nulos.

Introducción

Primera forma normal: 1FN (V)

Customer Id	First Name	Surname	Telephone Number
123	Rachel	Ingram	555-861-2025
456	James	Wright	555-403-1659, 555-776-4100
789	Cesar	Dure	555-808-9633

- No cumple la 1FN porque la columna "`Telephone Number`" puede ahora almacenar o un número de teléfono o una lista de los mismos. No existe un tipo/dominio definido para dicha columna (la columna no es atómica).

Introducción

Primera forma normal: 1FN (VI)

Customer Id	First Name	Surname
123	Rachel	Ingram
456	James	Wright
789	Cesar	Dure

Customer Id	Telephone Number
123	555-861-2025
456	555-403-1659
456	555-776-4100
789	555-808-9633

- **Solución:** emplear dos tablas diferentes: una para guardar los nombres de los clientes, la otra para guardar los teléfonos.

Introducción

Primera forma normal: 2FN (I)

- Una tabla está en segunda forma normal (**2FN**) si está en 1FN y además, sí y solo sí, dada una clave primaria, cualquier atributo que no constituya la clave primaria, depende íntegramente de la clave primaria al completo.

- Cuando una tabla no cumple la 2FN, puede sufrir problemas a la hora de mantener la información actualizada de manera correcta.

- Existen casos para los que aún cumpliendo la 2FN, es posible que se den problemas de actualización. Para estos casos existe la 3FN.

Introducción

Primera forma normal: 2FN (II)

Employee	Skill	Current Work Location
Brown	Light Cleaning	73 Industrial Way
Brown	Typing	73 Industrial Way
Harrison	Light Cleaning	73 Industrial Way
Jones	Shorthland	114 Main Street
Jones	Typing	114 Main Street
Jones	Whittling	114 Main Street

- La tabla anterior guarda información de empleados, sus habilidades y lugar de trabajo.
- La clave primaria de la tabla está conformada por las columnas "`Employee`" y "`Skill`".
- Este diseño de tabla no cumple la 2FN, ya que la columna "`Current Work Location`" **no depende de la clave primaria en su totalidad**, únicamente de "`Employee`".

Introducción

Primera forma normal: 2FN (III)

Employee	Skill
Brown	Light Cleaning
Brown	Typing
Harrison	Light Cleaning
Jones	Shorthland
Jones	Typing
Jones	Whittling

Employee	Current Work Location
Brown	73 Industrial Way
Harrison	73 Industrial Way
Jones	114 Main Street

- **Solución:** emplear dos tablas diferentes: una para guardar las habilidades, la otra para guardar las direcciones de trabajo.

Introducción

Primera forma normal: 3FN (I)

- Una tabla está en tercera forma normal (**3FN**) si está en 2FN y además, sí y solo sí, ningún atributo no primario (atributo no perteneciente a la clave primaria) de la tabla es dependiente transitivamente (a través de otro) de una clave primaria. Es decir, un tributo no primario depende de otro atributo no primario.

- La 3FN, soluciona los problemas de actualización no contemplados por la 2FN.

Introducción

Primera forma normal: 3FN (II)

Tournament	Year	Winner	Winner Date of Birth
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleavelanda Open	1999	Bob Alvertson	28 September 1968
Des Moines Masters	1999	Al Fredickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

- La tabla anterior guarda información de torneos, así como de sus vencedores.
- La clave primaria de la tabla está conformada por las columnas "`Tournament`" y "`Year`".
- Este diseño de tabla no cumple la 3FN, ya que la columna "`Winner Date of Birth`" **depende transitivamente de la clave primaria**, a través de la columna "`Winner`".

Introducción

Primera forma normal: 3FN (III)

Tournament	Year	Winner
Indiana Invitational	1998	Al Fredrickson
Cleavelanda Open	1999	Bob Alvertson
Des Moines Masters	1999	Al Fredickson
Indiana Invitational	1999	Chip Masterson

Winner	Winner Date of Birth
Al Fredrickson	21 July 1975
Bob Alvertson	28 September 1968
Al Fredickson	21 July 1975
Chip Masterson	14 March 1977

- **Solución:** emplear dos tablas diferentes: una para guardar los torneos, la otra para guardar las fechas de nacimiento de los ganadores.

Introducción

Sistema transaccional

- Un sistema transaccional, o sistema orientado a transacciones, es aquel que trabaja con operaciones indivisibles, llamadas transacciones.

- Una transacción es una operación (o conjunto de operaciones) que se ejecutan garantizando el cumplimiento de las propiedades ACID:
  - **Atomicidad:** cada transacción se ejecuta completamente y de manera correcta o no tiene impacto en el sistema. Si una parte de la transacción falla en un momento de su ejecución, entones dicha transacción no produce cambios.
  - **Consistencia:** cualquier transacción lleva el sistema de un estado válido a otro estado válido.
  - **Aislamiento (*Isolation*):** varias ejecuciones concurrentes de transacciones sobre el sistema se procesan una tras otra, evitando bloqueos.
  - **Durabilidad:** una vez finalizada la operación, esta persiste en el sistema aunque se produzcan errores o fallos.

- Los sistemas relacionales cumplen las características de un sistema transaccional.

Introducción

SQL (*Structured Query Language*) (I)

- Lenguaje de acceso a bases de datos relacionales.
- Basado en el álgebra relacional y el cálculo relacional.
- Es un estándar ANSI (American National Standards Institute) desde 1986. Desde su primera versión ha sufrido varias revisiones a lo largo de los años.
- No es *case-sensitive*, se puede escribir en mayúscula o minúsculas. Por claridad, es importante escribir las sentencias manteniendo un estilo.
- **Lenguaje de definición de datos (DDL, *Data Definitión Language*):** definición y modificación de objetos de base de datos (tablas, índices, esquemas, etc.).
- **Lenguaje de manipulación de datos (DML, *Data Manipulation Language*):** consulta y manipulación de los datos.

Introducción

SQL (*Structured Query Language*) (II)

- El **gestor de base de datos relacional (RDBMS, *Relational Data Base Management System*)** permite almacenar la información de una base de datos relacional.

- Cada base de datos comercial cumple con el estándar de SQL, aunque existen ligeras diferencias/peculiaridades de una a otra.

Introducción

SQL (*Structured Query Language*) (III)

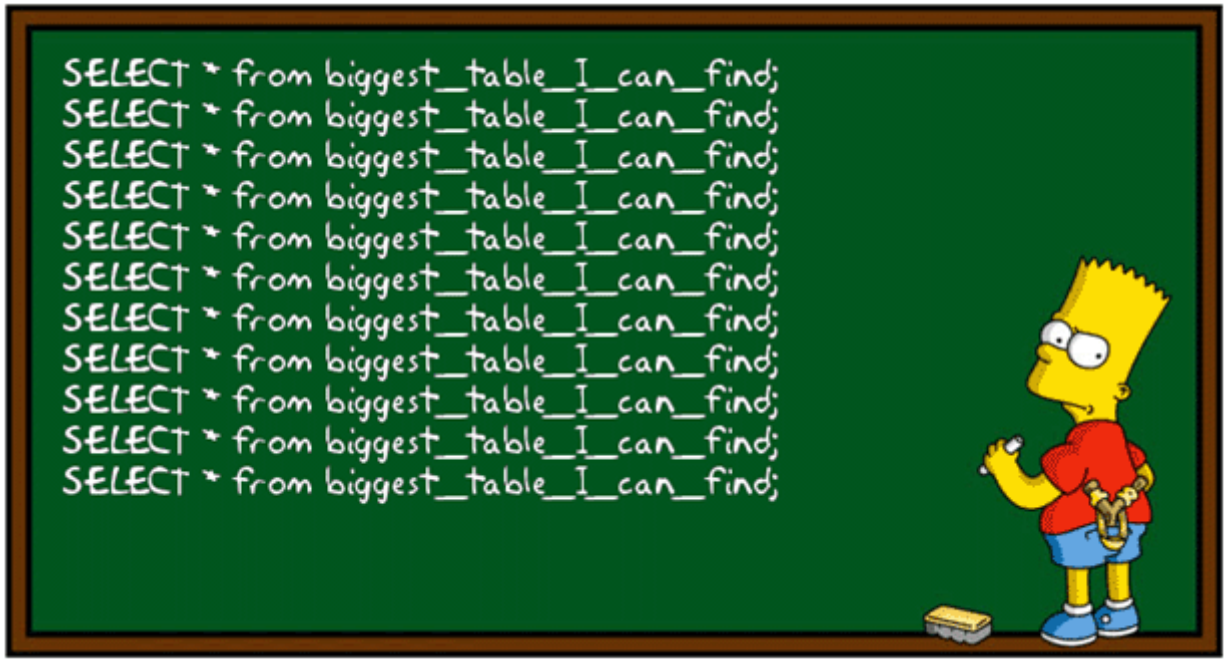
- Crear nuevas bases de datos.
- Crear nuevas tablas en una base de datos.
- Establecer permisos en las tablas, procedimientos y puntos de vista.
- Insertar registros en una base de datos.
- Actualizar registros en una base de datos.
- Eliminar registros de una base de datos.
- Ejecutar consultas en una base de datos.
- Crear procedimientos almacenados en una base de datos.
- Crear vistas en una base de datos.

