Bases de datos relacionales y SQL

Jesús Arias Fisteus

Aplicaciones Web (2018/19)

uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

Parte I

Introducción a las bases de datos relacionales

Bases de datos relacionales

- Colección de datos almacenados en una o más tablas.
- Las tablas constan de filas y columnas.
- Las tablas pueden estar relacionadas entre sí.

Ejemplo: una única tabla

Tabla "Paises"

nombre	continente	superficie	poblacion	capital
España	Europa	505370	46438422	Madrid
Francia	Europa	643801	64590000	París
Canadá	América	9984670	36155487	Ottawa
Alemania	Europa	357022	81770900	Berlín
Australia	Oceanía	7692024	23613193	Canberra

Ejemplo: varias tablas relacionadas

Tabla "Paises"

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Tabla "Continentes"

id	nombre	superficie	poblacion
1	Europa	10180000	731000000
2	América	42330000	910000000
3	Oceanía	9008458	38889988

Ejemplo: varias tablas relacionadas

Tabla "Ciudades"

id	nombre	superficie	poblacion
1	Madrid	605.77	3141991
2	Barcelona	102.15	1604555
3	París	105.40	2229621
4	Ottawa	2778.64	1083391
5	Nueva York	1214.00	8491079
6	Berlín	891.68	3469849
7	Canberra	814.20	381488

Relaciones entre tablas

- Las relaciones entre tablas se explicitan por medio de:
  - Clave primaria: columna o combinación de columnas que identifican unívocamente a las filas de una tabla.
  - Clave ajena (también conocida como clave externa o foránea): columna o combinación de columnas en una tabla que hacen referencia a la clave primaria de otra tabla.

Sistema gestor de bases de datos relacionales

- Programa que da soporte al uso de bases de datos relacionales.
- Ejemplos:
  - Oracle Database
  - Microsoft SQL Server
  - IBM DB2
  - SAP Advantage Database Server
  - MySQL
  - PostgreSQL
  - SQLite

El lenguaje SQL

- Lenguaje estándar para utilizar y mantener bases de datos relacionales.
- Utilizado en los principales gestores de bases de datos relacionales.
  - Aunque hay pequeñas variaciones (*dialectos*) dependiendo del gestor.

Parte II

El lenguaje SQL: recuperación de datos

La sentencia SELECT

- La sentencia SELECT se utiliza para recuperar datos de la base de datos.

```
1 SELECT
2 <columna_1>, <columna_2>, <columna_3>
3 FROM <tabla>
4 WHERE <condición>
```

La sentencia SELECT: ejemplos

```
1 SELECT *
2 FROM Paises;
```

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

La sentencia SELECT: ejemplos

```
1 SELECT nombre, poblacion
2 FROM Paises;
```

nombre	poblacion
España	46438422
Francia	64590000
Canadá	36155487
Alemania	81770900
Australia	23613193

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

La sentencia SELECT: ejemplos

```
1 SELECT superficie, poblacion
2 FROM Paises
3 WHERE nombre='España';
```

superficie	poblacion
505370	46438422

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

La sentencia SELECT: ejemplos

```
1 SELECT superficie, poblacion
2 FROM Paises
3 WHERE id=1;
```

superficie	poblacion
505370	46438422

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Lógica Booleana

- Operadores Booleanos: AND, OR, NOT.
- Expresiones de comparación: $=$, $<>$, $<$, $>$, $<=$, $>=$, BETWEEN.
- Pertenencia a un conjunto de valores: IN.
- Comparación con NULL: IS NULL, IS NOT NULL.

La sentencia SELECT: ejemplos

```
1 SELECT nombre, poblacion
2 FROM Paises
3 WHERE poblacion>50000000;
```

nombre	poblacion
Francia	64590000
Alemania	81770900

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

La sentencia SELECT: ejemplos

```
1 SELECT nombre, poblacion
2 FROM Paises
3 WHERE poblacion<50000000 AND continente=1;
```

nombre	poblacion
España	46438422

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

La sentencia SELECT: ejemplos

```
1 SELECT nombre, poblacion
2 FROM Paises
3 WHERE poblacion BETWEEN 50000000 AND 70000000;
```

nombre	poblacion
Francia	64590000

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

La sentencia SELECT: ejemplos

```sql
SELECT nombre
FROM Paises
WHERE continente IN (1, 3);
```

nombre
España
Francia
Alemania
Australia

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Ordenación y restricción del número de resultados

- Ordenación:
  - `ORDER BY` especifica la columna (o columnas) sobre la cual se deben ordenar las filas.
  - Sentido de ordenación: `ASC` (por defecto), `DESC`.
- Restricción del número de resultados:
  - Palabra clave `LIMIT`.
  - Palabra clave `OFFSET` para seleccionar el rango.

