



SQL Básico

Tecnofor
by SNGULAR



Contenidos

- SQL Básico
 - Módulo 1: Conceptos básicos de bases de datos
 - Módulo 2: Introducción a SQL
 - Módulo 3: Ordenación y filtrado en SQL
 - Módulo 4: Combinación de varias tablas con JOIN en SQL
 - Módulo 5: Uso de funciones integradas y Group by en SQL
 - Módulo 6: Modificación de datos con SQL



Conceptos Básicos

- base de datos
 - Es una colección organizada de datos, de un mismo contexto, y almacenados para su utilización.
 - Las hay de distintos tipos:
 - Bibliográficas
 - De texto
 - Directorios telefónicos
 - Bibliotecas especializadas de información
 - Otras
 - Existen bases de datos de sólo lectura, como por ejemplo las de consulta histórica, y también de escritura, como las que almacenan información de transacciones comerciales.





Conceptos Básicos

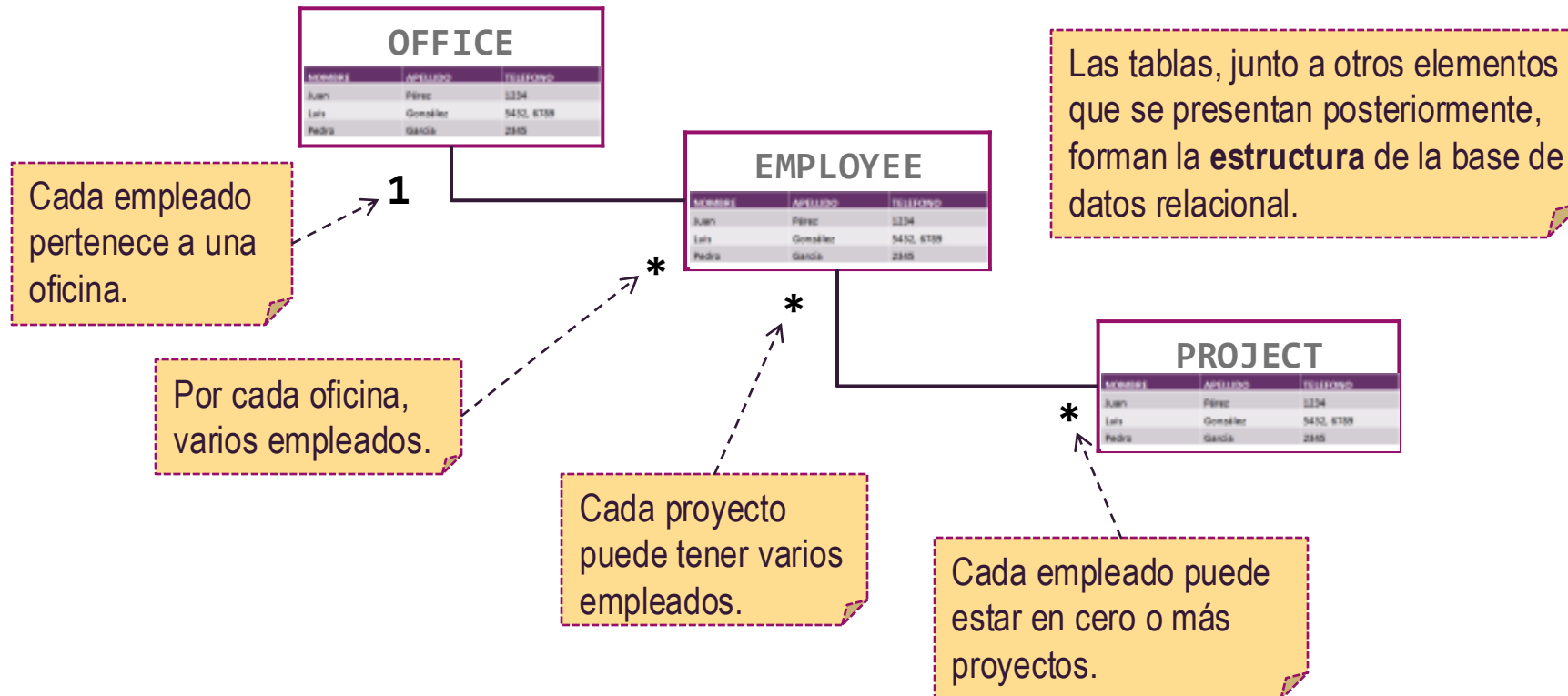
- ¿por qué una base de datos?
 - Con respecto a otras formas de almacenar información, las bases de datos tienen algunas ventajas:
 - Almacenamiento persistente.
 - Permite lectura y escritura de datos.
 - Múltiples conexiones.
 - Control de transacciones y concurrencia.
 - Maneja tipos de dato.

Conceptos Básicos

- modelos de bases de datos
 - Las bases de datos se pueden clasificar de acuerdo a la forma como se administran y organizan sus datos:
 - Jerárquicas: organización tipo árbol.
 - Transaccionales: permiten muchas operaciones concurrentes a gran velocidad.
 - Relacionales: conjunto de tablas relacionadas entre sí, que puede ser escrita y consultada a través de consultas.
 - Multidimensionales: orientadas a manejar grandes volúmenes.
 - Orientadas a objeto: almacenamiento de objetos complejos, utilizando los paradigmas de encapsulamiento, herencia y polimorfismo, de la OO.
 - Documentales: almacenamiento de documentos, con posibilidad de búsqueda por contenido.
 - En este curso, se trabaja con las bases de datos relacionales, muy utilizadas en la actualidad, y que permiten utilizar el lenguaje SQL.

Conceptos Básicos

- base de datos relacional
 - Una base de datos relacional es una colección de tablas con datos, y donde las tablas están relacionadas entre sí de acuerdo a la relación entre las entidades asociadas:



Conceptos Básicos

- tabla
 - Dentro de una base de datos relacional, es donde se guardan los datos. Su estructura se organiza en:

NOMBRE	APELLIDO	TELEFONO
Juan	Pérez	1234
Luis	González	5432
Pedro	García	2345

Conceptos Básicos

- DBMS
 - Para funcionar, una base de datos no sólo debe almacenar los datos. También debe proveer servicios que permitan consultarlos y administrarlos. Por ejemplo:
 - Manejar conexiones y comunicaciones remotas.
 - Control de errores de diverso tipo: estructura, datos, tiempo máximo de ejecución, inconsistencias, ...
 - Validación de valores de acuerdo al tipo de datos
 - Ejecución de consultas
 - Gestión de usuarios, roles y permisos
 - ...
 - Todo esto es realizado por un **DBMS** (Database Management System), conocido en castellano como SGBD (Sistema de gestión de bases de datos).

Conceptos Básicos

- algunos DBMS
 - Algunos de los **DBMS** más conocidos son:
 - Comerciales:
 - Oracle Database (Oracle)
 - SQL Server (Microsoft)
 - DB2 (IBM)
 - Abiertos:
 - MySQL (Oracle)
 - PostgreSQL o Postgres
 - H2
 - HSQL
 - Derby (Java DB)
 - SQLite

ORACLE[®]
DATABASE



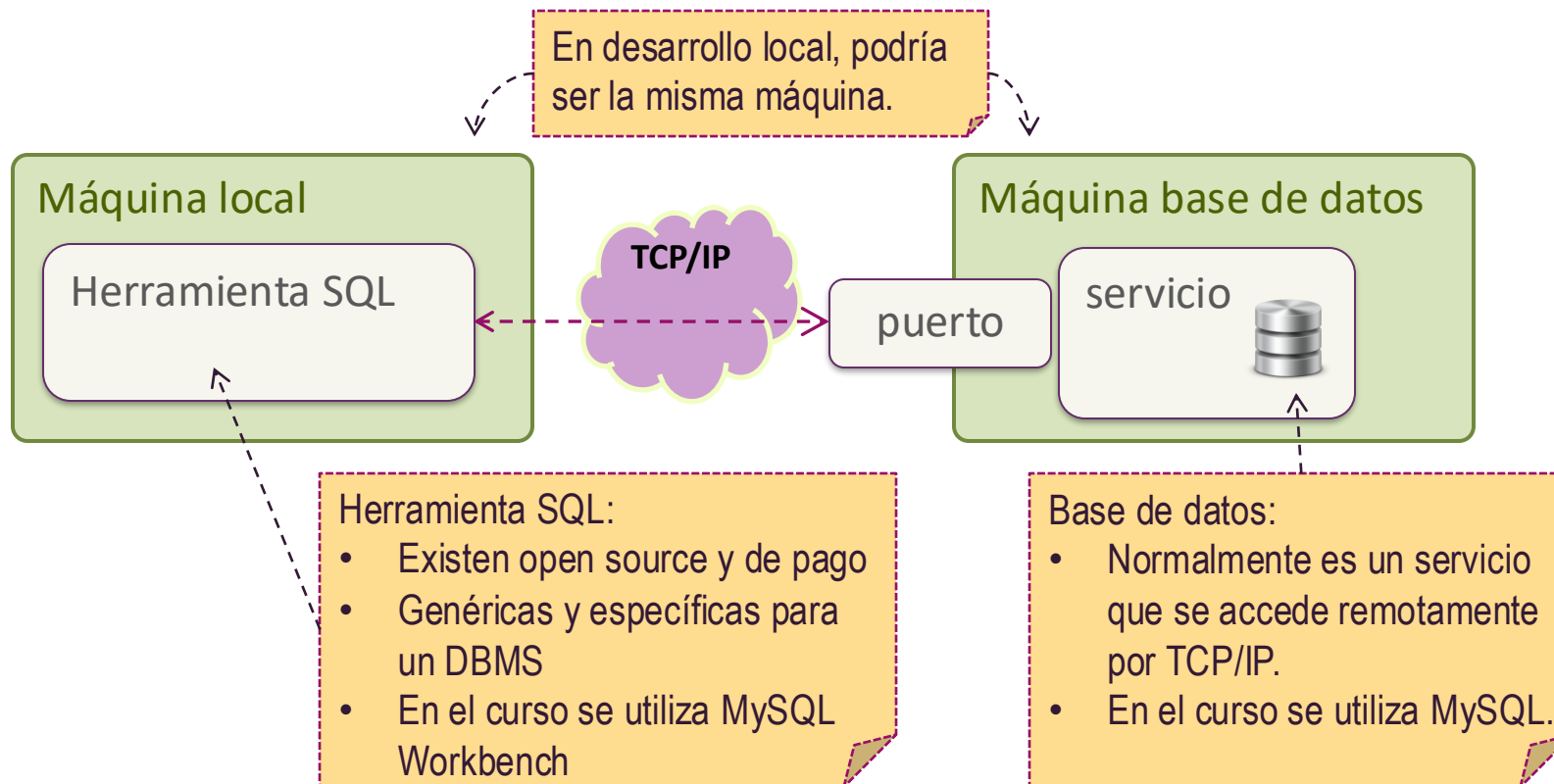


Conceptos Básicos

- lenguaje SQL
 - Para manejar la estructura, los datos y los permisos de una base de datos relacional, se utiliza un lenguaje declarativo llamado **SQL** (Structured Query Language).
 - Nació en los años 70, siendo precedido por el lenguaje SEQUEL de IBM, e incorporado por Oracle por primera vez en un producto.
 - En 1986 se publica por primera vez como un estándar, siendo revisado principalmente en 1992.
 - Ha evolucionado continuamente, y en la actualidad es el lenguaje estándar de los DBMS.
 - En este curso, se describe en forma básica la utilización del lenguaje SQL, para el manejo de estructura (Data Definition Language, o DDL) y la consulta y manipulación de datos (Data Manipulation Language, o DML). El manejo de usuarios y permisos se aborda en cursos posteriores.