Introducción

SQL... ¿Por qué?

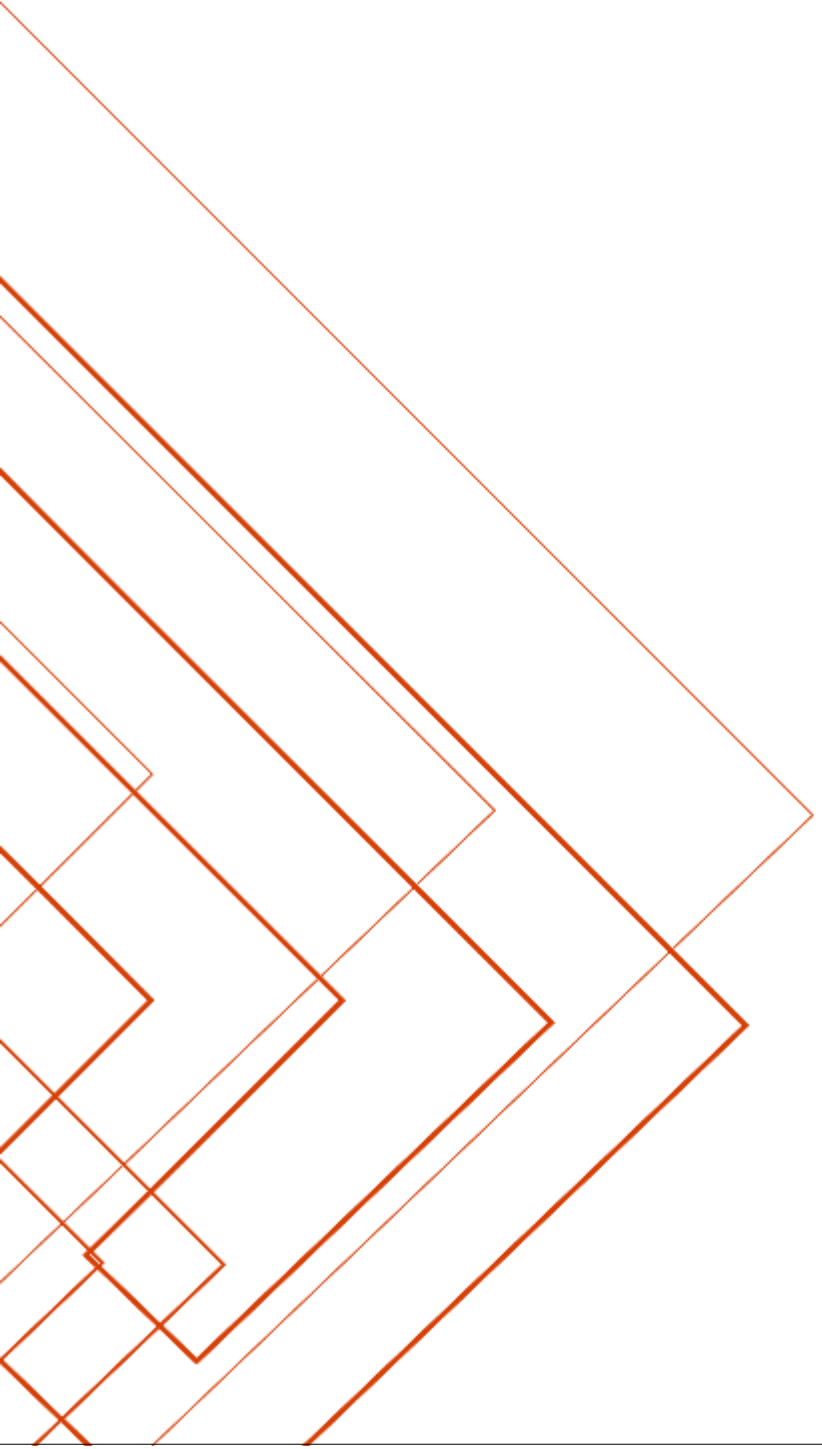
Introducción

SQL... Cuidado



Ejercicio 1

- Supongamos que vamos a desarrollar una aplicación para escuchar música. Necesitaremos:
  - Canciones
  - Artistas
  - Álbumes
  - Usuarios
  - Listas de reproducción
- Crea un modelo entidad relación para la base de datos de la aplicación.
- Conviértelo en tablas de una base de datos.



SQL *basics*

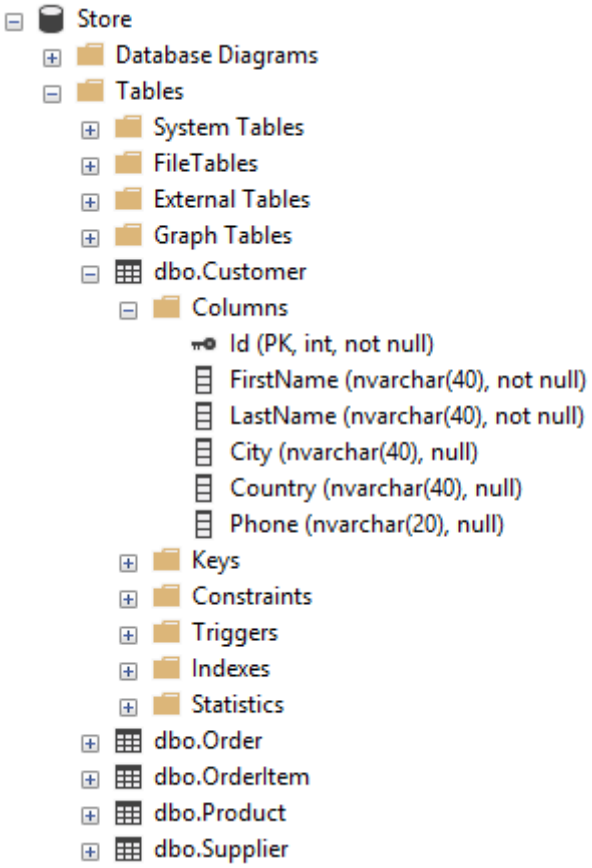
SQL *basics*

Elementos SQL (I)

- **Base de datos (`DATABASE`):** se puede definir como un conjunto de objetos (tablas, índices, restricciones, usuarios, datos…).
- **Tabla (`TABLE`):** estructura fija de campos.
- **Campo (`COLUMN, FIELD`):** son las columnas de una tabla.
- **Registro (`REGISTER, ROW`):** son las filas de una tabla (también se pueden denominar tuplas).
- **Clave primaria (`PK, PRIMARY KEY`):** identificador unívoco de una fila. Todas las PKs tienen asociado un índice siempre.
- **Clave foránea (`FK, FOREIGN KEY`):** establece una relación entre dos tablas. La relación se establece a una PK de otra tabla.
- **Índice (`INDEX`):** permite un acceso rápido a los datos.

SQL *basics*

Elementos SQL (II)



SQL *basics*

Tipos de datos (I)

- **Numérico:** para almacenar números enteros o decimales. Numeric Type Overview
  - **Exactos:**
    - **`INT`:** para números enteros.
    - **`BIT` o `BOOL`:** para elementos del tipo booleano o flags.
    - **`DECIMAL(precision, scale)` o `NUMERIC (precision, scale)`:** donde `precision` es el número de dígitos totales y `scale` el número de dígitos de la parte decimal.
  - **Aproximados:**
    - **`FLOAT`:** números con precisión simple. Tiene una precisión aproximada de 7 decimales.
    - **`DOUBLE`:** números con precisión doble. Tiene una precisión aproximada de 15 decimales.
    - **`REAL`**

- **Fecha y hora:** para almacenar campos de tipo fecha o de tipo horario. Date and Time Type Overview
  - **`DATE`:** almacena únicamente fechas ('yyyy-mm-dd').
  - **`DATETIME`:** combina fecha y hora ('yyyy-mm-dd hh:mm:ss')
  - **`TIMESTAMP`:** número de segundos transcurrido desde '1970-01-01 00:00:00' UTC.

SQL *basics*

Tipos de datos (II)

- **Cadenas:** para almacenar cadenas de texto. String Type Overview
  - **CHAR(n):** cadenas de longitud fija (n). Se puede establece el CHARACTER SET a almacenar en la definición del campo.
  - **VARCHAR(n):** cadenas de longitud variable (n indica la longitud máxima). Se puede establece el CHARACTER SET al almacenar en la definición del campo.
  - **TEXT, BLOB**: para campos de texto/binario más grandes. Por ejemplo: cuerpos de noticias, ficheros, etc…

- **NULL:** representa un valor desconocido en cualquiera de los tipos de datos anteriores.