La sentencia SELECT: ejemplos

```
1 SELECT nombre, poblacion
2 FROM Paises
3 ORDER BY poblacion;
```

nombre	poblacion
Australia	23613193
Canadá	36155487
España	46438422
Francia	64590000
Alemania	81770900

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

La sentencia SELECT: ejemplos

```
1 SELECT nombre, poblacion
2 FROM Paises
3 WHERE poblacion<40000000
4 ORDER BY poblacion;
```

nombre	poblacion
Australia	23613193
Canadá	36155487

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

La sentencia SELECT: ejemplos

```sql
SELECT nombre, poblacion
FROM Paises
ORDER BY poblacion DESC
LIMIT 2;
```

nombre	poblacion
Alemania	81770900
Francia	64590000

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

La sentencia SELECT: ejemplos

```sql
SELECT nombre, poblacion
FROM Paises
ORDER BY poblacion DESC
LIMIT 2 OFFSET 2;
```

nombre	poblacion
España	46438422
Canadá	36155487

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Ejercicios prácticos

- Práctica 3:
  - Ejercicio 1.1
  - Ejercicio 1.2

Agregación de datos: resultados únicos

- A veces no se desea obtener resultados duplicados en una consulta.

```
1 SELECT continente
2 FROM Paises;
```

continente
1
1
1
2
3

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Agregación de datos: resultados únicos

- Si en los resultados de una consulta aparecen filas duplicadas, el modificador DISTINCT elimina todas excepto una.

```
1 SELECT DISTINCT continente
2 FROM Paises;
```

continente
1
2
3

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Agregación de datos: sumas, medias, etc.

- Cómputo de agregaciones sobre las filas obtenidas: SUM, AVG, MIN, MAX.

```
1 SELECT SUM(superficie)
2 FROM Paises
3 WHERE continente=1;
```

SUM(superficie)
1506193

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Cuenta del número de resultados

- El operador COUNT modifica la consulta de tal forma que devuelva el número de filas seleccionadas.

```
1 SELECT COUNT(*)
2 FROM Paises
3 WHERE continente=1;
```

COUNT(*)
3

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Cuenta del número de resultados

- También es posible contar valores sin tener en cuenta sus repeticiones.

```
1 SELECT COUNT(DISTINCT continente)
2 FROM Paises;
```

COUNT(DISTINCT continente)
3

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Agregación de datos: agrupación de filas

- GROUP BY permite hacer cómputos agregados (suma, media, etc.) sobre grupos de filas.

```
1 SELECT continente, SUM(superficie)
2 FROM Paises
3 GROUP BY continente;
```

continente	SUM(superficie)
1	1506193
2	9984670
3	7692024

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Agregación de datos: agrupación de filas

- HAVING filtra los grupos resultantes.

```
1 SELECT continente, SUM(superficie)
2 FROM Paises
3 GROUP BY continente
4 HAVING SUM(superficie)>5000000;
```

continente	SUM(superficie)
2	9984670
3	7692024

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Agregación de datos: agrupación de filas

- Nótese que con WHERE se seleccionan las filas que serán agrupadas, mientras que con HAVING, los grupos.

```
1 SELECT continente, SUM(superficie)
2 FROM Paises
3 WHERE poblacion<70000000
4 GROUP BY continente
5 HAVING SUM(superficie)<9000000;
```

continente	SUM(superficie)
1	1149171
3	7692024

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Alias de columnas

- Se puede establecer el nombre que tomarán las columnas resultantes de una consulta.

```
1 SELECT
2 continente,
3 SUM(superficie) AS superficie_agregada
4 FROM Paises
5 GROUP BY continente;
```

continente	superficie_agregada
1	1506193
2	9984670
3	7692024

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Columnas calculadas

- Se pueden obtener columnas resultantes de realizar cálculos sobre los valores de otras columnas.

```
1 SELECT
2 nombre,
3 poblacion/superficie AS densidad
4 FROM Paises;
```

nombre	densidad
España	91.88994598017294
Francia	100.3260324230624
Canadá	3.6210998460640162
Alemania	229.0360257911277
Australia	3.0698283052678983

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Ejercicios prácticos

- Práctica 3:
  - Ejercicio 1.3

Parte III

Consultas sobre múltiples tablas

Consultas sobre múltiples tablas

- Es habitual que los datos a consultar estén distribuidos en varias tablas relacionadas entre sí. Por ejemplo:
  - Obtener los pares nombre de país y nombre de capital.
  - Obtener la lista de países cuya capital tenga más de 3 millones de habitantes.