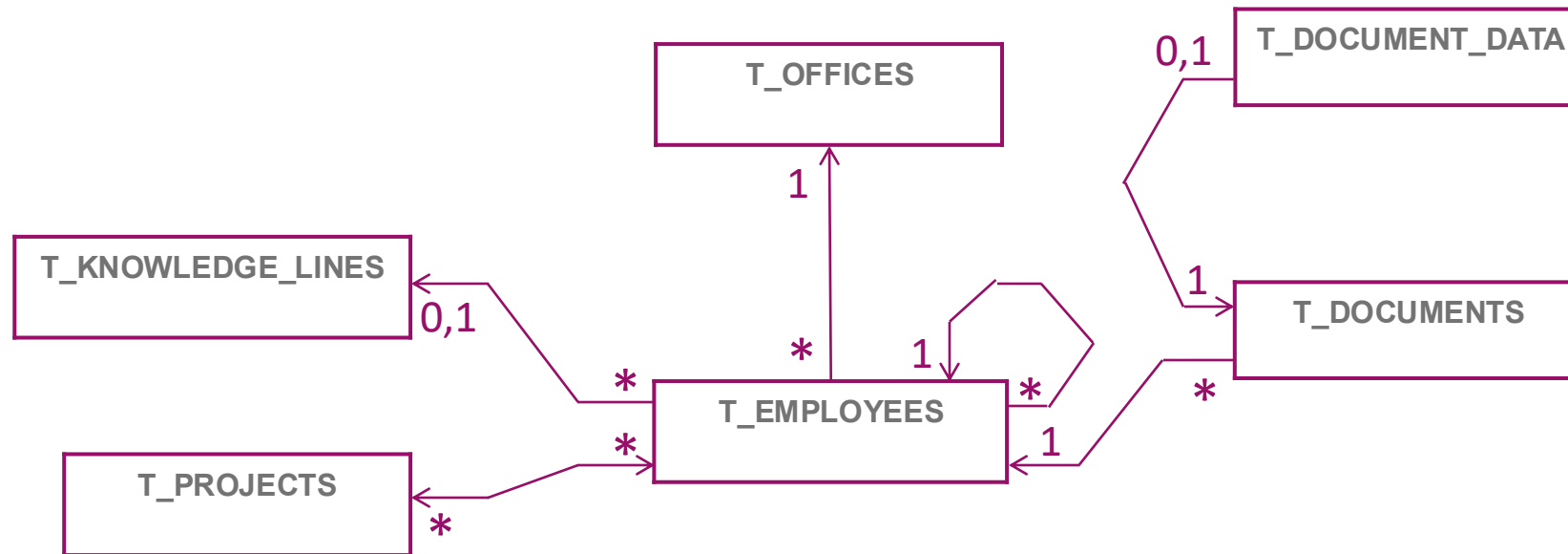
Conceptos Básicos

- herramientas de programación SQL
 - Cuando se utiliza una base de datos desde una herramienta de programación SQL, la herramienta funciona en forma independiente de la base de datos:



Conceptos Básicos

- modelo del caso de negocio



Manejo básico de estructura

- DDL
 - En SQL, el **DDL** (Data Definition Language) incluye sentencias para la creación, modificación y eliminación de la estructura de los elementos de la base de datos. Incluye sentencias del tipo CREATE, ALTER, DROP y TRUNCATE.
 - Los elementos que se pueden manejar dependen del DBMS utilizado. El más relevante es la tabla, cuya estructura puede ser manejada con las sentencias anteriores.

Manejo básico de estructura

- creación base de datos
 - La creación de una base de datos relacional tiene muchas opciones, y depende del DBMS utilizado.
 - En MySQL, se utiliza el siguiente comando SQL:

CREATE DATABASE *nombre*;

- SQL es un lenguaje *case-insensitive*, es decir, no distingue mayúsculas de minúsculas. En este curso, se utilizan mayúsculas para todo, tanto sentencias SQL como nombres de elementos.
- Una vez creada la base de datos, para utilizarla desde línea de comandos se utiliza:

USE *nombre*;

Manejo básico de estructura

- creación de tabla
 - Para crear una tabla, una versión básica de la sentencia SQL es:

```
CREATE TABLE nombre_tabla (  
  nombre1 tipo1 NOT NULL,  
  nombre2 tipo2 NOT NULL,  
  nombre3 tipo3,  
  ...  
)
```

Los saltos de línea no afectan.

Una declaración por campo.

Si el campo debe tener valor, se marca como NOT_NULL. En caso contrario, no se indica.

Manejo básico de estructura

- creación de tabla
 - Por ejemplo, en el modelo de referencia, para crear la tabla de oficinas, se utiliza:

```
CREATE TABLE `T_OFFICES` (  
  `OFFC_ID` INT NOT NULL,  
  `OFFC_COUNTRY` VARCHAR(30) NOT NULL,  
  `OFFC_CITY` VARCHAR(40) NOT NULL,  
  `OFFC_DESCRIPTION` VARCHAR(100)  
);
```

Nombre de tabla
a crear

En MySQL se utilizan **opcionalmente**
`delimitadores` para los nombres,
para evitar problemas de caracteres.

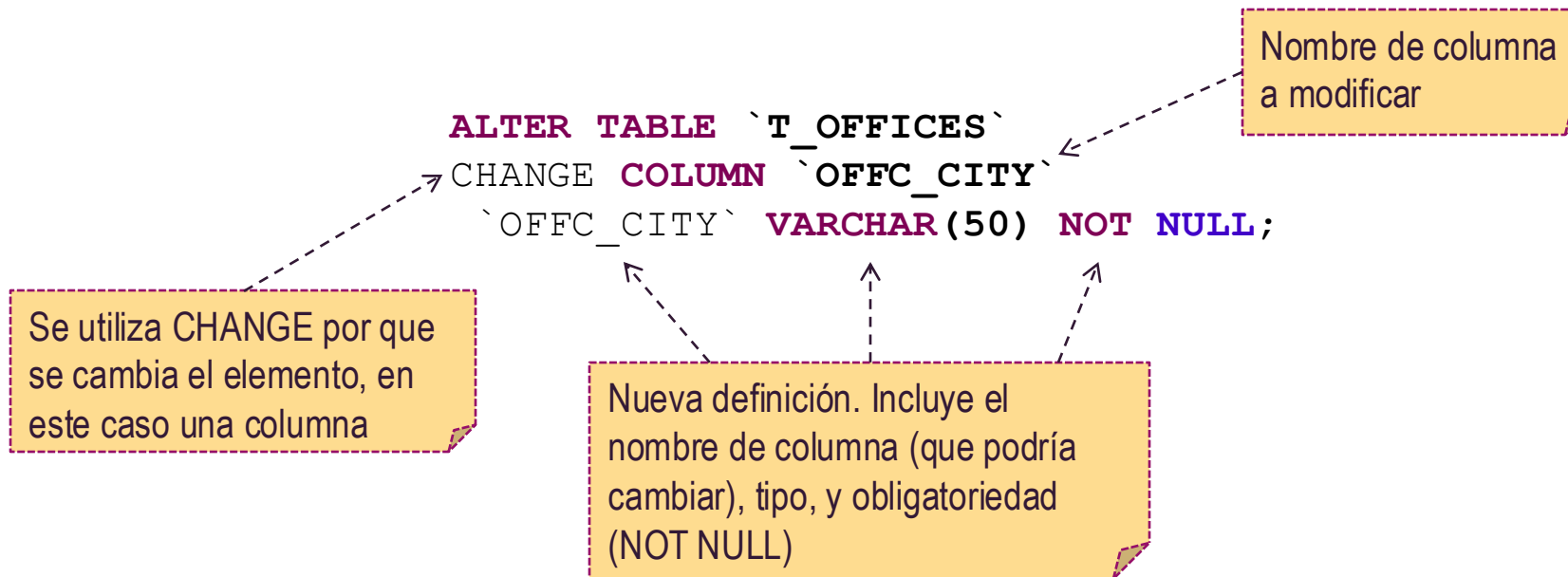
Campo llamado OFFC_ID, de
tipo entero, obligatorio.

Campo llamado OFFC_CITY, de tipo
texto, largo máximo 40, obligatorio.

Campo llamado OFFC_DESCRIPTION,
de tipo texto, largo máximo 100, opcional.