Id	FirstName	LastName	City	Country	Phone
1	Maria	Anders	Berlin	Germany	030-0074321
2	Ana	Trujillo	México D.F.	Mexico	(5) 555-4729
3	Antonio	Moreno	México D.F.	Mexico	(5) 555-3932
4	Thomas	Hardy	London	UK	(171) 555-7788
5	Christina	Berglund	Luleå	Sweden	0921-12 34 65
6	Hanna	Moos	Mannheim	Germany	0621-08460
7	Frédérique	Citeaux	Strasbourg	France	88.60.15.31
8	Martín	Sommer	Madrid	Spain	(91) 555 22 82
9	Laurence	Lebihan	Marseille	France	91.24.45.40

Id	OrderId	ProductId	UnitPrice	Quantity
1	1	11	14.00	12
2	1	42	9.80	10
3	1	72	34.80	5
4	2	14	18.60	9
5	2	51	42.40	40
6	3	41	7.70	10
7	3	51	42.40	35

SQL *basics*

Operadores básicos (I)

Tipo	Operador	Acción
Aritméticos	+	Operación de suma.
	-	Operación de resta.
	*	Operación de multiplicación.
	/	Operación de división.
	%	Operación de módulo (resto de la división)
Lógicos	**AND**	TRUE si ambas expresiones booleanas son TRUE.
	OR	TRUE si cualquiera de las dos expresiones booleana es TRUE.
	NOT	Invierte el valor de cualquier otro operador booleano.

SQL *basics*

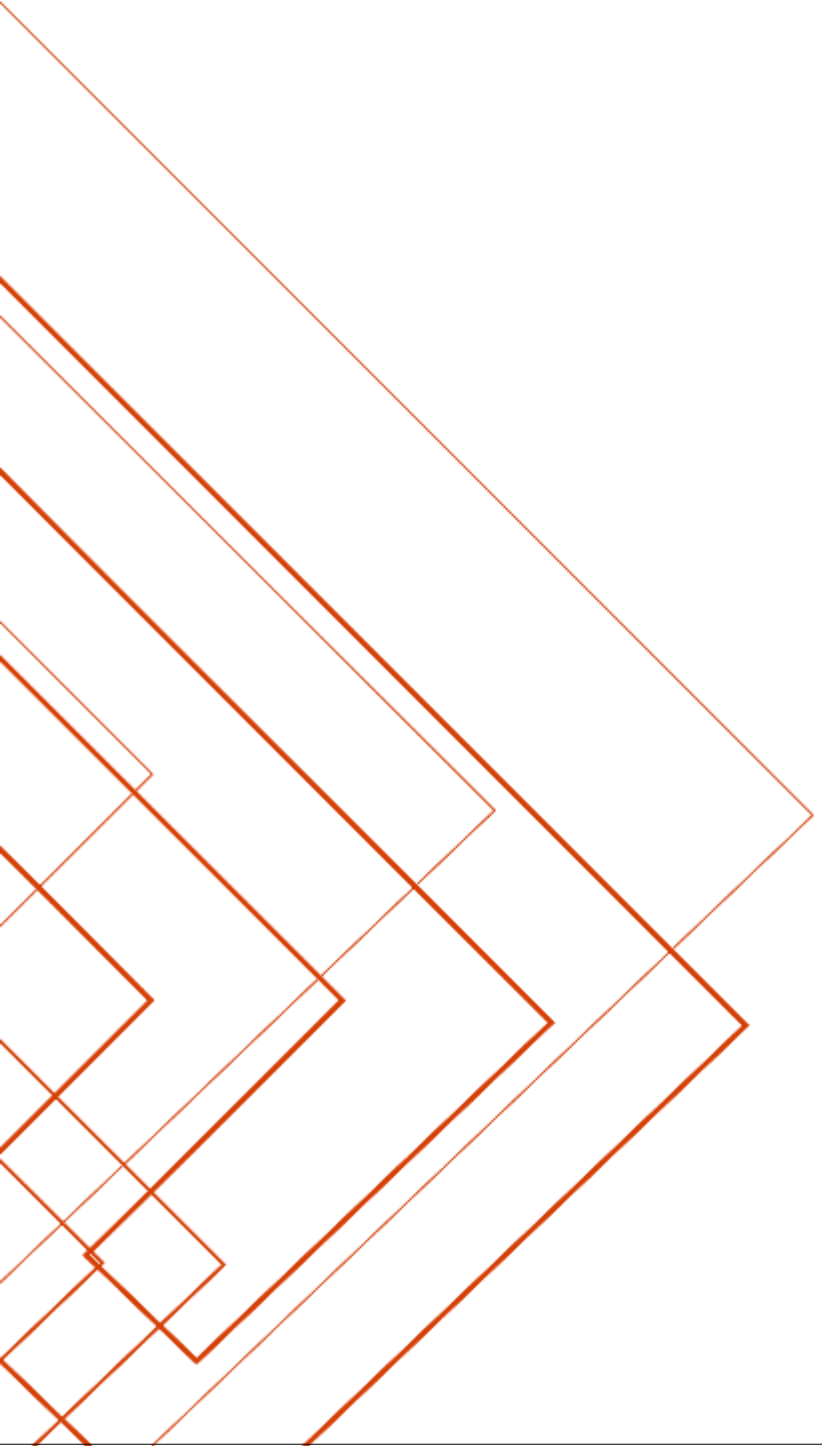
Operadores básicos (I)

Tipo	Operador	Acción
Comparación	=	Igualdad.
	>	Mayor que...
	>=	Mayor o igual que...
	<	Menor que...
	<=	Menor o igual que…
	`IS NULL`	Selecciona únicamente los valores nulos (`NULL`).
	`IS NOT NULL`	Selecciona únicamente los valores no nulos (`NULL`).
	`BETWEEN`	TRUE si el operando está dentro de un intervalo.
	`IN`	TRUE si el operando es igual a uno de la lista de expresiones.
	`LIKE`	TRUE si el operando coincide con un patrón.
	`EXISTS`	TRUE si una subconsulta contiene cualquiera de las filas.

Ejercicio 2

- Dentro de la máquina **MDS – DS** abre **Microsoft SQL Server Management Studio**.
- Abre el fichero **store_ddl.sql** y **store_dml.sql** que encontrarás dentro del fichero ***store_db.zip***.
- Ejecuta el fichero como si fuera un script:

- Explora las tablas, sus campos, claves primarias, claves foráneas. Familiarízate con todos los objetos de la base de datos.



SQL DML (*Data Manipulation Language*)

SQL DML (*Data Manipulation Language*)

Consulta – `SELECT statement`

Se usa para **extraer la información** de la base de datos. El resultado de la consulta se almacena en otra "tabla" denominada *resultset*. Normalmente denominamos a una consulta **QUERY**.

```
SELECT <column_list>
FROM <table_list>
[WHERE <conditions>];
```

- **`<column_list>`:** lista de todos los atributos a recuperar por la QUERY. Separados por comas. En el caso de no seleccionar atributos se emplea el asterisco (**`*`**). Es posible renombrar una columna (o tabla) empleando **`AS`**. Se emplea sobre todo cuando se realizan cruces de tablas que tengan campos con el mismo nombre, así evitamos ambigüedades.

- **`<table_list>`:** lista de las tablas que procesará la QUERY. Separadas por comas.

- **`<conditions>`:** expresión booleana que deberán cumplir los registros (**tuplas**) que devolverá la QUERY. Es opcional, no tienen porque establecerse filtros.

Ejercicio 3

- Obtén un listado de todos los clientes.
- Selecciona únicamente el ***FirstName***, ***LastName*** y ***Phone***.
- Renombra los campos de la siguiente manera:
  - **FirstName – Nombre**
  - **LastName – Apellido**
  - **Phone – Telefono**

SQL DML (*Data Manipulation Language*)

Filtrado – `WHERE clause`

Se usa dentro de una consulta (QUERY) para filtrar la información a recuperar de una base de datos. Dentro de la clausula **`WHERE`** podemos emplear los operadores condicionales vistos anteriormente.

```
SELECT <column_list>
FROM <table_list>
WHERE <condition> [AND|OR <condition>];
```

Ejercicio 4

- ¿Qué clientes son de Reino Unido?
- ¿Cuál es el teléfono de **“Howard Snyder”**?
- Selecciona el nombre del producto cuyo precio unitario se encuentre entre 100 y 250.
- Selecciona todos los campos de las tablas ***Customer*** y ***Product***. ¿Qué ha pasado?

SQL DML (*Data Manipulation Language*)

Unión de tablas – `JOIN clause` (I)

Las uniones de tablas se emplean para combinar registros o filas de dos o más tablas. Para realizar la unión es necesario que dichas tablas tengan campos en común (PKs y FKs).

Existen distintos tipos de JOIN:

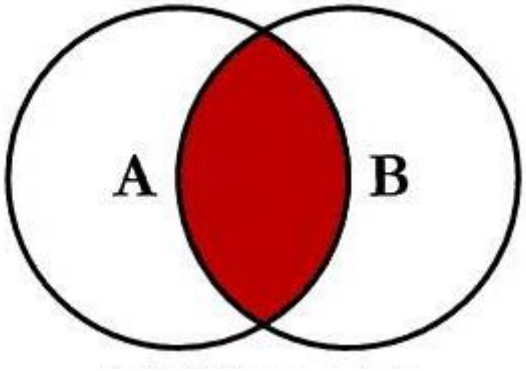
- **`INNER JOIN`:** devuelve todos los registros para los que hay coincidencia entre **ambas** tablas.
- **`LEFT JOIN`:** devuelve **todos** los registros de la tabla de la **izquierda**, y las filas que coinciden de la tabla de la derecha.
- **`RIGHT JOIN`:** devuelve **todos** los registros de la tabla de la **derecha**, y las filas que coinciden de la tabla de la izquierda.
- **`FULL OUTER JOIN`:** devuelve **todas** las filas haya o no coincidencia.