Ejemplo

Tabla "Paises"

id	nombre	continente	superficie	poblacion	capital
1	España	1	505370	46438422	1
2	Francia	1	643801	64590000	3
3	Canadá	2	9984670	36155487	4
4	Alemania	1	357022	81770900	6
5	Australia	3	7692024	23613193	7

Tabla "Ciudades"

id	nombre	superficie	poblacion
1	Madrid	605.77	3141991
2	Barcelona	102.15	1604555
3	París	105.40	2229621
4	Ottawa	2778.64	1083391
5	Nueva York	1214.00	8491079
6	Berlín	891.68	3469849
7	Canberra	814.2	381488

Consultas `INNER JOIN`

- El operador `INNER JOIN` permite unir dos tablas indicando bajo qué criterio se deben emparejar las filas de una tabla con las de la otra:
  - Se forman todos los pares posibles de filas que cumplan el criterio.

Ejemplo

```sql
SELECT *
FROM Paises
INNER JOIN Ciudades
ON capital=Ciudades.id;
```

id	nombre	cont.	sup.	pob.	cap.	id	nombre	sup.	pob.
1	España	1	505370	46438422	1	1	Madrid	605.77	3141991
2	Francia	1	643801	64590000	3	3	París	105.40	2229621
3	Canadá	2	9984670	36155487	4	4	Ottawa	2778.64	1083391
4	Alemania	1	357022	81770900	6	6	Berlín	891.68	3469849
5	Australia	3	7692024	23613193	7	7	Canberra	814.2	381488

Ejemplo

```
1 SELECT
2 Paises.nombre, Ciudades.nombre
3 FROM Paises
4 INNER JOIN Ciudades
5 ON capital=Ciudades.id;
```

nombre	nombre
España	Madrid
Francia	París
Canadá	Ottawa
Alemania	Berlín
Australia	Canberra

Ejemplo

```
1 SELECT
2 Paises.nombre, Ciudades.nombre
3 FROM Paises
4 INNER JOIN Ciudades
5 ON capital=Ciudades.id
6 WHERE Ciudades.poblacion>3000000;
```

nombre	nombre
España	Madrid
Alemania	Berlín

Alias de nombres de tablas

```sql
SELECT
P.nombre, C.nombre
FROM Paises AS P
INNER JOIN Ciudades AS C
ON capital=C.id
WHERE C.poblacion>3000000;
```

nombre	nombre
España	Madrid
Alemania	Berlín

Consultas `INNER JOIN`

- Las filas de cualquiera de las dos tablas que no cumplan el criterio para ningún posible par no aparecerán entre los resultados.
  - Ejemplo: filas de Nueva York y Barcelona.
- Las filas de cualquiera de las dos tablas que cumplan el criterio para varios pares aparecerán varias veces entre los resultados, una por cada par.

Ejemplo

Tabla "Ciudades"

id	nombre	superficie	poblacion
1	Madrid	605.77	3141991
2	Barcelona	102.15	1604555
3	París	105.40	2229621
4	Ottawa	2778.64	1083391
5	Nueva York	1214.00	8491079
6	Berlín	891.68	3469849
7	Canberra	814.2	381488

Tabla "Atracciones"

id	nombre	ciudad
1	Estatua de la Libertad	5
2	Torre Eiffel	3
3	Empire State Building	5
4	Coliseo de Roma	NULL

Ejemplo

```
1 SELECT
2 Atracciones.nombre, Ciudades.nombre
3 FROM Atracciones
4 INNER JOIN Ciudades
5 ON ciudad=Ciudades.id;
```

nombre	nombre
Estatua de la Libertad	Nueva York
Torre Eiffel	París
Empire State Building	Nueva York

Ejercicios prácticos

- Práctica 3:
  - Ejercicio 1.4

Consultas `OUTER JOIN`

- Se recuperan todas las filas de la *tabla primaria* aunque no cumplan el criterio con ninguna fila de la otra tabla.
- Tres tipos de `OUTER JOIN`:
  - `LEFT JOIN`: la tabla primaria es la especificada antes del operador `JOIN` (a la izquierda).
  - `RIGHT JOIN`: la tabla primaria es la especificada tras el operador `JOIN` (a la derecha).
  - `FULL JOIN`: ambas tablas son primarias (no disponible en MySQL).