Manejo básico de estructura

- modificación de tabla
 - Para modificar la estructura de una tabla, se utiliza ALTER TABLE. Por ejemplo, para cambiar el largo del campo OFFC_CITY de 40 a 50, se utiliza:



Manejo básico de estructura

- modificación de tabla
 - Ejemplo donde se elimina un campo:

```
ALTER TABLE `T_OFFICES`  
DROP COLUMN `OFFC_CITY`;
```

- Ejemplo donde se agrega un nuevo campo:

```
ALTER TABLE `T_OFFICES`  
ADD COLUMN  
`OFFC_CITY` VARCHAR(50) NOT NULL  
AFTER `OFFC_ID`;
```


Definición de columna
a agregar

Opcional: ubicación
de columna a agregar

Manejo básico de estructura

- eliminación de tabla
 - Para eliminar una tabla, se utiliza la sentencia DROP TABLE:

```
DROP TABLE `T_OFFICES` ;
```



Ejercicio 1: Creación de base de datos y tablas

- Resumen del ejercicio:
 - Utilizar herramienta de manejo de base de datos MySQL.
 - Crear base de datos.
 - Crear una tabla de acuerdo a una definición de estructura.
 - Realizar modificaciones sobre la estructura.



Operaciones sobre datos

- DML
 - En SQL, el **DML** (Data Manipulation Language) incluye sentencias para la lectura, creación, modificación y eliminación de datos de la base de datos. Incluye sentencias del tipo INSERT, SELECT, UPDATE y DELETE, principalmente sobre tablas.

Operaciones básicas sobre datos

- inserción

- Para insertar una nueva fila en una tabla existente, se utiliza la sentencia **INSERT**. Ejemplo para la tabla `T_OFFICES`:

```
INSERT INTO `T_OFFICES`  
(  
    OFFC_ID,  
    OFFC_COUNTRY,  
    OFFC_CITY,  
    OFFC_DESCRIPTION  
)  
VALUES  
(  
    10,  
    'España',  
    'Madrid',  
    'Oficina central'  
) ;
```

Diagram illustrating the SQL INSERT statement and its result:

- Tabla**: Points to the table name `T_OFFICES``.
- Campos**: Points to the list of fields `OFFC_ID, OFFC_COUNTRY, OFFC_CITY, OFFC_DESCRIPTION`.
- Datos. Los textos van entre comillas simples.**: Points to the values `'España', 'Madrid', 'Oficina central'`.
- Nuevo registro**: Points to the new row added to the table.

OFFC ID	OFFC COUNTRY	OFFC CITY	OFFC DESCRIPTION
10	España	Madrid	Oficina central

Operaciones básicas sobre datos

- inserción múltiple
 - Algunos DBMS, como MySQL, soportan inserción de múltiples registros en una sola sentencia. Ejemplo para la tabla `T_OFFICES`:

```
INSERT INTO `T_OFFICES`  
(  
  OFFC_ID, OFFC_COUNTRY, OFFC_CITY, OFFC_DESCRIPTION  
)  
VALUES  
(  
  20, 'Chile', 'Santiago', 'Principal de Chile'  
) , ←----- Grupos de valores separados por coma  
(  
  30, 'Argentina', 'Buenos Aires', NULL ←----- Valor NULL  
) ;
```



	OFFC ID	OFFC_COUNTRY	OFFC_CITY	OFFC_DESCRIPTION
▶	10	España	Madrid	Oficina central
	20	Chile	Santiago	Principal de Chile
	30	Argentina	Buenos Aires	NULL

Operaciones básicas sobre datos

- inserción de campos null
 - Si un campo permite valores NULL, puede omitirse en la sentencia de inserción:

```
INSERT INTO T_OFFICES
(
  OFFC_ID, OFFC_COUNTRY, OFFC_CITY
)
VALUES
(
  11, 'España', 'Barcelona'
),
(
  12, 'España', 'Valladolid'
);
```



	OFFC ID	OFFC COUNTRY	OFFC CITY	OFFC DESCRIPTION
▶	10	España	Madrid	Oficina central
	11	España	Barcelona	NULL
	12	España	Valladolid	NULL
	20	Chile	Santiago	Principal de Chile
	30	Argentina	Buenos Aires	NULL

Operaciones básicas sobre datos

- selección básica
 - Para seleccionar filas de una tabla existente, se utiliza la sentencia **SELECT**, con distintas opciones. Por ejemplo:
 - Para seleccionar todos los campos y registros:

SELECT * FROM T_OFFICES;

Todos los campos

Tabla



	OFFC ID	OFFC COUNTRY	OFFC CITY	OFFC DESCRIPTION
▶	10	España	Madrid	Oficina central
	11	España	Barcelona	NULL
	12	España	Valladolid	NULL
	20	Chile	Santiago	Principal de Chile
	30	Argentina	Buenos Aires	NULL

Operaciones básicas sobre datos

- selección básica
 - Para seleccionar algunas columnas, y todos los registros:

```
SELECT OFFC_COUNTRY, OFFC_CITY  
FROM T_OFFICES;
```

Campos, separados por coma

Tabla



	OFFC_COUNTRY	OFFC_CITY
▶	España	Madrid
	España	Barcelona
	España	Valladolid
	Chile	Santiago
	Argentina	Buenos Aires

Operaciones básicas sobre datos

- selección básica
 - Para seleccionar algunas columnas, y filtrar los registros por criterios:

```
SELECT OFFC_COUNTRY, OFFC_CITY  
FROM T_OFFICES  
WHERE OFFC_COUNTRY = 'España';
```

Condiciones

	OFFC_COUNTRY	OFFC_CITY
▶	España	Madrid
	España	Barcelona
	España	Valladolid

```
SELECT OFFC_CITY, OFFC_DESCRIPTION  
FROM T_OFFICES  
WHERE OFFC_COUNTRY = 'España'  
AND OFFC_DESCRIPTION LIKE 'Oficina%';
```

Equivale a "Comenzar con".
Con LIKE, '%' significa 0 o
más caracteres cualquiera.


	OFFC_CITY	OFFC_DESCRIPTION
▶	Madrid	Oficina central

Operaciones básicas sobre datos

- selección básica
 - Filtro con criterio de valores múltiples:

```
SELECT OFFC_COUNTRY, OFFC_CITY  
FROM T_OFFICES  
WHERE OFFC_CITY IN ('Madrid', 'Barcelona');
```

Se utiliza IN para múltiples valores. No recomendable si son demasiados.



	OFFC_COUNTRY	OFFC_CITY
▶	España	Madrid
	España	Barcelona

Operaciones básicas sobre datos

- selección básica
 - Filtro con criterio de valor NULL:

```
SELECT OFFC_COUNTRY, OFFC_CITY  
FROM T_OFFICES  
WHERE OFFC_DESCRIPTION IS NULL;
```

Nótese que se utiliza 'IS NULL',
y no '= NULL'.



	OFFC_COUNTRY	OFFC_CITY
▶	España	Barcelona
	Argentina	Buenos Aires

- Y de valor NOT NULL, que entrega los otros registros:

```
SELECT OFFC_COUNTRY, OFFC_CITY  
FROM T_OFFICES  
WHERE OFFC_DESCRIPTION IS NOT NULL;
```

Operaciones básicas sobre datos

- actualización
 - Para actualizar filas de una tabla existente, se utiliza la sentencia **UPDATE**, utilizando una condición para seleccionar el o los registros a actualizar.
 - Se quieren actualizar los campos OFFC_CITY y OFFC_DESCRIPTION del siguiente registro, obtenido con una SELECT:

```
SELECT * FROM T_OFFICES WHERE OFFC_ID = 12;
```

	OFFC ID	OFFC COUNTRY	OFFC CITY	OFFC DESCRIPTION
▶	12	España	Valladolid	NULL

Campos a actualizar

Operaciones básicas sobre datos

- actualización

- Actualización:

```
UPDATE T_OFFICES  
SET
```

```
OFFC_CITY = 'Valladolid',  
OFFC_DESCRIPTION = 'Colabora en j-everis'
```

```
WHERE OFFC_ID = 12;
```

Tabla

Nuevos valores,
separados por coma

Condición, normalmente sobre identificador.

	OFFC ID	OFFC COUNTRY	OFFC CITY	OFFC DESCRIPTION
▶	12	España	Valladolid	NULL



	OFFC ID	OFFC COUNTRY	OFFC CITY	OFFC DESCRIPTION
▶	12	España	Valladolid	Colabora en j-everis

Nuevo valor igual a anterior

Operaciones básicas sobre datos


- eliminación
 - Para eliminar filas de una tabla existente, se utiliza la sentencia **DELETE**, utilizando una condición para seleccionar el o los registros a eliminar.

```
DELETE
FROM T_OFFICES
WHERE OFFC_ID = 30;
```

Tabla

Condición, normalmente sobre identificador.

	OFFC ID	OFFC COUNTRY	OFFC CITY	OFFC DESCRIPTION
▶	10	España	Madrid	Oficina central
	11	España	Barcelona	NULL
	12	España	Valladolid	Colabora en j-everis
	20	Chile	Santiago	Principal de Chile
	30	Argentina	Buenos Ai...	NULL



	OFFC ID	OFFC COUNTRY	OFFC CITY	OFFC DESCRIPTION
▶	10	España	Madrid	Oficina central
	11	España	Barcelona	NULL
	12	España	Valladolid	Colabora en j-everis
	20	Chile	Santiago	Principal de Chile



Ejercicio 2: Lectura y escritura básica

- Resumen del ejercicio:
 - Insertar, modificar y eliminar registros de tablas, utilizando sentencias SQL.
 - Seleccionar registros de una tabla dados criterios de búsqueda.

Bases de datos relacionales

- normalización
 - Las bases de datos relacionales están **normalizadas**. Para analizar la normalización, se coloca el siguiente ejemplo, con una tabla con datos de personas que tiene algunas deficiencias:

NOMBRE	APELLIDO	TELEFONO
Juan	Pérez	1234
Luis	González	5432, 6789
Pedro	García	2345

No tiene identificador.
Podrían haber filas repetidas.

El campo contiene
más de un valor

Bases de datos relacionales

- normalización
 - Agregando un identificador (**clave primaria**):

ID_PERSONA	NOMBRE	APELLIDO	TELEFONO
1	Juan	Pérez	1234
2	Luis	González	5432, 6789
3	Pedro	García	2345

La clave primaria
permite identificar cada
registro en forma única.

Bases de datos relacionales

- normalización
 - Solución del dato múltiple: crear una tabla nueva para los teléfonos, que incluya una relación con la tabla de personas.

ID_PERSONA	NOMBRE	APELLIDO
1	Juan	Pérez
2	Luis	González
3	Pedro	García

Con esto, se logra la **primera forma normal**.