SQL DML (*Data Manipulation Language*)

Unión de tablas – `INNER JOIN clause` (II)

Devuelve todos los registros para los que hay coincidencia entre ambas tablas.

```
SELECT <column_list>
FROM TableA A
INNER JOIN TableB B
ON A.KEY = B.KEY;
```



Ejercicio 5

- Crea una consulta que devuelva el nombre y apellidos de un cliente junto con los pedidos que ha realizado. ¿Se te ocurre más de una manera de realizarla?

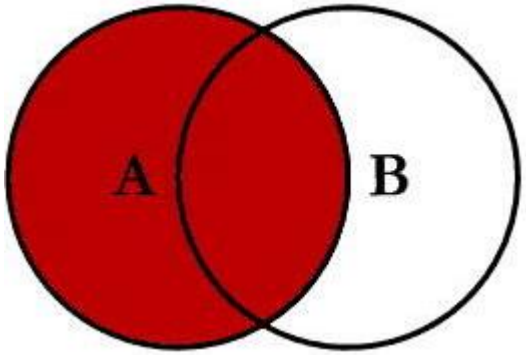
- Añade al resultado de la consulta anterior el nombre de los productos comprados. Realiza la consulta de las dos maneras que hemos aprendido. ¿Cuál te gusta más?

SQL DML (*Data Manipulation Language*)

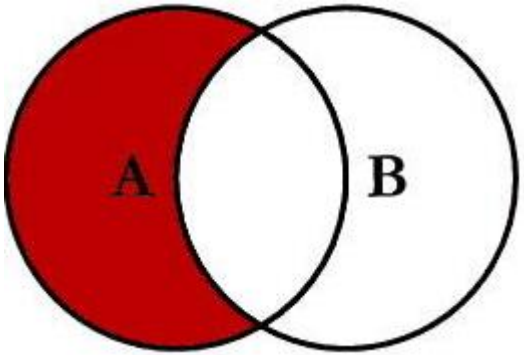
Unión de tablas – `LEFT JOIN clause` (III)

Devuelve todas las filas de la tabla de la izquierda (tabla A), con las filas coincidentes de la tabla de la derecha (tabla B). El resultado será `NULL` en el lado derecho cuando no haya ninguna coincidencia. Es posible eliminar todo el lado derecho de los resultados.

```
SELECT <column_list>
FROM TableA A
LEFT JOIN TableB B
ON A.KEY = B.KEY;
```




```
SELECT <column_list>
FROM TableA A
LEFT JOIN TableB B
ON A.KEY = B.KEY
WHERE B.KEY IS NULL;
```



Ejercicio 6

- Crea una consulta que devuelva el código del cliente y los códigos de los pedidos que ha realizado. La consulta debe devolver **todos** los clientes.

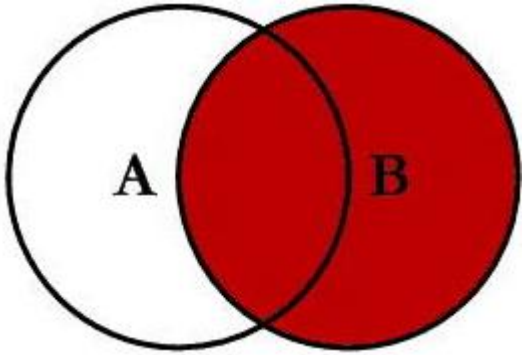
- Modifica la consulta anterior para quedarte únicamente con aquellos clientes que **no** han realizado ningún pedido. ¿Se te ocurre alguna otra manera de realizar la consulta? Ver los operadores básicos.

SQL DML (*Data Manipulation Language*)

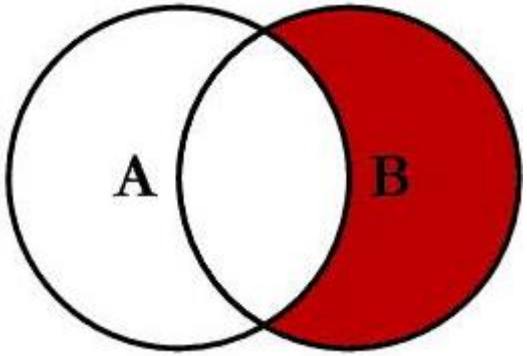
Unión de tablas – `RIGHT JOIN clause` (IV)

Devuelve todas las filas de la tabla de la derecha (tabla A), con las filas coincidentes de la tabla de la izquierda (tabla B). El resultado será `NULL` en el lado izquierdo cuando no haya ninguna coincidencia. Es posible eliminar todo el lado izquierdo de los resultados.

```
SELECT <column_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.KEY = B.KEY;
```




```
SELECT <column_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.KEY = B.KEY
WHERE B.KEY IS NULL;
```



Ejercicio 7

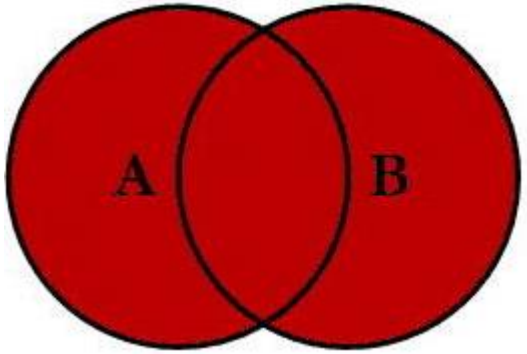
- Cambia la consulta anterior para que funcione con un `RIGHT JOIN`.
- ¿Qué conclusión puedes sacar?

SQL DML (*Data Manipulation Language*)

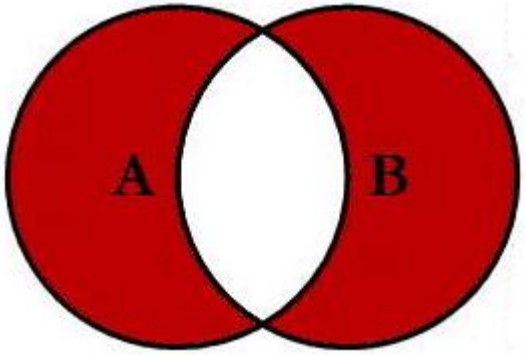
Unión de tablas – `FULL OUTER JOIN` clause (V)

Devuelve todas las filas cuando hay una coincidencia en una de las tablas. El resultado será `NULL` en el lado izquierdo cuando no haya ninguna coincidencia. El resultado será `NULL` en el lado derecho cuando no haya ninguna coincidencia.

```
SELECT <column_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.KEY = B.KEY;
```

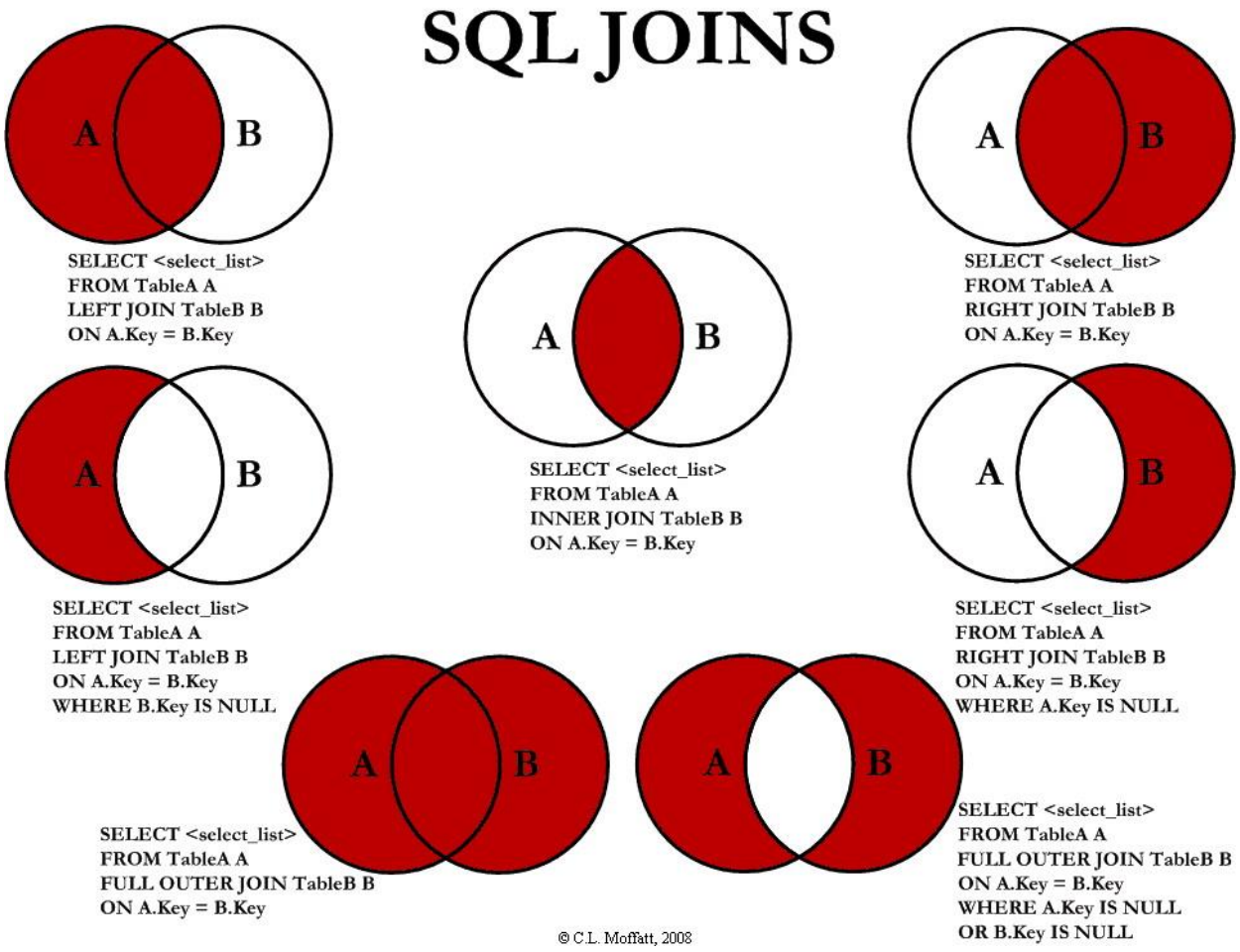



```
SELECT <column_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.KEY = B.KEY
WHERE A.KEY IS NULL
OR  B.KEY IS NULL;
```