Ejemplo

```
1 SELECT
2 Atracciones.nombre, Ciudades.nombre
3 FROM Atracciones
4 LEFT JOIN Ciudades
5 ON ciudad=Ciudades.id;
```

nombre	nombre
Estatua de la Libertad	Nueva York
Torre Eiffel	París
Empire State Building	Nueva York
Coliseo de Roma	NULL

Ejemplo

```
1 SELECT
2 Atracciones.nombre, Ciudades.nombre
3 FROM Atracciones
4 RIGHT JOIN Ciudades
5 ON ciudad=Ciudades.id;
```

nombre	nombre
NULL	Madrid
NULL	Barcelona
Torre Eiffel	París
NULL	Ottawa
Estatua de la Libertad	Nueva York
Empire State Building	Nueva York
NULL	Berlín
NULL	Canberra

Ejemplo

```
1 SELECT
2 Atracciones.nombre, Ciudades.nombre
3 FROM Ciudades
4 LEFT JOIN Atracciones
5 ON ciudad=Ciudades.id;
```

nombre	nombre
NULL	Madrid
NULL	Barcelona
Torre Eiffel	París
NULL	Ottawa
Estatua de la Libertad	Nueva York
Empire State Building	Nueva York
NULL	Berlín
NULL	Canberra

Ejemplo (no disponible en MySQL)

```
1 SELECT
2 Atracciones.nombre, Ciudades.nombre
3 FROM Ciudades
4 FULL JOIN Atracciones
5 ON ciudad=Ciudades.id;
```

nombre	nombre
Estatua de la Libertad	Nueva York
Torre Eiffel	París
Empire State Building	Nueva York
NULL	Madrid
NULL	Barcelona
NULL	Ottawa
NULL	Berlín
NULL	Canberra
Coliseo de Roma	NULL

Consultas *self join*

- Es posible unir una tabla consigo misma mediante el uso de alias de nombres de tablas:
  - Se toman dos "copias" de la misma tabla, cada una con un alias que la diferencia de la otra.
  - Es compatible con cualquier tipo de operador `JOIN`.
- Ejemplo:
  - Obtener todos los pares de monumentos que estén en la misma ciudad.

Ejemplo

```sql
SELECT
A.nombre, B.nombre
FROM Atracciones AS A
INNER JOIN Atracciones AS B
ON A.ciudad=B.ciudad;
```

nombre	nombre
Estatua de la Libertad	Estatua de la Libertad
Empire State Building	Estatua de la Libertad
Torre Eiffel	Torre Eiffel
Estatua de la Libertad	Empire State Building
Empire State Building	Empire State Building

Ejemplo

```sql
SELECT
A.nombre, B.nombre
FROM Atracciones AS A
INNER JOIN Atracciones AS B
ON A.ciudad=B.ciudad
WHERE A.id<B.id;
```

nombre	nombre
Estatua de la Libertad	Empire State Building

Parte IV

El lenguaje SQL: inserción y modificación de datos

Creación de tablas

- Se crean tablas con `CREATE TABLE`.
- Al crear una tabla, se especifican aspectos como:
  - Nombre de la tabla.
  - Para cada columna:
    - Nombre y tipo de datos
    - Valor por defecto
    - Si puede tomar valor `NULL`.
    - Si es un campo de auto-incremento.
  - Claves primarias y ajenas.
  - Índices.