ID_PERSONA	TELEFONO
1	1234
2	5432
2	6789
3	2345

Este campo relaciona la tabla de teléfonos con la de personas.

Bases de datos relacionales

- normalización
 - Existen otros casos con problemas de normalización, que se resuelven con las llamadas segunda y tercera forma normal.
 - Si se tiene la siguiente tabla, se pueden presentar problemas:

NOMBRE	APELLIDO	SKILL	CATEGORIA
Juan	Pérez	Java	Programador
Juan	Pérez	SQL	Programador
Juan	Pérez	HTML	Programador
Luis	González	Gestión de equipos	Analista
Luis	González	Gestión de equipos	Analista
Pedro	García	e-mail server	Sistemas
Pedro	García	Linux admin	Sistemas

Si la categoría del empleado cambia, se deben modificar varios registros.

Solución: separar en dos tablas, una de categorías y otra de skills. Esto es la llamada **segunda forma normal**.

Bases de datos relacionales

- normalización
 - Si se tiene la siguiente tabla, también se pueden presentar problemas:

NOMBRE	APELLIDO	CATEGORIA	FECHA CAMBIO
Juan	Pérez	Programador T1	01/09/2008
Juan	Pérez	Programador T2	01/03/2009
Juan	Pérez	Analista T1	01/09/2009
Luis	González	Analista T1	01/09/2008
Luis	González	Analista T2	01/09/2009

La fecha de cambio de categoría está vinculada a la categoría, que depende del nombre y apellido. Es decir, la fecha de cambio depende de un atributo no primario.

Solución: separar en dos tablas, una de categorías y otra de cambios, que incluye una referencia a la categoría y la fecha. Esto es la llamada **tercera forma normal**.

Clave primaria

- definición
 - La **clave primaria** (primary key) es un campo o combinación de campos que identifica en forma única a un registro.
 - Existen claves primarias **naturales**: número de identificación de personas, matrículas, teléfonos.



- La opción recomendada es utilizar una **clave numérica generada**. En la mayoría de los DBMS existen los campos autonuméricos, que se incrementan automáticamente con cada nuevo registro insertado.
- Nota:** en el caso particular de **Oracle**, existen elementos llamados **secuencias** (SEQUENCE), que generan los valores autoincrementales que luego son utilizados en la inserción.

Clave primaria

- autonumérico y secuencia
 - Varios DBMS tienen la capacidad de generar **claves primarias autonuméricas**:
 - MySQL
 - SQL Server
 - Otras
 - En el caso de Oracle, se utilizan objetos especiales llamados secuencias para la generación de una clave primaria.
 - ¿Cuándo utilizar claves autogeneradas? En las tablas donde se agregan nuevos registros, normalmente la mayoría.
 - ¿Cuándo no utilizar claves autogeneradas? En las tablas constantes de sólo lectura o escritura muy esporádica, tipo catálogo. Por ejemplo, la tabla de provincias. En ese caso es mejor asignarla directamente.

Clave primaria

- creación
 - En particular, en MySQL la creación de una clave autonumérica se realiza especificando en la clave el atributo `AUTO_INCREMENT`. Por ejemplo, la creación de la tabla de empleados es:

```
CREATE TABLE `T_EMPLOYEES` (  
  `EMPL_ID` INT NOT NULL AUTO_INCREMENT,  
  `OFFC_ID` INT NOT NULL,  
  `EMPL_FORNAME` VARCHAR(50) NOT NULL,  
  `EMPL_MIDDLE_NAME` VARCHAR(50),  
  `EMPL_SURNAME` VARCHAR(50) NOT NULL,  
  `EMPL_NUMBER` INT NOT NULL,  
  `EMPL_HIRE_DATE` DATETIME NOT NULL,  
  `EMPL_MENTOR_ID` INT,  
  PRIMARY KEY (`EMPL_ID`));
```

- Con esto, cuando se realiza una sentencia `INSERT`, se **omite** el campo de la clave primaria, ya que el propio DBMS lo agrega internamente.



Clave primaria

- consulta por clave primaria
 - La clave permite identificar a un registro de manera única, en un modelo de datos normalizado. Por ejemplo, se puede hacer una SELECT por la clave primaria:

```
SELECT * FROM T_OFFICES  
WHERE OFFC_ID = 12;
```

- Internamente, la base de datos tiene una forma optimizada de acceder al registro dado su clave primaria.



Ejercicio 3: Uso de clave primaria

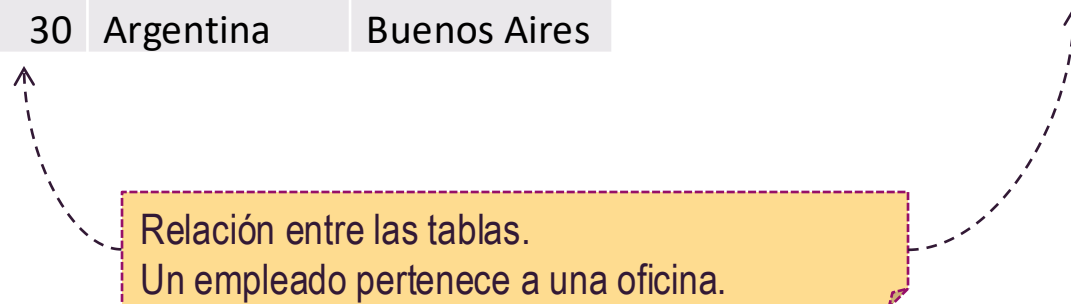
- Resumen del ejercicio:
 - Crear una tabla con clave primaria autogenerada.
 - Insertar registros utilizando clave primaria autogenerada.

Estructuración relacional

- concepto de relación
 - Una **relación** vincula a dos tablas de una base de datos relacional normalizada. Por ejemplo:

OFFC_ID	COUNTRY	CITY
10	España	Madrid
11	España	Barcelona
20	Chile	Santiago
30	Argentina	Buenos Aires

EMPL_ID	OFFC_ID	FORNAME	SURNAME
150	10	Juan	Pérez
160	11	Luis	González
180	20	Pedro	García



Estructuración relacional

- clave foránea
 - El campo de una tabla que referencia a la clave primaria de una tabla relacionada es una **clave foránea** (foreign key o FK).

OFFC_ID	COUNTRY	CITY
10	España	Madrid
11	España	Barcelona
20	Chile	Santiago
30	Argentina	Buenos Aires

EMPL_ID	OFFC_ID	FORNAME	SURNAME
150	10	Juan	Pérez
160	11	Luis	González
180	20	Pedro	García

Clave foránea.

- Más adelante, en el punto de índices y constraints, se describe cómo crear una relación entre una clave foránea y una primaria.

Estructuración relacional

- relación many-to-one
 - **many-to-one** es una relación de muchos a uno. Normalmente es desde una tabla que tiene una clave foránea que apunta a una clave primaria.



OFFC_ID	COUNTRY	CITY	EMPL_ID	OFFC_ID	FORNAME	SURNAME
10	España	Madrid	150	10	Juan	Pérez
11	España	Barcelona	160	11	Luis	González
20	Chile	Santiago	180	20	Pedro	García
30	Argentina	Buenos Aires	8080	20	Erick	Johnson
			12300	30	Mariano	Gómez

Estructuración relacional

- integridad referencial
 - Se refiere a no permitir modificar o eliminar una relación que genera inconsistencias. Por ejemplo:
 - Eliminar un registro cuya clave primaria es referenciada por alguna clave foránea.
 - Modificar el valor de una clave foránea, por un nuevo valor que no existe en la clave primaria.

OFFC_ID	COUNTRY	CITY
10	España	Madrid
11	España	Barcelona
20	Chile	Santiago

No permite eliminar registro, ya que es referenciado desde T_EMPLOYEES

EMPL_ID	OFFC_ID	FORNAME	SURNAME
150	10	Juan	Pérez
160	11	Luis	González
180	20	Pedro	García

No permite modificar por un valor que no es PK de T_OFFICES

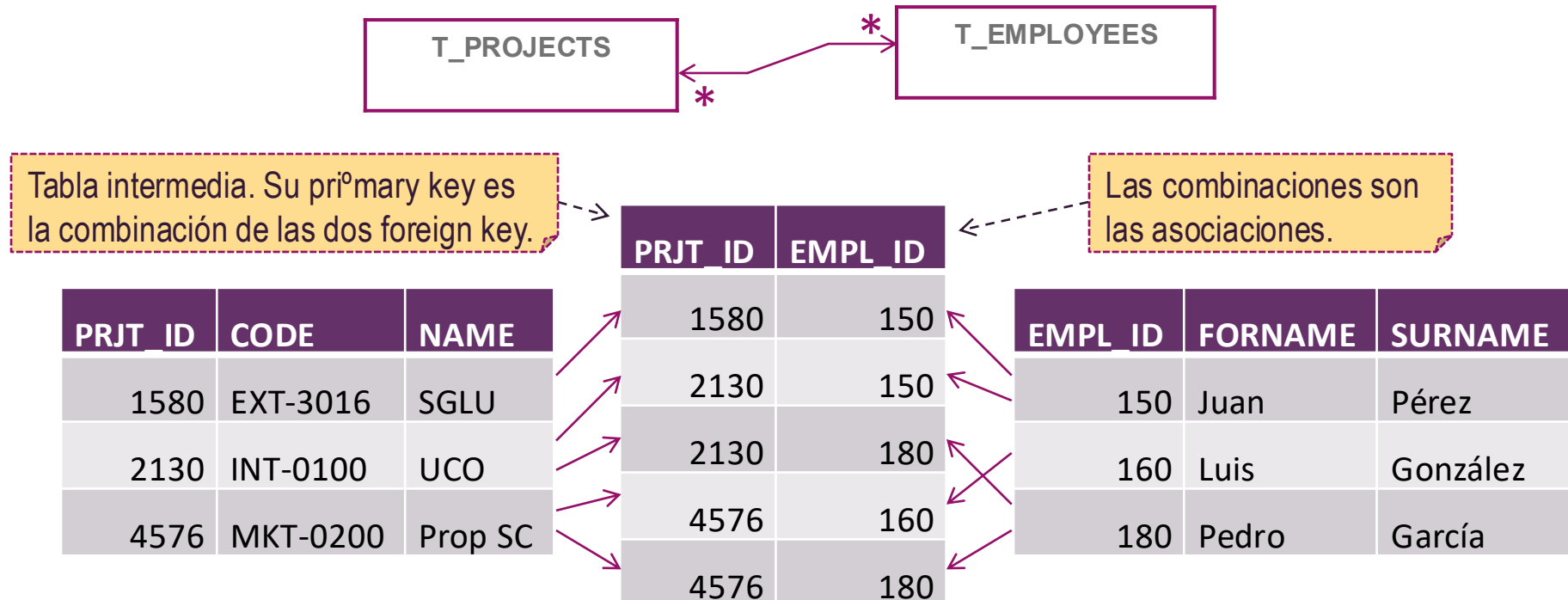


Ejercicio 4: Uso de relaciones

- Resumen del ejercicio:
 - Utilizar una tabla con clave primaria autogenerada.
 - Insertar registros basado en la relación entre tablas.