SQL DML (*Data Manipulation Language*)

Unión de tablas – `JOIN clause` (VI)



SQL DML (*Data Manipulation Language*)

Agrupación y agregación – `GROUP BY clause` (I)

Agrupan los resultados por una o más columnas. Se emplea junto con las funciones de agregación.

```
SELECT <column_list>, aggregate_function(<column>)
FROM <table_list>
[WHERE <conditions>]
GROUP BY <column_list>;
```

SQL DML (*Data Manipulation Language*)

Agrupación y agregación – `GROUP BY clause` (II)

Las funciones de agregación realizan una determinada operación sobre una de las columnas de la tabla. Las funciones de agregación más extendidas se pueden ver en la siguiente tabla:

Función	Operación
`AVG(column)`	Calcula la media de las columna.
`COUNT(column)`	Cuenta el número de filas.
`FIRST(column)`	Devuelve el primer valor.
`LAST(column)`	Devuelve el último valor.
`MAX(column)`	Devuelve el valor máximo.
`MIN(column)`	Devuelve el valor mínimo.
`SUNM(columna)`	Calcula la suma de todos los registros de la columna.

Ejercicio 8

- ¿Cuántos clientes hay de cada país?
- ¿Cuál es el precio medio de los productos de cada pedido?
- ¿Cuántos productos en total tiene cada pedido?

SQL DML (*Data Manipulation Language*)

Agrupación y agregación – `HAVING clause`

Se utiliza dentro de una consulta (QUERY) para filtrar la información a recuperar de una base de datos tras una agrupación y/o agregación.

```
SELECT <column_list>, aggregate_function(<column>)
FROM <table_list>
[WHERE <conditions>]
GROUP BY <column_list>
HAVING <having_conditions>;
```

La expresión booleana de `<having_conditions>` se ejecuta en último lugar, tras aplicar las agrupaciones y funciones de agregación pertinentes. Se usarán funciones de agregación dentro de dicha condición.

Ejercicio 9

- ¿Qué nombres de clientes se repiten más de una vez? ¿Cuántas veces?

- ¿Alguno de los productos se ha comprado menos de 10 veces? Considera aquellos que no se han comprado ninguna vez.

SQL DML (*Data Manipulation Language*)

Ordenación – `ORDER BY clause`

Se utiliza para ordenar los resultados de una QUERY por una o más columnas del *resultset*. Se puede establecer ordenación ascendente o descendente (por defecto, la ordenación es ascendente). El criterio de ordenación puede ser diferente para cada columna.

```
SELECT <column_list>
FROM <table_list>
[WHERE <conditions>]
ORDER BY <column1> ASC|DESC, <column2> ASC|DESC;
```

Ejercicio 10

- Obtén un listado de clientes con su nombre, apellido y sus pedidos. Ordena el listado según la fecha del pedido de manera descendente.

- Acaba de ordenar el listado anterior, por nombre y apellidos de manera ascendente.

- Obtén un listado del identificador de producto y las veces que se ha comprado. Ordena el listado por la segunda columna mencionada.

SQL DML (*Data Manipulation Language*)

Combinación – `Union operator`

Permite combinar varios *resultsets* en uno solo. Cada *resultset* tiene que tener el mismo número de columnas, ser del mismo tipo y estar en el mismo orden. Por defecto, no permite valores duplicados. Para permitir duplicados se utiliza el modificador **ALL**.

```
SELECT <column_list>
FROM table1
UNION [ALL]
SELECT <column_list>
FROM table2
UNION [ALL]
SELECT <column_list>
FROM table3;
```

Ejercicio 11

- Junta las tablas "*Customer*" y "*Orders*", teniendo en cuenta el identificador de cliente.

- Repite el apartado anterior pero permitiendo duplicados.

SQL DML (*Data Manipulation Language*)

Eliminación de duplicador – `Distinct operator`

El operador `DISTINCT` devuelve únicamente valores diferentes de una o más columnas (elimina duplicados). Se puede producir el mismo efecto mediante un `GROUP BY`.

```
SELECT DISTINCT <column_list>
FROM <table_list>
[WHERE <conditions>];
```

Ejercicio 12

- Hallar las fechas en las que se ha realizado al menos un pedido y ordénalas de forma descendente.

SQL DML (*Data Manipulation Language*)

Patrones – `LIKE operator`

El operador LIKE empleado sobre una cadena detecta patrones. Dentro del patrón se pueden emplear *wildcards*.

```
SELECT <column_list>
FROM <table_list>
WHERE <column> LIKE pattern;
```

Un *wildcard* permite sustituir uno o varios caracteres por otros en una cadena de texto.

Wildcard	Descripción
%	Sustitución de cero o más caracteres.
_	Cuenta el número de filas.

Ejercicio 13

- Encuentra los clientes que tienen la cadena 'Ma' en su nombre.
- Incluye a la consulta anterior los clientes que tienen en su número de teléfono la cadena '91'.

SQL DML (*Data Manipulation Language*)

Paginación – `TOP operator`

Se emplea para especificar cuántos registros se quieren devolver. **No es estándar, por lo que otros lenguajes SQL usarán otros operadores y/o sintaxis.**

```
SELECT TOP # <column_list>
FROM <table_list>
[WHERE <conditions>];
```

Ejercicio 14

- Haz un ranking con los 10 productos más comprados.

SQL DML (*Data Manipulation Language*)

Funciones SQL

Igual que las funciones de agregación, realizan una determinada operación sobre los valores de una de las columnas.

Función	Operación
`UPPER(column)`	Convierte el valor de la columna a mayúsculas.
`LOWER(column)`	Convierte el valor de la columna a minúsculas.
`LEN(column)`	Devuelve la longitud de la columna (número de caracteres).
`ROUND(column,decimals)`	Redondea el valor de una columna a los decimales especificados.
`NOW()`	Devuelve el día y la hora del sistema.
`IFNULL(column, value)`	Convierte el valor de la columna por el parámetro `value` en el caso de que sea `NULL`.

Ejercicio 15

- Crea una consulta que devuelva el nombre de los clientes en mayúscula y el apellido en minúsculas.
- Obtén un listado de todos los clientes cuyo apellido tenga más de 5 caracteres.
- Calcula el precio medio de todos los pedidos y redondéalo a dos decimales.

SQL DML (*Data Manipulation Language*)

Inserción – `INSERT statement`

Se utiliza para **insertar** nuevos registros en una tabla de base de datos.

```
INSERT INTO <table_name>[(<column_list>)]
VALUES (<value_list>);
```

- **`<table_name>`:** tabla donde se insertará el nuevo registro.
- **`<column_list>`:** lista de las columnas o campos de la tabla de base de datos a rellenar. Si no se establecen, la base de datos entenderá que se pasarán todos lo campos de la tabla y en el mismo orden que fueron creados.
- **`<value_list>`:** valores a insertar en el nuevo registro de la tabla.

SQL DML (*Data Manipulation Language*)

Actualización – `UPDATE statement`

Se utiliza para **actualizar** registros existentes en una tabla de base de datos.

```
INSERT INTO <table_name>[(<column_list>)]
VALUES (<value_list>);
```

- **`<table_name>`:** tabla donde se actualizarán los registros.
- **`<column1> = <value1>…`:** nuevos valores para las columnas que sea necesario actualizar.
- **`<conditions>`:** es posible actualizar sólo determinados registros que cumplen una condición.

¡Atención! Si no se establecen condiciones se actualizarán todos los registros de la tabla.