Ejemplo

```sql
CREATE TABLE Continentes (
        id INT NOT NULL auto_increment,
        nombre VARCHAR(255) NOT NULL,
        superficie DOUBLE NOT NULL,
        poblacion LONG NOT NULL,
        PRIMARY KEY(id)
);
```

Ejemplo

```sql
CREATE TABLE Ciudades (
        id INT NOT NULL auto_increment,
        nombre VARCHAR(255) NOT NULL,
        superficie DOUBLE NOT NULL,
        poblacion LONG NOT NULL,
        PRIMARY KEY(id)
);
```

Ejemplo

```
1  CREATE TABLE Paises (
2          id INT NOT NULL auto_increment,
3          nombre VARCHAR(255) NOT NULL,
4          continente INT NOT NULL,
5          superficie DOUBLE NOT NULL,
6          poblacion LONG NOT NULL,
7          capital INT NOT NULL,
8          PRIMARY KEY(id),
9          CONSTRAINT FOREIGN KEY (continente)
10                 REFERENCES Continentes(id),
11         CONSTRAINT FOREIGN KEY (capital)
12                 REFERENCES Ciudades(id)
13 );
```

Ejemplo

```sql
CREATE TABLE AreasGeograficas (
        id INT NOT NULL auto_increment,
        nombre VARCHAR(255) NOT NULL,
        tipo ENUM('continente', 'pais',
                  'region', 'ciudad') NOT NULL,
        area DOUBLE NOT NULL,
        habitantes LONG NOT NULL,
        PRIMARY KEY(id)
);
```

Tipos de datos

- Las columnas tienen un tipo de datos asociado.
- Principales grupos de tipos de datos:
  - Numéricos.
  - Cadenas.
  - Fechas / horas.
- Los tipos de datos concretos varían según el gestor de bases de datos.
- Valor especial NULL: ausencia de valor.

Tipos de datos numéricos en MySQL

- Bits: `BIT` (p.e. cuatro bits: `BIT(4)`).
- Enteros con signo (opcional `UNSIGNED`):
  - 1 byte: `TINYINT` / `BOOL` / `BOOLEAN`.
  - 2 bytes: `SMALLINT`.
  - 3 bytes: `MEDIUMINT`.
  - 4 bytes: `INT` / `INTEGER`.
  - 8 bytes: `BIGINT`.
- Punto flotante:
  - 4 bytes (precisión aprox. 7 decimales): `FLOAT`.
  - 8 bytes (precisión aprox. 15 decimales): `DOUBLE`.
- Punto fijo: `DECIMAL` / `NUMERIC`
  - P.e. `DECIMAL(6, 2)` representa 6 dígitos, de los cuales 2 son decimales.

Tipos de datos de cadenas

- Cadenas de caracteres:
  - Tamaño fijo: `CHAR` (p.e. `CHAR(8)`).
  - Tamaño variable: `VARCHAR` (p.e. `VARCHAR(255)`).
  - Tamaño grande: `TEXT`.
- Cadenas de bytes:
  - Tamaño fijo: `BINARY` (p.e. `BINARY(8)`).
  - Tamaño variable: `VARBINARY` (p.e. `VARBINARY(255)`).
  - Tamaño grande: `BLOB`.
- Enumerados: `ENUM`
  - P.e.: `ENUM('profesor', 'alumno', 'administrativo')`.
- Conjuntos: `SET`
  - P.e.: `SET('rojo', 'verde', 'azul')`.

Tipos de datos de fechas y horas en MySQL

- Fecha: `DATE`.
- Fecha y hora (con minutos y segundos, sin zona horaria): `DATETIME`.
- Hora: `TIME`.
- Año: `YEAR` (para año con dos dígitos, `YEAR(2)`).
- Sello temporal (número de segundos desde 1-1-1970: `TIMESTAMP`.
  - Por defecto, la primera columna de tipo `TIMESTAMP` se actualiza automáticamente con `INSERT` y `UPDATE`, salvo que se le asigne un valor o se especifique lo contrario.

Otras acciones de gestión de tablas

- Mostrar todas las tablas de una base de datos: `SHOW TABLES`
- Mostrar la estructura de una tabla: `DESCRIBE <nombre_de_tabla>`
- Eliminar una tabla: `DROP TABLE <nombre_de_tabla>;`
- Añadir, modificar o eliminar columnas en una tabla: `ALTER TABLE ...`

Inserción de filas en tablas

- Se insertan filas nuevas en tablas mediante `INSERT INTO`:
  - Se puede insertar una o más filas en una única sentencia.
  - Se puede especificar qué columnas se proporcionan y en qué orden (por defecto, se deben proporcionar todas y en el mismo orden en que se definieron al crear la tabla).

```
1 INSERT INTO <tabla>
2 (<columna_1>, <columna_2>, <columna_3>)
3 VALUES
4 (<valores_fila_1>),
5 (<valores_fila_2>)
```