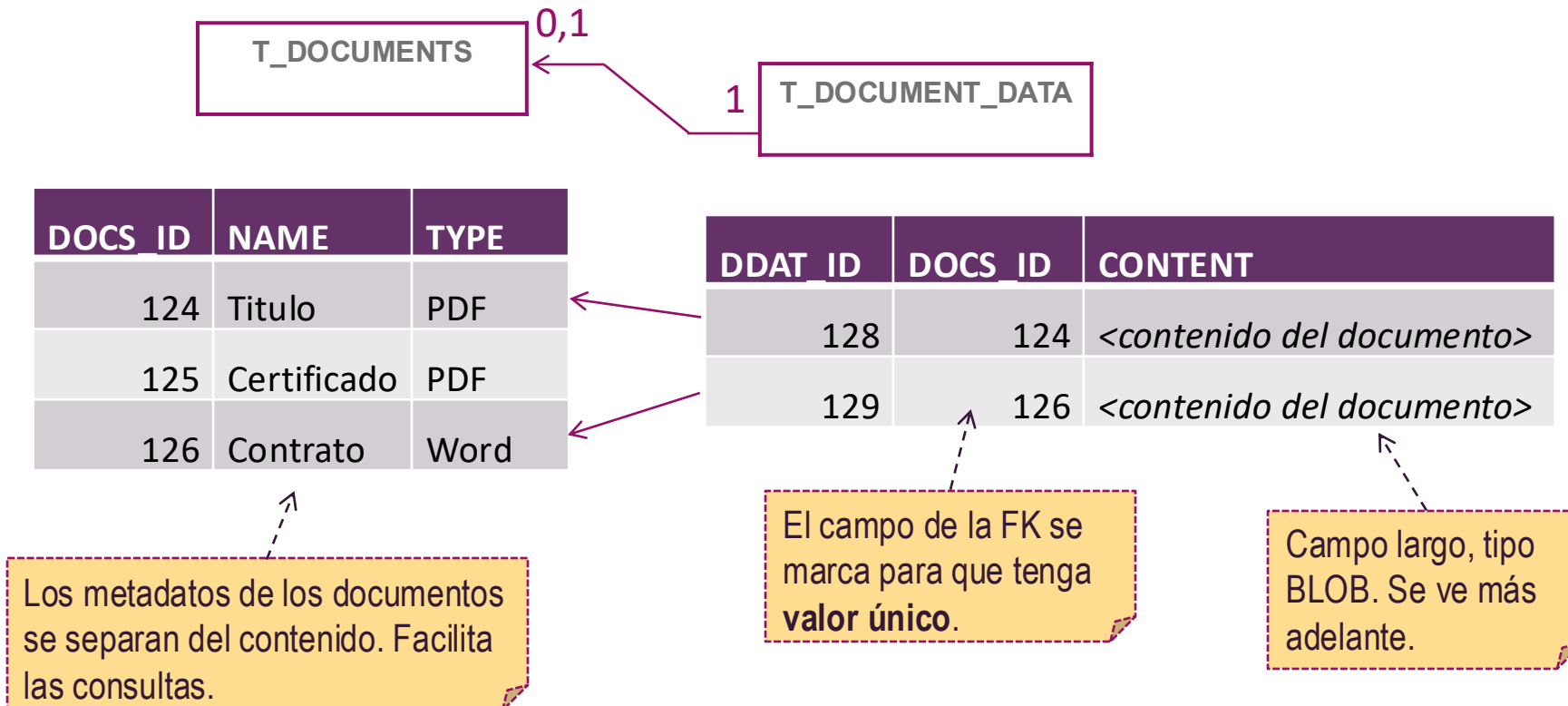
Otras relaciones

- relación many-to-many
 - many-to-many** es una relación del tipo muchos a muchos. Se resuelve utilizando una **tabla de relación intermedia**, que tiene una clave primaria formada por la combinación de las claves foráneas a las tablas relacionadas.



Otras relaciones

- relación one-to-one
 - **one-to-one** es cuando una tabla tiene a lo más un registro relacionado en otra tabla. Se utiliza como complemento a la tabla principal.



Otras relaciones

- auto-referencia tipo many-to-one
 - Cuando la clave foránea referencia a la clave primaria de la misma tabla, es una auto-referencia. Este tipo de relaciones se utilizan para modelar una jerarquía.



EMPL_ID	FORNAME	SURNAME	MENTOR_ID
150	Juan	Pérez	
160	Luis	González	150
180	Pedro	García	150

Es FK de la PK de la propia tabla.

Nota: Como el elemento raíz de la jerarquía no puede tener un elemento relacionado, y por integridad referencial no puede ser el mismo, entonces la clave foránea no puede ser not null, a no ser que se cambia a dicha condición después de colocado el primer dato.



Otras relaciones

- auto-referencia tipo many-to-many
 - También se pueden construir autoferencias del tipo many-to-many. En ese caso, se utiliza también una tabla de relación, en la que las dos claves foráneas que forman la clave primaria apuntan a la misma tabla.

Ejercicio 5: Tipos de relaciones

- Resumen del ejercicio:
 - Crear la tabla de proyectos.
 - Crear la tabla de relación intermedia entre proyectos y empleados.
 - Añadir datos de proyectos y de asociaciones entre proyecto y empleado.

Claves compuestas

- clave primaria compuesta
 - Una clave primaria es preferentemente simple y autogenerada. Sin embargo, en modelos, normalmente antiguos, se pueden encontrar claves primarias compuestas.
 - Una **clave primaria compuesta** está formada por la combinación de dos o más campos, en la que la combinación de valores es única y puede identificar al registro.

Claves compuestas

- clave foránea compuesta
 - Cuando desde una tabla se referencia a otra tabla que tiene una clave primaria compuesta, entonces la clave foránea es también compuesta, formada por los mismos campos.
 - Esta complejidad adicional es una de las razones por las cuales se recomienda **no utilizar claves compuestas**.

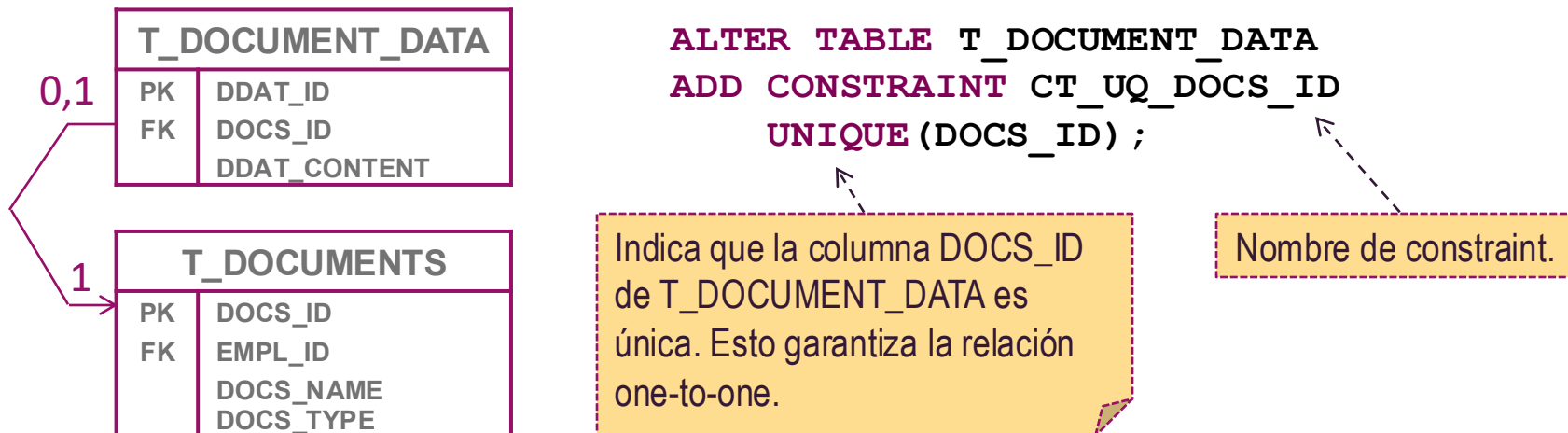


Claves compuestas

- utilización y recomendaciones
 - En un modelo nuevo, **no hay razones** que justifiquen la utilización de claves compuestas.
 - En los puntos anteriores se observa que el manejo de claves compuestas es más complejo que las simples, sobre todo cuando participan en relaciones.
 - Por lo tanto, se recomienda **no utilizar claves compuestas**.
 - Si se necesita que la combinación de dos o más campos sea única, siempre se puede agregar una clave autogenerada como primaria, y aplicar a dichos campos una constraint unique, concepto que se ve más adelante.

Constraint

- definición
 - Una **constraint** es una restricción que se aplica a algún elemento, que facilita la integridad de los datos. Las hay de distintos tipos:
 - Constraint unique: garantiza la unicidad del valor de un campo en sus registros. Se utiliza, por ejemplo, en las claves primarias en forma implícita, y también se puede aplicar a otros campos. Por ejemplo, en las relaciones tipo one-to-one, se utiliza una constraint unique para la FK, que asegura que sólo un registro está relacionado:



Constraint

- definición
 - Constraint check:** restringe los valores de un campo a un conjunto predefinido, lo que complementa las restricciones asociadas al tipo. Por ejemplo, si se quiere restringir el campo DOCS_TYPE de la tabla T_DOCUMENTS, la sentencia es la siguiente:

```
ALTER TABLE T_DOCUMENTS
ADD CONSTRAINT CT_CK_DOCS_TYPE
CHECK (DOCS_TYPE IN ('PDF', 'DOC', 'XLS'));
```

Si se intenta colocar un valor distinto a los de la lista del CHECK en el campo DOCS_TYPE, se produce un error.