SQL DML (*Data Manipulation Language*)

Borrado – `DELETE statement`

Se utiliza para **eliminar** registros existentes en una tabla de base de datos.

```
DELETE
FROM <table_name>
[WHERE <conditions>];
```

- **`<table_name>`:** tabla donde se efectuará la operación de borrado de registros.
- **`<conditions>`:** borrará aquellos registros que cumplan la expresión booleana.

¡Atención! Si no se establecen condiciones se borrarán todos los registros de la tabla.

SQL DML (*Data Manipulation Language*)

Borrado – `TRUNCATE statement`

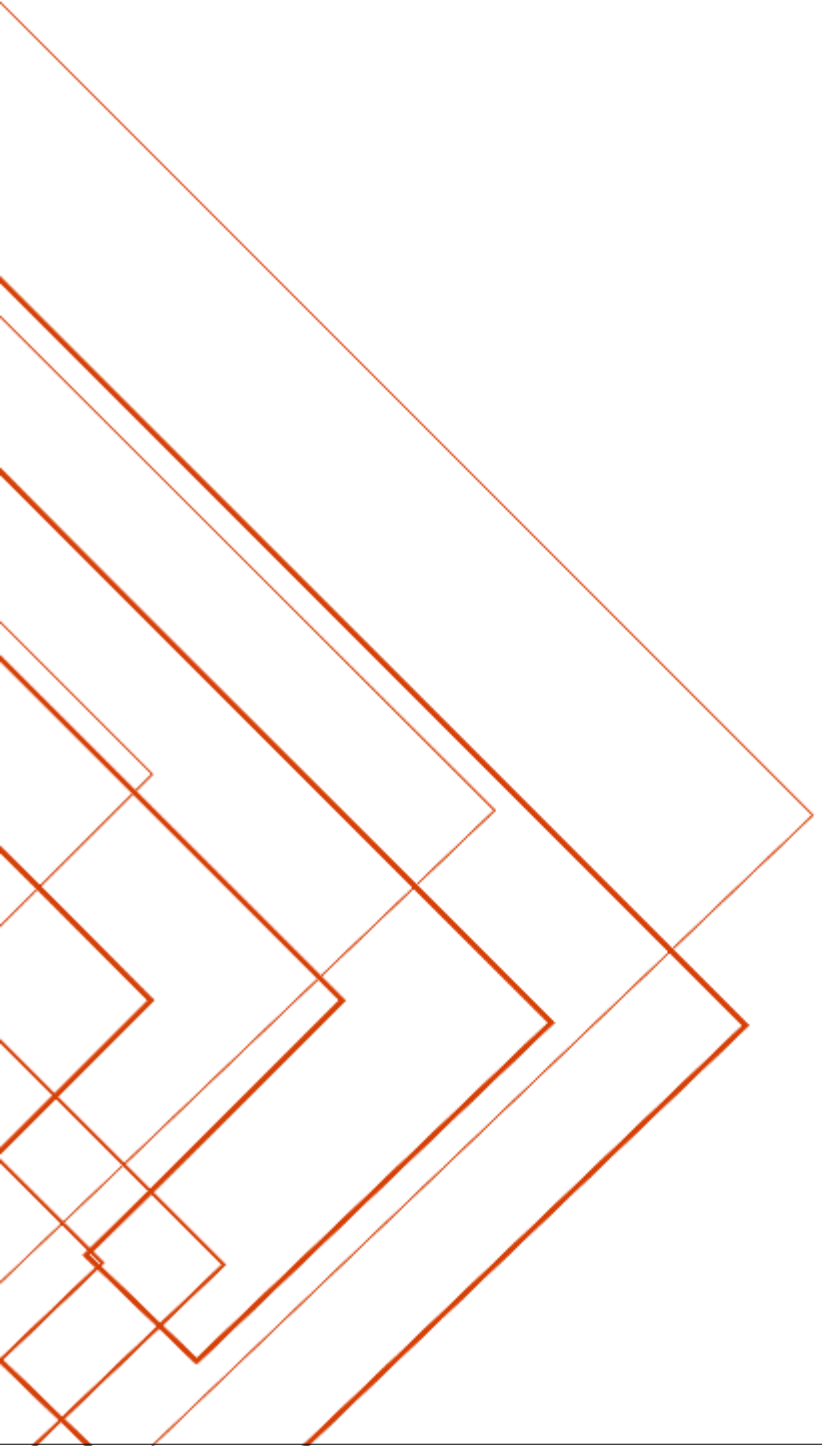
Para eliminar el contenido al completo de una tabla es más eficiente emplear la sentencia **TRUNCATE**, en lugar que un **DELETE FROM** sin clausula **WHERE**. Tras ejecutar la sentencia la tabla estará vacía.

```
TRUNCATE TABLE <table_name>;
```

- **<table_name>:** nombre de tabla a truncar.

Ejercicio 16

- Añade un cliente nuevo obteniendo el siguiente Id disponible.
- Actualiza el número de teléfono del cliente añadido anteriormente.
- Elimina todos los pedidos realizados el 4 de julio del 2012.



SQL DML (*Data Definition Language*)

SQL DDL (*Data Definition Language*)

Creación/Eliminación de base de datos – `CREATE/DROP DATABASE statement`

Sentencia para crear una base de datos vacía.

```
CREATE DATABASE <database_name>;
```

Sentencia para eliminar una base de datos.

¡Atención! Si eliminamos una base de datos perderemos todo su contenido, tanto datos como objetos.

```
DROP DATABASE <database_name>;
```

- **`<database_name>`:** nombre de la base de datos a crear o eliminar.

SQL DDL (*Data Definition Language*)

Uso de base de datos – `USE DATABASE statement`

Sentencia para usar una base de datos en concreto.

```
USE <database_name>;
```

Desde el momento en que ejecutemos esta sentencia todas las operaciones (queries, updates, creates, etc.) se ejecutarán en la base de datos seleccionada.

Desde `Microsoft SQL Server Management Studio`, podemos seleccionar la base de datos en uso.

SQL DDL (*Data Definition Language*)

Creación de tablas – `CREATE TABLE statement`

Sentencia usada para crear nuevas tablas en base de datos.

```
CREATE TABLE <table_name>
(
        <column_name1> data_type1,
        <column_name2> data_type2,
        <column_name3> data_type3,
        …
);
```

- **`<table_name>`:** nueva tabla a crear.

- **`<column_name1> data_type1…`:** columnas de la tabla junto con el tipo de datos que tiene cada una de ellas.

SQL DDL (*Data Definition Language*)

Creación de tablas – Restricciones

Las restricciones (`CONSTRAINTS`) sirven para establecer reglas y comprobaciones sobre los datos. En el caso de que no se cumpla la restricción, la acción realizada sobre la tabla (inserción o actualización) se aborta y el motor de base de datos lanzará un error. Existen los siguientes tipos:

- `PRIMARY KEY`
- `FOREIGN KEY`
- `DEFAULT`
- `NOT NULL`
- `UNIQUE`
- `CHECK`

¡Atención! Las restricciones garantizan la integridad de la información de la base de datos, pero a su vez consumen recursos en el momento de inserción / actualización / borrado. Sacrifican rendimiento por integridad. Esto puede ser problemático en entornos con grandes volúmenes de información, en los que la velocidad de procesado es crítica.

SQL DDL (*Data Definition Language*)

Creación de tablas – `PRIMARY KEY constraint`

Es posible añadir una clave primaria a una tabla en el momento de creación de la misma. La clave primaria debe contener valores únicos y no nulos. Al crear una clave primaria se creará un índice asociado.

```
CREATE TABLE <table_name>
(
        <column_name> data_type,
        …,
        [CONSTRAINT <pk_name>] PRIMARY KEY (<column_name>, …)
);
```

- **<pk_name>:** nombre de la clave primaria a crear. El nombre es opcional, si no se pone uno, el sistema de base de datos asignará uno por defecto.

- **<column_name>…:** columna o columnas que formarán parte de la clave primaria de la tabla.

SQL DDL (*Data Definition Language*)

Creación de tablas – `FOREIGN KEY constraint`

Es posible añadir una clave foránea a una tabla en el momento de creación de la misma. La clave foránea referencia un campo que es clave primaria de otra tabla.

```
CREATE TABLE <table_name>
(
        <column_name> data_type,
        …,
        [CONSTRAINT <fk_name>] FOREIGN KEY (<column_name>) REFERENCES
        <table_name_ref>(<pk_table_name_ref>)
);
```

- **<fk_name>:** nombre de la clave foránea a crear. El nombre es opcional, si no se pone uno, el sistema de base de datos asignará uno por defecto.
- **<column_name>…:** columna que tiene la restricción.
- **<table_name_ref>…:** nombre de la tabla a la que hace referencia la clave foránea.
- **<pk_table_name_ref>…:** campo de la tabla a la que hace referencia la clave foránea.

SQL DDL (*Data Definition Language*)

Creación de tablas – `DEFAULT constraint`

Permite establecer un valor por defecto para una columna en el caso de dejarla vacía en el momento de la inserción.

```
CREATE TABLE <table_name>
(
        <column_name> data_type DEFAULT <default_value>,
        …,
);
```

- **<default_value>…:** valor por defecto de la columna. Tiene que estar acorde con el tipo de dato que almacena la columna.