Ejemplo

```
1 INSERT INTO Continentes
2 VALUES
3 (4, 'Asia', 44579000.0, 4164252000);
```

Columnas para las que no se proporciona valor

- Se puede omitir el valor de algunas columnas:
  - Columnas autoincrementales:
    - Reciben automáticamente un valor numérico único que se incrementa a medida que se insertan filas en la tabla.
    - Se suelen utilizar como clave primaria.
  - Otros tipos de columnas:
    - Reciben el valor por defecto definido para dicha columna, o `NULL` si no se ha definido dicho valor.

Ejemplo

```
1 INSERT INTO Continentes
2 (nombre, superficie, poblacion)
3 VALUES
4 ('Africa', 30370000, 1100000000),
5 ('Antártida', 14000000, 135);
```

Inserción de los resultados de una consulta

- Se puede insertar filas resultantes de una consulta `SELECT`.

```
1 INSERT INTO Ciudades
2 (nombre, superficie, poblacion)
3 SELECT
4 nombre, area, habitantes
5 FROM AreasGeograficas
6 WHERE tipo='ciudad';
```

Inserción de los resultados de una consulta

- También es posible insertar el resultado de una consulta en una columna concreta.

```
1  INSERT INTO Paises
2  (nombre, capital, continente, superficie,
       poblacion)
3  VALUES
4  ('Italia',
5   (SELECT id
6    FROM Ciudades
7    WHERE nombre='Roma'),
8   (SELECT id
9    FROM Continentes
10   WHERE nombre='Europa'),
11  301340.0,
12  60600000
13 );
```

Ejercicios prácticos

- Práctica 4:
  - Ejercicio 1.1
  - Ejercicio 1.2

Eliminación de filas

- Se eliminan filas en una tabla mediante la sentencia `DELETE`.
- Se indica el nombre de la tabla y la condición que deben cumplir las filas a eliminar:
  - Si no se indica condición, se eliminan todas las filas de la tabla.

```
1 DELETE
2 FROM <tabla>
3 WHERE <condición>
```

Ejemplo

```
1 DELETE
2 FROM Paises
3 WHERE poblacion<4000000;
```

Actualización de datos

- Se actualizan valores mediante la sentencia `UPDATE`.
- Se indica qué columnas se desea cambiar y su nuevo valor. El resto de columnas mantendrán su valor.
- Con la cláusula `WHERE` se indica qué filas se modifican (todas si se omite esta cláusula).

```
1 UPDATE <table>
2 SET <Columna_1> = <Expressión_1>,
3     <Columna_2> = <Expressión_2>
4 WHERE <condición>
```

Ejemplo

```
1 UPDATE Paises
2 SET
3 poblacion=46438500,
4 superficie=505371
5 WHERE id=1;
```

Ejemplo

```
1 UPDATE Paises
2 SET
3 poblacion=poblacion-1
4 WHERE continente=1;
```

Ejemplo

```
1  UPDATE Paises
2  SET
3  poblacion=
4  (SELECT habitantes
5   FROM AreasGeograficas
6   WHERE Paises.nombre=AreasGeograficas.nombre)
7  WHERE EXISTS
8  (SELECT *
9   FROM AreasGeograficas
10  WHERE Paises.nombre=AreasGeograficas.nombre);
```

Parte V

Consistencia de datos

Consistencia en las claves ajenas

- Una clave ajena debe corresponderse con una clave primaria que exista en la tabla a la que dicha clave ajena hace referencia.
  - Ejemplo: si `continente`=2 en la fila con `id`=3 en la tabla `Paises`, en la tabla `Continentes` debe existir una fila con `id`=2.
- De lo contrario, la base de datos está en un estado inconsistente.
- La columna de clave ajena puede admitir valor `NULL` si se establece así al crear la tabla. Esto no supondría una inconsistencia.