Indice

- definición
 - Cuando se utiliza el valor de un campo como criterio de búsqueda, una forma de evitar que se busquen los registros que cumplen con su valor uno a uno es utilizando un **índice** (index). Por ejemplo, en las antiguas guías telefónicas:



Como los apellidos están ordenados alfabéticamente, se puede buscar relativamente rápido un número dados el apellido y el nombre.

En cambio, si se quiere buscar un teléfono sin el nombre, habría que comenzar desde el principio, y verlos uno a uno, ya que no están ordenados por número. Eso es una tarea muchísimo más larga e ineficiente.

Indice

- creación
 - Para crear un índice, se utiliza una sentencia SQL de tipo ALTER TABLE, ya que se modifica la tabla asociada, seguida de un ADD INDEX. Por ejemplo, si se quiere mejorar el rendimiento de la búsqueda de empleados dada la fecha de contratación (HIRE_DATE), se puede crear el siguiente índice:

```
ALTER TABLE T_EMPLOYEES  
ADD INDEX IX_HIRE_DATE (EMPL_HIRE_DATE) ;
```

- En el caso que no exista en índice, en una consulta que utiliza el campo como condición, el DBMS realiza una búsqueda registro a registro, lo que se conoce como un **full scan**. Si la tabla tiene muchos registros, esto puede ser un proceso lento e ineficiente.
- **Nota:** se recomienda utilizar siempre índices en las claves foráneas.
- El otro extremo, de agregar demasiados índices, afecta el rendimiento al momento de crear o modificar datos, ya que se deben actualizar los índices. Por lo tanto, el óptimo es una solución intermedia, que depende de varios factores.

Indice

- búsquedas tipo case-insensitive
 - En algunos DBMS no se permite la búsqueda del tipo case-insensitive, es decir, ignorando mayúsculas y minúsculas. En ese caso, la forma de buscar es pasando a minúsculas tanto el campo como el valor buscado. Por ejemplo, si se hace una búsqueda de ese tipo en el campo OFFC_DESCRIPTION de un OFFICE, resulta:

```
SELECT OFFC_CITY, OFFC_DESCRIPTION  
FROM T_OFFICES  
WHERE LOWER(OFFC_DESCRIPTION) LIKE LOWER('Algún texto');
```
 - Para evitar que suceda un *full scan* de la tabla, ya que el LOWER del campo (o cualquier función) no está indexada aunque el campo lo esté, se debe crear un índice asociado a dicha función.
 - Notas: Los índices sobre funciones no están permitidos en MySQL. Se utiliza normalmente en **Oracle**. En otras bases de datos como PostgreSQL, existe el ILIKE, que compara directamente ignorando mayúsculas y minúsculas.

Claves foráneas

- creación
 - Hasta el momento, se han definido los conceptos de clave foránea e integridad referencial, pero no se ha especificado cómo crear la relación con la clave primaria.
 - La creación de la relación FK – PK **depende del DMBS utilizado**. En **MySQL**, se hace creando un **índice** y luego una **constraint**. Por ejemplo, la relación entre T_EMPLOYEES y T_OFFICES, a través de los campos OFFC_ID (mismo nombre para FK y PK) se define de la siguiente manera:

```
ALTER TABLE T_EMPLOYEES
  ADD INDEX FK_EMPL_OFFC (OFFC_ID) ,
  ADD CONSTRAINT FK_EMPL_OFFC
    FOREIGN KEY (OFFC_ID)
    REFERENCES T_OFFICES (OFFC_ID) ;
```

Índice sobre la FK.

Constraint con el mismo nombre que el índice.

Índice que es una constraint de una FK y el campo de la FK.

Tabla referenciada por la FK, y nombre de su PK.

Claves foráneas

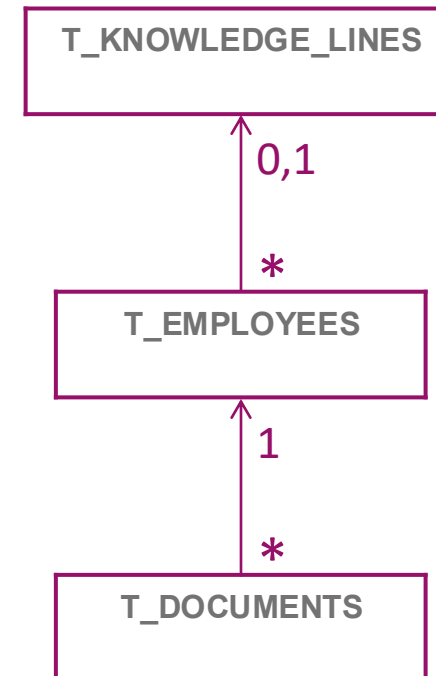
- creación

- Siguiendo la misma lógica, la relación entre las tablas T_EMPLOYEES y T_KNOWLEDGE_LINES es:

```
ALTER TABLE T_EMPLOYEES
  ADD INDEX FK_EMPL_KNLN (KNLN_ID),
  ADD CONSTRAINT FK_EMPL_KNLN
    FOREIGN KEY (KNLN_ID)
    REFERENCES T_KNOWLEDGE_LINES (KNLN_ID);
```

- Y entre T_DOCUMENTS y T_EMPLOYEES :

```
ALTER TABLE T_DOCUMENTS
  ADD INDEX FK_DOCS_EMPL (EMPL_ID),
  ADD CONSTRAINT FK_DOCS_EMPL
    FOREIGN KEY (EMPL_ID)
    REFERENCES T_EMPLOYEES (EMPL_ID);
```





Ejercicio 6: Creación de claves foráneas

- Resumen del ejercicio:
 - Crear las claves foráneas especificadas en el manual para tablas del modelo.
 - Comprobar la integridad referencial.
 - Crear las claves foráneas de una relación many-to-many.

Joins

- definición
 - Un **join** junta en una consulta el resultado del cruce de dos o más tablas. Utiliza la relación entre ellas para construir el resultado. Existen dos tipos principales de join:
 - Inner join
 - Left outer join
 - Los join normalmente utilizan las relaciones FK – PK para cruzar las tablas.
 - Además de estos, existen otros tipos de joins menos utilizados.

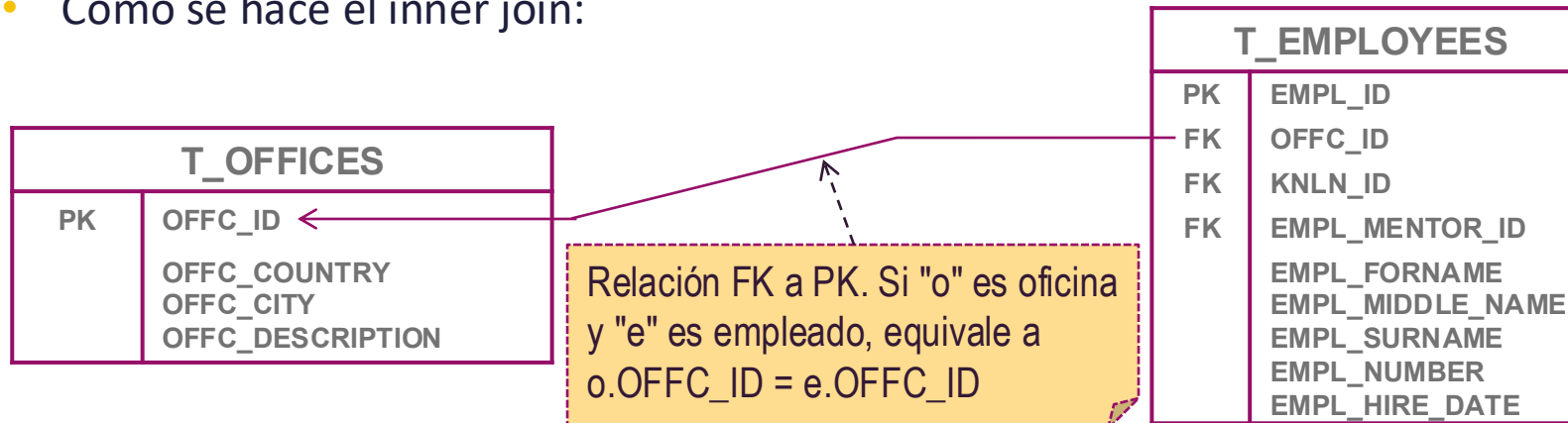
Joins

- inner join
 - Aplicación de **inner join**, que muestra el nombre de los empleados y la oficina a la que pertenecen:
 - Datos (se omiten prefijos y algunos campos):

OFFC_ID	COUNTRY	CITY
10	España	Madrid
11	España	Barcelona
20	Chile	Santiago

EMPL_ID	OFFC_ID	FORNAME	SURNAME
150	10	Juan	Pérez
160	11	Luis	González
180	20	Pedro	García
8080	20	Erick	Johnson

- Cómo se hace el inner join:



Joins

- inner join

- Aplicando el inner join sin otras condiciones aparte de la del join:
- Consulta SQL:

```
SELECT e.EMPL_FORNAME, e.EMPL_SURNAME, o.OFFC_COUNTRY, o.OFFC_CITY
FROM
  T_EMPLOYEES e
  INNER JOIN T_OFFICES o
    ON e.OFFC_ID = o.OFFC_ID;
```

Especifica inner join

Como se combinan tablas, se utilizan alias para referenciarlas

En el "ON" se especifica la relación FK a PK.

- Resultado (se omiten prefijos):

FORNAME	SURNAME	COUNTRY	CITY
Juan	Pérez	España	Madrid
Luis	González	España	Barcelona
Pedro	García	Chile	Santiago
Erick	Johnson	Chile	Santiago

Joins

- inner join

- A un inner join se le pueden agregar condiciones utilizando cualquiera de las tablas involucradas. Por ejemplo, el nombre de la ciudad de la oficina:

- Consulta SQL:

```
SELECT e.EMPL_FORNAME, e.EMPL_SURNAME, o.OFFC_COUNTRY, o.OFFC_CITY  
FROM T_EMPLOYEES e INNER JOIN T_OFFICES o ON e.OFFC_ID = o.OFFC_ID  
WHERE o.OFFC_CITY = 'Santiago';
```

Las condiciones se aplican con el nombre del campo, incluyendo el alias.