SQL DDL (*Data Definition Language*)

Creación de tablas – `NOT NULL constraint`

Establece que un campo/columna no puede contener nunca valores nulos.

```
CREATE TABLE <table_name>
(
      <column_name> data_type NOT NULL,
      …,
);
```

SQL DDL (*Data Definition Language*)

Creación de tablas – `UNIQUE constraint`

Establece que un campo/columna no puede contener nunca valores repetidos, su valor siempre tiene que ser único.

```
CREATE TABLE <table_name>
(
        <column_name> data_type UNIQUE,
        …,
);
```

Las columnas de una clave primaria, simplemente por el hecho de serlo, siempre tienen dos restricciones: son únicas (`UNIQUE`) y no puede contener valores nulos (`NOT NULL`).

SQL DDL (*Data Definition Language*)

Creación de tablas – `CHECK constraint`

Es posible añadir condiciones de manera que limiten los valores que pueden tomar los registros en una columna. Por ejemplo: `column_value1 > 10 AND column_value2 = 1000`.

```
CREATE TABLE <table_name>
(
        <column_name> data_type,
        …,
        [CONSTRAINT <chk_name>] CHECK (<chk_condition>)
);
```

- **<chk_name>:** nombre de la comprobación/restricción que se creará. El nombre es opcional, si no se pone uno, el sistema de base de datos asignará uno por defecto.

- **<chk_condition>…:** expresión booleana para comprobar el contenido del registro.

SQL DDL (*Data Definition Language*)

Creación de tablas – `IDENTITY field`

La opción de autoincrementado genera un número que se incrementa de manera automática en cada inserción. El valor del incremento se puede establecer y no tiene porque ser siempre 1 (valor por defecto).

Normalmente es empleado para la generación de valores de una PRIMARY KEY.

```
CREATE TABLE <table_name>
(<column_name> INT IDENTITY(<start_value>,<increment_value>),
        …,
);
```

Microsoft SQL Server

- **<start_value>:** valor inicial.
- **<increment_value>:** valor a incrementar en cada inserción.

SQL DDL (*Data Definition Language*)

Creación de tablas – `ALTER TABLE statement (I)`

Mediante esta sentencia se pueden **añadir nuevas columnas** a una tabla de base de datos. Las columnas nuevas se añadirán al final de la tabla.

```
ALTER TABLE <table_name>
ADD <column_name> <data_type>;
```

Es posible establecer restricciones como las que hemos visto antes en el momento de creación de las nuevas columnas.

SQL DDL (*Data Definition Language*)

Modificación de tablas – `ALTER TABLE statement (II)`

Mediante esta sentencia se pueden **eliminar columnas existentes** de una tabla de base de datos.

ALTER TABLE <table_name>
DROP COLUMN <column_name>;

¡Atención! El proceso de eliminación de columnas debe realizarse con cuidado, no se pierda información.

SQL DDL (*Data Definition Language*)

Modificación de tablas – `ALTER TABLE statement (III)`

Mediante esta sentencia se puede **modificar el tipo de dato** que contiene una columna de una tabla de base de datos.

```
ALTER TABLE <table_name>
ALTER COLUMN <column_name> <data_type>;
```

¡Atención! Al cambiar una columna de tipo de dato los registros ya guardados en la tabla deben ser compatibles con el tipo de dato. En caso contrario se producirá un error o una perdida de información (por ejemplo al cambiar la precisión de un campo numérico).

SQL DDL (*Data Definition Language*)

Modificación de tablas – `ALTER TABLE statement (IV)`

También es posible **añadir una clave primaria** a una tabla después de su creación (en el caso de que no la tuviera).

```
ALTER TABLE <table_name>
ADD [CONSTRAINT < pk_name >] PRIMARY KEY (<column_name>, …);
```

Y borrarla.

```
ALTER TABLE <table_name> DROP PRIMARY KEY;
```

SQL DDL (*Data Definition Language*)

Modificación de tablas – `ALTER TABLE statement` (V)

También es posible **añadir claves foráneas** a una tabla después de su creación.

```
ALTER TABLE <table_name>
ADD [CONSTRAINT <fk_name>] FOREIGN KEY  <column_name>) REFERENCES
<table_name_ref>(<pk_table_name_ref>);
```

Y borrarlas.

```
ALTER TABLE <table_name> DROP FOREIGN KEY <fk_name>;
```

SQL DDL (*Data Definition Language*)

Modificación de tablas – `ALTER TABLE statement (VI)`

También es posible **añadir restricciones** a una tabla después de su creación.

```
ALTER TABLE <table_name>
ADD [CONSTRAINT <chk_name>] CHECK (<chk_condition>);
```

Y borrarlas.

```
ALTER TABLE <table_name> DROP CONSTRAINT <chk_name>;
```

SQL DDL (*Data Definition Language*)

Borrado de tablas – `DROP TABLE statement`

Elimina una tabla y todo su contenido de la base de datos.

```
DROP TABLE <table_name>;
```

¡Atención! Al borrar una tabla perderemos todo su contenido.

SQL DDL (*Data Definition Language*)

Creación de vistas – `CREATE VIEW statement`

Sentencia empleada para la **creación de una vista**. Una vista es una tabla virtual que se crea como el resultado de una `SELECT`.

```
CREATE VIEW <view_name> AS
SELECT <column_list>
FROM <table_list>
[WHERE <conditions>];
```

- **<view_name>:** nombre de la vista.

SQL DDL (*Data Definition Language*)

Modificación de vistas – `CREATE VIEW statement`

Sentencia empleada para la **modificación de una vista**.

```
CREATE VIEW <view_name> AS
SELECT <column_list>
FROM <table_list>
[WHERE <conditions>];
```

SQL DDL (*Data Definition Language*)

Modificación de vistas – `CREATE VIEW statement`

Sentencia empleada para la **eliminación de una vista** de la base de datos.

```
DROP VIEW <view_name>;
```

¡Atención! Al borrar una vista no se produce borrado de registros de base de datos.

Ejercicio 17

- Crea una base de datos para el ejercicio 1.
- Elabora las sentencias de creación de las tablas del ejercicio 1.
- Inserta registros de prueba en las tablas que has creado (`INSERT`).
- Inventa alguna sentencia de actualización y/o borrado (`UPDATE/DELETE`).

SQL avanzado

SQL avanzado

Subconsultas

Una subconsulta es una consulta anidada en una instrucción **SELECT**, **INSERT**, **UPDATE** o **DELETE**, o bien en otra subconsulta.

```
SELECT <column_list>
FROM <table_list>
WHERE <column_name1> expression operator
      ( SELECT <column_name2>
        FROM <table_name>
        [WHERE <conditions>] );
```

Una subconsulta por comparación (excepto **EXISTS** e **IN**) únicamente puede seleccionar una columna, siendo ésta del mismo tipo que la presente en la condición de la consulta externa.

Ejercicio 18

- Repite el segundo apartado del ejercicio 6 mediante el uso de subconsultas.

SQL avanzado

Importación de datos – `BULK INSERT statement`

Importa un archivo de datos en una tabla o vista de base de datos con un formato especificado por el usuario.

```
BULK INSERT <tabla_name>
FROM <data_file>
[ WITH (
                FIRSTROW = <first_row>,
                FIELDTERMINATOR = <field_terminator>,
                ROWTERMINATOR = <row_terminator>
) ];
```

- **`<first_row>`**: especifica el número de la primera fila que se va a cargar (por defecto 1).
- **`<field_terminator>`**: especifica el terminador de campo que se va a usar (por defecto \t).
- **`<row_terminator>`**: especifica el terminador de fila que se va a usar (por defecto \r\n).

Ejercicio 19

- Crea una tabla con las mismas características que OrderItem, eliminando la propiedad `IDENTITY` de la columna Id.
- Importa los datos del fichero ejercicio_19.csv en la tabla creada anteriormente.
- Borra el contenido de la tabla OrderItem.
- Rellena la tabla OrderItem con los datos importados. Para ello puedes emplear sentencias `INSERT SELECT`, que no es otra cosa que la unión de las sentencias `INSERT` y `SELECT`.

SQL avanzado

Transacciones (I)

Mediante estas sentencias SQL podemos **crear una transacción** (operación ACID) sobre la información de la base de datos. Al finalizar la modificación de información, podemos confirmar los cambios o descartarlos.

```
BEGIN TRANSACTION|TRAN;
<insert|update|delete statements>;
COMMIT|ROLLBACK TRAN;
```

Microsoft SQL Server

- **BEGIN TRANSACTION:** comienza la transacción en base de datos.
- **COMMIT:** finaliza la transacción confirmando los cambios.
- **ROLLBACK:** finaliza la transacción descartando los cambios.