Consistencia en las claves ajenas

- Existe el riesgo de introducir inconsistencias en claves ajenas cuando:
  - Se inserta una nueva fila:
    - Ejemplo: se inserta una fila en `Paises` con `continente`=27, pero no existe esta fila en `Continentes`.
  - Se elimina una fila:
    - Ejemplo: se elimina la fila con `id`=1 en `Continentes`, pero se mantienen filas con `continente`=1 en `Paises`.
  - Se modifica una columna en una fila:
    - Ejemplo: se establece el valor `continente`=27 en una fila de `Paises`, pero no existe esta fila en `Continentes`.
    - Ejemplo: se modifica `id` de 1 a 7 en una fila de `Continentes`, pero se mantienen filas con `continente`=1 en `Paises`.

Ejemplo

```
1  DELETE FROM Continentes WHERE id=1;
2
3  INSERT INTO Paises
4  (nombre, continente, capital,
5   superficie, poblacion)
6  VALUES
7  ('Estados␣Unidos', 27, 5, 0.0, 0);
8
9  UPDATE Paises SET continente=27 WHERE id=1;
10
11 UPDATE Continentes SET id=7 WHERE id=1;
```

Consistencia en las claves ajenas

- El gestor de bases de datos puede detectar sentencias que introducirían inconsistencias y evitarlo.
- Para cada clave ajena se puede configurar el comportamiento deseado en estas situaciones:
  - No ejecutar la sentencia y notificar el error (`RESTRICT` o `NO ACTION`). Es la opción por defecto.
  - Establecer valor NULL en la clave ajena afectada si esta lo permite (`SET NULL`).
  - Propagar el cambio a la clave ajena (`CASCADE`):
    - Ejemplo: si se elimina la fila con `id`=1 en `Continentes`, se eliminan automáticamente todas las filas con `continente`=1 en `Paises`.
    - Ejemplo: si se modifica `id` de 1 a 7 en una fila de `Continentes`, se cambia `continente` de 1 a 7 en todas las filas afectadas de `Paises`.
  - Establecer el valor por defecto de la clave ajena (`SET DEFAULT`).

Ejemplo

```
1  CREATE TABLE Paises (
2          id INT NOT NULL auto_increment,
3          nombre VARCHAR(255) NOT NULL,
4          continente INT NOT NULL,
5          superficie DOUBLE NOT NULL,
6          poblacion LONG NOT NULL,
7          capital INT NOT NULL,
8          PRIMARY KEY(id),
9          CONSTRAINT FOREIGN KEY (continente)
10                 REFERENCES Continentes(id)
11                 ON UPDATE CASCADE
12                 ON DELETE SET NULL,
13         CONSTRAINT FOREIGN KEY (capital)
14                 REFERENCES Ciudades(id)
15 );
```

Tipos de tablas en MySQL

- En MySQL hay varios tipos de tablas que difieren en cómo se almacena la información y qué funcionalidad ofrecen:
  - MyISAM: no transaccional, sin integridad referencial.
  - BerkeleyDB: transaccional, sin integridad referencial.
  - InnoDB: transaccional, con integridad referencial (por defecto desde Mysql 5.5).
  - Otros: `http://dev.mysql.com/doc/refman/5.5/en/storage-engines.html`
- Se puede establecer el tipo de tabla en el comando que la crea.

Ejemplo

```
1  CREATE TABLE Paises (
2          id INT NOT NULL auto_increment,
3          nombre VARCHAR(255) NOT NULL,
4          continente INT NOT NULL,
5          superficie DOUBLE NOT NULL,
6          poblacion LONG NOT NULL,
7          capital INT NOT NULL,
8          PRIMARY KEY(id),
9          CONSTRAINT FOREIGN KEY (continente)
10                 REFERENCES Continentes(id)
11                 ON UPDATE CASCADE
12                 ON DELETE SET NULL,
13         CONSTRAINT FOREIGN KEY (capital)
14                 REFERENCES Ciudades(id)
15 ) ENGINE=INNODB;
```

Parte VI

Índices

Índices

- Los índices consisten en estructuras de datos adicionales cuyo objeto es agilizar la ejecución de determinadas búsquedas de datos en una tabla.
- Ventajas:
  - Localización más rápida de datos en la tabla en acceso aleatorio.
  - Acceso a datos en orden de forma más rápida.
- Desventajas:
  - Coste adicional en la inserción/modificación de datos.
  - Necesidad de más espacio de almacenamiento.
- Para la clave primaria se construye implícitamente un índice.