- Resultado (se omiten prefijos):

FORNAME	SURNAME	COUNTRY	CITY
Pedro	García	Chile	Santiago
Erick	Johnson	Chile	Santiago

Joins

- inner join, theta-style
 - Un inner join se puede aplicar también colocando las condiciones del join en el where. Esto se conoce como el *theta-style*. Para el ejemplo anterior:
 - Consulta SQL:

```
SELECT e.EMPL_FORNAME, e.EMPL_SURNAME, o.OFFC_COUNTRY, o.OFFC_CITY
FROM
    T_EMPLOYEES e, T_OFFICES o
WHERE e.OFFC_ID = o.OFFC_ID
AND o.OFFC_CITY = 'Santiago';
```

En el from, tablas separadas por coma.

La condición del join se coloca en el where, junto al resto. Nótese que podría no utilizarse un FK-PK, sino cualquier otra que sea coherente.

- El resultado (se omiten prefijos) es el mismo:

FORNAME	SURNAME	COUNTRY	CITY
Pedro	García	Chile	Santiago
Erick	Johnson	Chile	Santiago

Joins

- left outer join
 - Ejemplo de **left outer join**, que muestra el nombre de los empleados y la línea de conocimientos asociada, que es **opcional**:
 - Datos (se omiten prefijos y algunos campos):

KNLN_ID	NAME
10	Java
20	.NET
30	Mainframe

EMPL_ID	KNLN_ID	FORNAME	SURNAME
150	10	Juan	Pérez
160	20	Luis	González
180	<NULL>	Pedro	García
8080	10	Erick	Johnson

- Cómo se hace el left outer join:

T_KNOWLEDGE_LINES	
PK	KNLN_ID
	KNLN_NAME

Relación FK a PK. Equivale a
 $k.KNLN_ID = e.KNLN_ID$,
incluyendo $e.KNLN_ID$ IS NULL

T_EMPLOYEES	
PK	EMPL_ID
FK	OFFC_ID
FK	KNLN_ID
FK	EMPL_MENTOR_ID
	EMPL_FORNAME
	EMPL_MIDDLE_NAME
	EMPL_SURNAME
	EMPL_NUMBER
	EMPL_HIRE_DATE

Joins

- left outer join
 - Aplicando el left outer join, se observan los empleados y sus líneas de conocimiento, y también los que no la tienen:
 - Consulta SQL:

```
SELECT e.EMPL_FORNAME, e.EMPL_SURNAME, k.KNLN_NAME
FROM T_EMPLOYEES e
LEFT OUTER JOIN T_KNOWLEDGE_LINES k
ON e.KNLN_ID = k.KNLN_ID;
```

Especifica left outer join

En el "ON" se especifica la relación FK a PK.

- Resultado (se omiten algunos prefijos):

FORNAME	SURNAME	KNLN_NAME
Juan	Pérez	Java
Luis	González	.NET
Pedro	García	<NULL>
Erick	Johnson	Java

Nótese que también obtiene los empleados que no tienen línea de conocimiento

Joins

- left outer join
 - Para aclarar la diferencia, se compara el resultado con inner join:
 - Consulta SQL:

```
SELECT e.EMPL_FORNAME, e.EMPL_SURNAME, k.KNLN_NAME
FROM T_EMPLOYEES e
      INNER JOIN T_KNOWLEDGE_LINES k
              ON e.KNLN_ID = k.KNLN_ID;
```

- Resultado (se omiten algunos prefijos):

FORNAME	SURNAME	KNLN_NAME
Juan	Pérez	Java
Luis	González	.NET
Erick	Johnson	Java

En este caso, si no tiene línea de conocimiento, no obtiene el resultado. Es decir, el inner join es más restrictivo.

Joins

- left outer join
 - A un left outer join se pueden agregar condiciones utilizando cualquiera de las tablas, incluyendo las de las relaciones que pueden ser null. Por ejemplo, el nombre de la línea de conocimientos:
 - Consulta SQL:

```
SELECT e.EMPL_FORNAME, e.EMPL_SURNAME, k.KNLN_NAME
FROM T_EMPLOYEES e
     LEFT OUTER JOIN T_KNOWLEDGE_LINES k
                   ON e.KNLN_ID = k.KNLN_ID
WHERE k.KNLN_NAME = 'Java';
```

Las condiciones se aplican con el nombre del campo, incluyendo el alias.

- Resultado (se omiten prefijos):

FORNAME	SURNAME	KNLN_NAME
Juan	Pérez	Java
Erick	Johnson	Java

El resultado no incluye el empleado sin línea de conocimiento, porque no cumple la condición.

Joins

- left outer join
 - Si la condición es sobre la tabla principal, entonces no omite las relaciones NULL:
 - Consulta SQL:

```
SELECT e.EMPL_FORNAME, e.EMPL_SURNAME, k.KNLN_NAME
FROM T_EMPLOYEES e
     LEFT OUTER JOIN T_KNOWLEDGE_LINES k
                   ON e.KNLN_ID = k.KNLN_ID
WHERE e.OFFC_ID = 20; <-----
```

Condición sobre la
tabla principal.

- Resultado (se omiten prefijos):

FORNAME	SURNAME	KNLN_NAME
Pedro	García	<NULL>
Erick	Johnson	Java

El resultado incluye el empleado
sin línea de conocimiento, porque
cumple la condición sobre la tabla
principal.

Ejercicio 7: Uso de join

- Resumen del ejercicio:
 - Realizar consultas con inner join y left outer join.
 - Se utilizan relaciones tipo many-to-one y many-to-many.

Vista

- definición
 - Una vista de base de datos es una consulta accesible, equivalente en la práctica a una tabla, pues contiene registros y campos. La diferencia está en que no contiene datos, sino que es resultado de una consulta. Permiten abstraer el resultado de consultas intermedias.
 - Por ejemplo, se puede crear una vista con el resultado de la consulta de empleados y la oficina a la que pertenecen:

```
CREATE VIEW v_employee_office
(EMPL_ID, EMPL_FORNAME, EMPL_SURNAME, OFFC_COUNTRY, OFFC_CITY)
AS
SELECT
    e.EMPL_ID, e.EMPL_FORNAME, e.EMPL_SURNAME,
    o.OFFC_COUNTRY, o.OFFC_CITY
FROM T_EMPLOYEES e
    INNER JOIN T_OFFICES o
        ON e.OFFC_ID = o.OFFC_ID;
```

Vista

- definición

- A la vista anterior se le pueden hacer consultas, como si fuera una tabla:

```
SELECT EMPL_FORNAME, EMPL_SURNAME, OFFC_CITY  
FROM v_employee_office  
WHERE OFFC_CITY = 'Madrid'
```

- Resultado:

EMPL_FORNAME	EMPL_SURNAME	OFFC_CITY
Juan	Pérez	Madrid

- Se observa que con la vista se abstrae de la complejidad de la consulta asociada, pudiendo obtener información con sentencias más simples.



Ejercicio 8: Uso de vistas

- Resumen del ejercicio:
 - Crear una vista dada la definición de su consulta asociada.
 - Realizar consultas sobre la vista.

Esquema

- definición
 - Hasta el momento, se ha utilizado sólo una agrupación de tablas y elementos asociados (vistas, índices, constraints, ...). Una base de datos normalmente contiene varias agrupaciones, separadas por distintos criterios, como:
 - Módulos funcionales
 - Aplicaciones
 - Permisos y roles
 - Estas agrupaciones se llaman normalmente **esquemas** (**schema**), aunque el nombre exacto varía según el DBMS. En este caso, el schema es "db_empl".
 - Así, por ejemplo, dentro de una base de datos se pueden almacenar distintos modelos de datos, en distintos esquemas con sus respectivos usuarios y permisos, y todos conviven. Dependiendo de los permisos, incluso se pueden llamar unos a otros, utilizando el nombres del esquema junto al elemento.

Otras operaciones sobre datos

- tipos de datos básicos
 - Los tipos soportados por las bases de datos varían levemente de nombre. Entre los más comunes están:
 - VARCHAR(largo): campo de texto de largo variable. En Oracle, existe VARCHAR2. El largo es en bytes, y utiliza el juego de caracteres configurado.
 - CHAR(largo): campo de texto de largo fijo. El resto se rellena con espacios. Recomendable sólo para textos pequeños de largo fijo, principalmente flags.
 - Número(largo, precisión): campo numérico con un largo (incluye parte decimal) y precisión (número de decimales). Por ejemplo, el número 123,45 tiene largo 5 y precisión 2. El nombre varía según el DBMS, y puede ser NUMBER, NUMERIC o DECIMAL. Para los enteros existe INT, INTEGER, BIGINT y TINYINT, entre otros.
 - Fecha: campo que representa una fecha, y en algunos casos puede incluir hora. Puede llamarse DATE, DATETIME o TIMESTAMP.

Otras operaciones sobre datos

- tipos de dato largos
 - La mayoría de los DBMS soportan tipos de datos largos, conocidos como LOB (large object), para almacenar cantidades de información de un largo mayor que el máximo soportado por los campos. Existen:
 - Binarios: almacena cadenas de bytes. Los nombres son:
 - BLOB, en Oracle y MySQL, el nombre más conocido.
 - VARBINARY(MAX), en SQL Server
 - Texto: almacena cadenas de caracteres. Los nombres son:
 - CLOB, en Oracle.
 - TEXT, en MySQL.
 - VARCHAR(MAX), en SQL Server.
 - El manejo de este tipo de objetos desde SQL es complejo, especialmente los binarios. Desde las tecnologías de base de datos se puede simplificar, lo que se ve en los cursos de persistencia.