SQL avanzado

Transacciones (II)

Por defecto, SSMS se ejecuta con el **modo autocommit habilitado**. Esto quiere decir, que tan pronto se ejecuta una actualización de información en base de datos, los cambios se convierten en permanentes. No se puede dar marcha atrás a los cambios. Se puede cambiar este modo de funcionamiento con la siguiente sentencia.

```
SET IMPLICIT_TRANSACTIONS ON;
```

SQL avanzado

Creación de usuarios

Los usuarios de una base de datos determinan quién puede tener acceso a la misma. También se puede establecer a que objetos de la base de datos y que tipo de permisos sobre los objetos tienen los usuarios (lectura, escritura, lectura/escritura…).

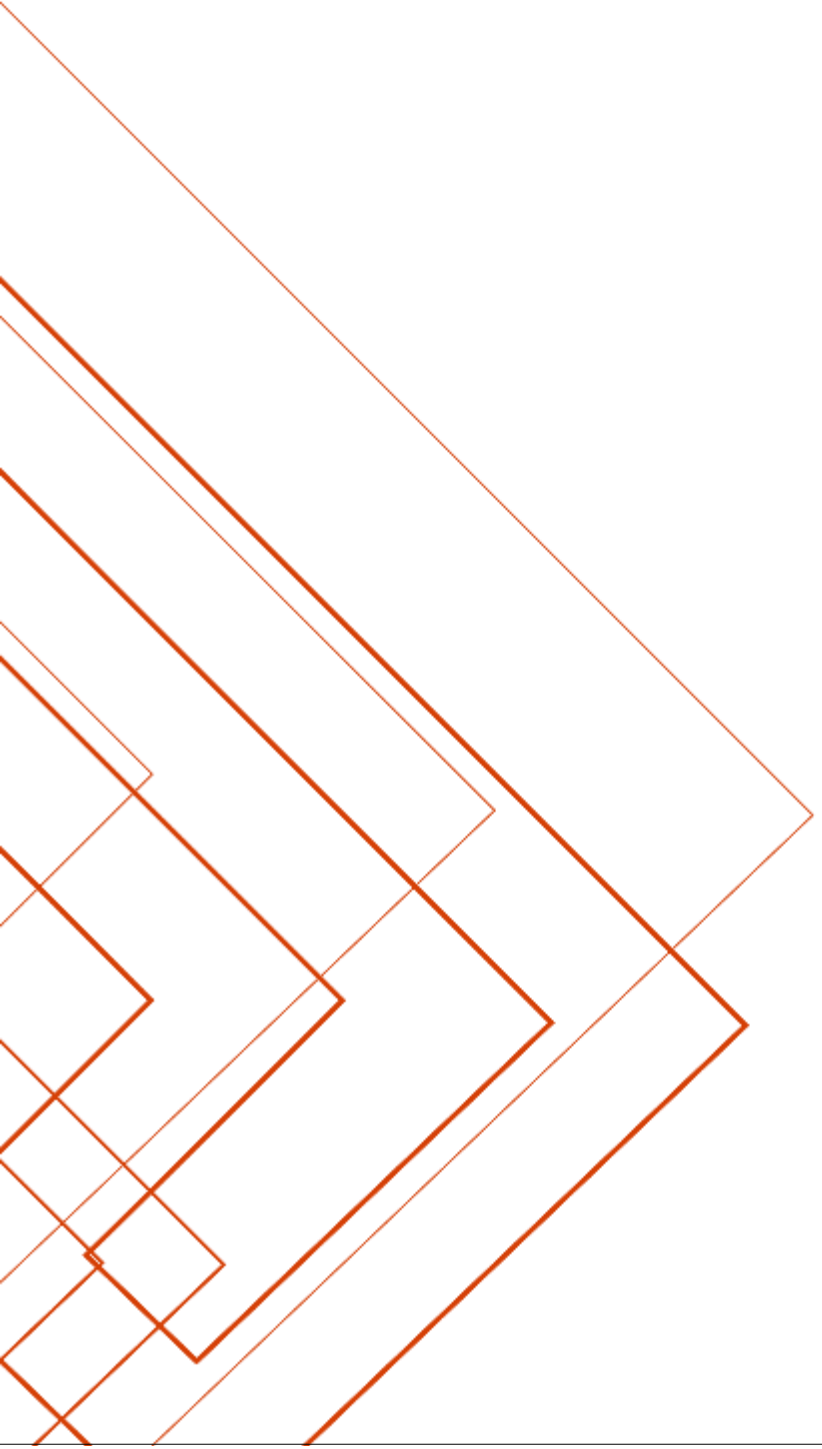
```
CREATE LOGIN <user> WITH PASSWORD = '<pwd>';
CREATE USER [<user>@<server>] FOR LOGIN <login_user>;
GRANT <priv_type> ON <object> TO <user>@<server>;
```

- **<user>:** nombre del usuario a crear.
- **<server>:** base de datos donde se creará el usuario.
- **<priv_type>:** tipo de privilegio que se dará al usuario.
- **<object>:** objeto o objetos para los que se dará privilegios al usuario.

https://docs.microsoft.com/es-es/sql/t-sql/statements/grant-transact-sql?view=sql-server-ver15
https://docs.microsoft.com/es-es/sql/t-sql/statements/grant-object-permissions-transact-sql?view=sql-server-ver15#permissions

Ejercicio 20

- Sobre la base de datos creada en el ejercicio 18:
  - Prueba a crear unas sentencias de actualización (INSERT, UPDATE, DELETE) dentro de una transacción. Prueba tanto a realizar un COMMIT como ROLLBACK al final de la misma.
  - Crea un usuario y dale permisos únicamente para realizar consultas.



Uso de SQL Server desde R y Python

Uso de SQL Server desde R y Python

Paquetes/Módulos a utilizar en SQL Server

Para acceder a SQL Server desde R, el paquete más sencillo y con un uso más extendido actualmente es **odbc** disponible a través de CRAN.

```
install.packages('odbc')
library(odbc)
```

Para acceder a SQL Server desde R, el paquete más sencillo y con un uso más extendido actualmente es **odbc** disponible a través de CRAN.

```
(pip | conda) install pyodbc
import pyodbc
```

Uso de SQL Server desde R y Python

Flujo de trabajo

El flujo de trabajo con SQL Server (y cualquier otra base de datos) desde R y Python siempre será el mismo:

1. Abrir una conexión a la base de datos: utilizando **nombre de usuario, contraseña, host y nombre de la base de datos**.

2. Enviar a través de la conexión una consulta bien de lectura (**SELECT**) o de escritura (**INSERT, UPDATE, DELETE, CREATE, ALTER**).

3. Procesar los resultados recibidos (**SELECT**) o los códigos de éxito/error (**INSERT, UPDATE, DELETE, CREATE, ALTER**) asociados a la consulta y liberar, en su caso, el conjunto de resultados recibido.

4. Cerrar la conexión abierta (**¡¡MUY IMPORTANTE!!**)

Uso de SQL Server desde R y Python

Apertura de conexión en SQL Server

Para abrir una conexión a SQL Server desde R:

```
conn <- dbConnect(odbc(), Driver='SQL Server', Server='xxx',
      Database='xxx', UID='xxx', PWD='xxx' , Port=1443)
```

Para abrir una conexión a SQL Server desde Python:

```
conn = pyodbc.connect(driver='{SQL Server}', server='xxx',
      database='xxx', user='xxx', password='xxx')
```

Uso de SQL Server desde R y Python

Envío de consultas y procesado de resultados

Para enviar una query a SQL Server y procesar los resultados desde R:

```
res <- dbSendQuery(conn, 'SELECT | INSERT …')
df <- dbFetch(res, n=<num_records>)
dbClearResults(res)
```

Para procesar los resultados de una query a SQL Server desde Python:

```
results = pd.read_sql_query(sql, conn)
```

Uso de SQL Server desde R y Python

Cierre de conexión

Para cerrar una conexión a SQL Server desde R:

```r
dbDisconnect(conn)
```

Para cerrar una conexión a SQL Server desde Python:

```python
conn.close()
```

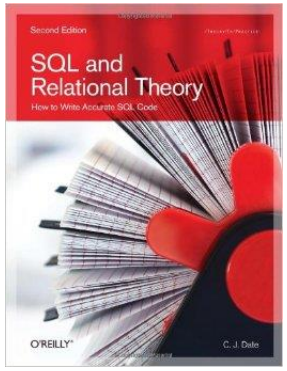
Materiales

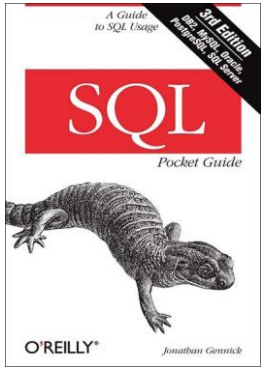
Materiales

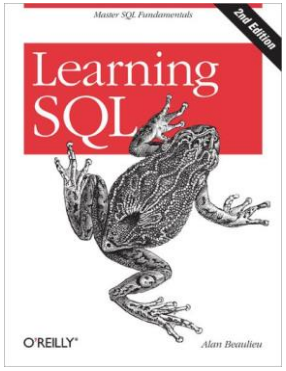
Materiales

- Try SQL: https://www.codeschool.com/courses/try-sql
- SQL Tutorial: http://www.w3schools.com/sql/

- SQL and Relational Theory
- SQL Pocket Guide
- Learning SQL







Afi Escuela