Creación de índices

```
1  -- Crear un índice en una tabla existente
2  CREATE [UNIQUE] INDEX <nombre_de_índice>
3      ON <nombre_de_tabla> (<col_1>, <col_2>,...);
4
5  -- Crear un índice al mismo tiempo que la tabla:
6  CREATE TABLE <nombre_de_tabla> (
7      (...)
8      [UNIQUE] INDEX [<nombre_de_índice>]
9          (<col_1>, <col_2>,...)
10 );
```

Ejemplo

```
1 CREATE INDEX idx_ciudades_nombre
2 ON Ciudades (nombre);
```

Ejemplo

```sql
CREATE TABLE Ciudades (
        id INT NOT NULL auto_increment,
        nombre VARCHAR(255) NOT NULL,
        superficie DOUBLE NOT NULL,
        poblacion LONG NOT NULL,
        PRIMARY KEY(id),
        INDEX (nombre)
);
```

Ejercicios prácticos

- Práctica 4:
  - Ejercicio 2

Parte VII

Transacciones en SQL

Transacciones en SQL

- El gestor de bases de datos puede recibir sentencias desde varias conexiones concurrentes.
- Una transacción es una secuencia de sentencias SQL que deben ser tratadas como una unidad.
- Deben cumplirse los principios ACID:
  - Atomicidad.
  - Consistencia.
  - Aislamiento.
  - Durabilidad.

Transacciones en SQL

- Atomicidad:
  - O se ejecutan con éxito todas las sentencias de la transacción, o la base de datos debe volver al estado previo al inicio de la transacción.
- Consistencia:
  - Una vez finalizada la transacción, la base de datos debe estar en un estado consistente (se deben cumplir todas las restricciones de consistencia de los datos).

Transacciones en SQL

- Aislamiento:
  - Durante la ejecución de una transacción, sus cambios no pueden ser visibles para el resto de transacciones.
- Durabilidad:
  - Una vez finaliza una transacción con éxito, se debe garantizar que los cambios perduren incluso antes situaciones de fallo en el sistema.

Transacciones en SQL

- Normalmente, cada sentencia SQL individual se ejecuta como una transacción separada.
  - Se puede deshabilitar de forma temporal para realizar solo una transacción con varias sentencias mediante `START TRANSACTION`
  - Se puede deshabilitar en la sesión actual mediante `SET AUTOCOMMIT=0`
- Para finalizar la transacción:
  - Cancelándola: `ROLLBACK`
  - Confirmándola: `COMMIT`

Cerrojos en filas de tablas

- Los gestores de bases de datos relacionales suelen utilizar cerrojos para controlar el acceso concurrente a tablas.
- Principalmente, se usan dos tipos de cerrojos:
  - Cerrojos S (*shared*)
  - Cerrojos X (*exclusive*)

Cerrojos en filas de tablas

- La adquisición de un cerrojo S es compatible con otros cerrojos S sobre la misma fila.
- La adquisición de un cerrojo X no es compatible con ningún otro cerrojo sobre la misma fila.
- El intento de adquisición de un cerrojo bloquea la operación hasta que sea posible. Si hay interbloqueo con otra transacción, la sentencia falla.

Cerrojos en filas de tablas

- En modificaciones y eliminaciones se adquiere X automáticamente, hasta el final de la transacción.
- Las lecturas se pueden realizar de tres formas:
  - *Consistent read*: no se adquiere cerrojo (por defecto).
  - Adquisición de S: `SELECT ... LOCK IN SHARE MODE`.
  - Adquisición de X: `SELECT ... FOR UPDATE`.

Niveles de aislamiento

- En SQL se puede configurar el nivel de aislamiento entre transacciones concurrentes:
  - *READ UNCOMMITTED*
  - *READ COMMITTED*
  - *REPEATABLE READ* (por defecto)
  - *SERIALIZABLE*
- Niveles mayores implican mayor protección en transacciones concurrentes pero peor rendimiento.

Niveles de aislamiento

```
1 SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
2 SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
3 SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
4 SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Referencias

- *The Language of SQL, Second Edition*, Larry Rockoff. Course Technology PTR (2016).
  - Accesible en Safari:

    `http://proquest.safaribooksonline.com/book/databases/sql/9780134658346`

Referencias

- MySQL Transactional and Locking Statements:
  - `http://dev.mysql.com/doc/refman/5.5/en/sql-syntax-transactions.html`
- The InnoDB Transaction Model and Locking:
  - `http://dev.mysql.com/doc/refman/5.5/en/innodb-transaction-model.html`