Otras operaciones sobre datos

- casos especiales de selección
 - Para realizar una **concatenación** de campos en una consulta, la nomenclatura cambia según el DBMS utilizado:
 - En Oracle, se coloca entre los campos la expresión ||, y también existe la función CONCAT.
 - En MySQL, se utiliza CONCAT.
 - En SQL Server, se coloca entre los campos un +.
 - Por ejemplo, para obtener el nombre y apellido de los empleados, en un solo campo con el alias 'FULL_NAME', junto a la PK, en MySQL es:

```
SELECT EMPL_ID, CONCAT(EMPL_FORNAME, ' ', EMPL_SURNAME) FULL_NAME  
FROM T_EMPLOYEES;
```

EMPL ID	FULL NAME
1	Juan Pérez
2	Luis González
3	Pedro García
4	Erick Johnson

Alias del campo

Otras operaciones sobre datos

- casos especiales de selección
 - Cuando se necesita colocar comillas simples (o apóstrofes) en el contenido de un texto, se colocan dos seguidas. Por ejemplo, si se quiere agregar al empleado John O'Donnel, la sentencia es:

```
INSERT INTO `T_EMPLOYEES`  
(`OFFC_ID`, `KNLN_ID`,  
`EMPL_FORNAME`, `EMPL_MIDDLE_NAME`,  
`EMPL_SURNAME`,  
`EMPL_NUMBER`, `EMPL_HIRE_DATE`)  
VALUES  
(10, 10,  
'John', NULL, 'O' 'Donnel',  
210, '2003-04-15');
```

Dos apóstrofes seguidos
equivalen a uno

Otras operaciones sobre datos

- conteo
 - Cuando se requiere contar el número de registros que cumplen una condición, y no obtenerlos, se utiliza la sentencia SELECT COUNT.
 - Por ejemplo, para saber cuántas oficinas hay en España en la base de datos de prueba, se hace la consulta:

```
SELECT COUNT (*)  
FROM T_OFFICES  
WHERE OFFC_COUNTRY = 'España'
```

COUNT(*)
3

- Con esto se evita la mala práctica de obtener los registros para posteriormente sólo contarlos.

Otras operaciones sobre datos

- registros distintos
 - En el caso que existan registros iguales, es decir, que todos los campos resultantes de la consulta son los mismos, se pueden excluir los repetidos colocando DISTINCT después de SELECT.
 - Por ejemplo, si se quieren saber los países y ciudades a las que pertenecen las oficinas de los empleados, la consulta es:

```
SELECT o.OFFC_COUNTRY, o.OFFC_CITY
FROM T_EMPLOYEES e
      INNER JOIN T_OFFICES o
            ON e.OFFC_ID = o.OFFC_ID;
```

- Utilizando DISTINCT:

```
SELECT DISTINCT o.OFFC_COUNTRY, o.OFFC_CITY
FROM T_EMPLOYEES e
      INNER JOIN T_OFFICES o
            ON e.OFFC_ID = o.OFFC_ID;
```

COUNTRY	CITY
España	Madrid
España	Madrid
España	Barcelona
Chile	Santiago
Chile	Santiago

COUNTRY	CITY
España	Madrid
España	Barcelona
Chile	Santiago

Otras operaciones sobre datos

- funciones matemáticas
 - Para obtener valores como el máximo, mínimo, suma o promedio de los valores de una columna de tipo numérico, SQL provee las funciones **MAX**, **MIN**, **SUM** y **AVG** respectivamente.
 - Por ejemplo, para obtener el menor número de empleado, se utiliza:

```
SELECT MIN(EMPL_NUMBER) MIN_NUM_EMPL  
FROM T_EMPLOYEES;
```

Alias del campo

MIN_NUM_EMPL
150

- Para los campos de texto o fecha, las funciones MAX y MIN se aplican de acuerdo a su orden, ya sea alfabético o en el tiempo.

Otras operaciones sobre datos

- agrupación
 - Las funciones matemáticas y el conteo se han aplicado sobre tablas completas. Si se quisiera saber, por ejemplo, cuántos empleados hay en cada oficina, es necesario que el count sea sobre agrupaciones por la FK de la oficina. Para realizar agrupaciones, se utiliza la sentencia **GROUP BY**:

```
SELECT COUNT (EMPL_ID) AS CNT_EMPL, OFFC_ID
FROM T_EMPLOYEES
GROUP BY OFFC_ID
```

CNT_EMPL	OFFC_ID
2	10
1	1
2	20

count de empleados por cada grupo, es decir, oficina.

Define que la agrupación es por oficinas

Si se coloca un campo en la parte select, tiene que ser agrupado o común al grupo.

Si se hiciera un inner join con la tabla de oficinas, se podría obtener el número y el país y ciudad.

Otras operaciones sobre datos

- subselección
 - Si se quiere hacer una consulta donde un campo está en un conjunto de valores que es el resultado de otra consulta, se puede utilizar un **IN**. Por ejemplo, si se quiere obtener los nombres y apellidos de los empleados que pertenecen a las oficinas de España, una alternativa a hacer un inner join es:

```
SELECT EMPL_FORNAME, EMPL_SURNAME
FROM T_EMPLOYEES
WHERE OFFC_ID IN (
    SELECT OFFC_ID FROM T_OFFICES WHERE OFFC_COUNTRY = 'España')
```

Campo IN significa que el valor está en el conjunto resultante de la SELECT

La SELECT es sobre el campo correspondiente de la tabla de oficinas

FORNAME	SURNAME
Juan	Pérez
Luis	González
John	O'Donnel

Otras operaciones sobre datos

- ordenación

- Los registros en las tablas no se almacenan necesariamente en el mismo orden que se insertan, sino en una forma conveniente manejada internamente por el DBMS. Esto implica que los resultados de las consultas no tienen un orden predefinido. Por lo tanto, para ordenar los registros resultantes de una consulta es necesario especificarlo, a través de una sentencia **ORDER BY**.
- Por ejemplo, para obtener los empleados ordenados por apellido ascendente, junto a información de la oficina, se utiliza:

```
SELECT e.EMPL_FORNAME, e.EMPL_SURNAME, o.OFFC_COUNTRY, o.OFFC_CITY  
FROM T_EMPLOYEES e INNER JOIN T_OFFICES o ON e.OFFC_ID = o.OFFC_ID  
ORDER BY e.EMPL_SURNAME; <
```

FORNAME	SURNAME	COUNTRY	CITY
Pedro	García	Chile	Santiago
Luis	González	España	Barcelona
Erick	Johnson	Chile	Santiago
John	O'Donnel	España	Madrid
Juan	Pérez	España	Madrid

Nombre del campo. El ascendente es default.

Otras operaciones sobre datos

- ordenación
 - Si se quieren los empleados ordenados por fecha de contratación descendente, es decir, desde el más nuevo al más antiguo, y en caso de ser iguales, por apellido ascendente, la consulta es:

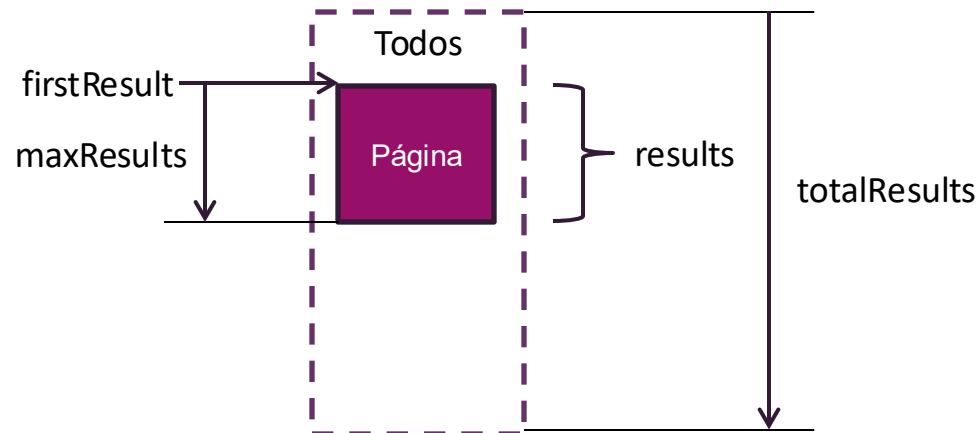
```
SELECT e.EMPL_FORNAME, e.EMPL_SURNAME, e.EMPL_HIRE_DATE  
FROM T_EMPLOYEES e  
ORDER BY e.EMPL_HIRE_DATE DESC, e.EMPL_SURNAME ASC;
```

FORNAME	SURNAME	HIRE DATE
Pedro	García	2006-05-18
Luis	González	2006-05-18
Juan	Pérez	2005-04-15
John	O'Donnel	2003-04-15
Erick	Johnson	2000-02-18

Se pueden colocar varios campos. A igualdad del primero, aplica el segundo.

Otras operaciones sobre datos

- paginación
 - La **paginación** se refiere a obtener los datos de una consulta de muchos resultados por bloques, evitando tener que obtenerlos todos a la vez. La solución específica depende del DBMS utilizado:
 - En MySQL, se utiliza LIMIT al final de la consulta.
 - En Oracle, se controla con ROWNUM.
 - En SQL Server, se puede controlar con TOP, aunque sólo la cantidad máxima. Desde la versión 2005 se puede utilizar ROW_NUMBER(), aunque es más complejo.



Otras operaciones sobre datos

- paginación

- Por ejemplo, si de la consulta de los empleados ordenados por fecha de contratación, se quieren obtener desde el segundo registro los siguientes tres, la sentencia es:

```
SELECT e.EMPL_FORNAME, e.EMPL_SURNAME, e.EMPL_HIRE_DATE
FROM T_EMPLOYEES e
ORDER BY e.EMPL_HIRE_DATE DESC, e.EMPL_SURNAME ASC LIMIT 1, 3;
```

El segundo registro es el 1 (comienza en 0).

El último registro es el cuarto (número 3).

FORNAME	SURNAME	HIRE DATE
Pedro	García	2006-05-18
Luis	González	2006-05-18
Juan	Pérez	2005-04-15
John	O'Donnel	2003-04-15
Erick	Johnson	2000-02-18

1
3



FORNAME	SURNAME	HIRE DATE
Luis	González	2006-05-18
Juan	Pérez	2005-04-15
John	O'Donnel	2003-04-15

Sin paginación

Tecnofor
by SNGULAR



¡Gracias!

Plaza de la Independencia, 8
28001-Madrid

www.tecnofor.es