RESUMEN <u>TEMA 4</u> BASE DE DATOS

CONSULTAS DE SELECCIÓN SIMPLE

1. ORÍGENES Y EVOLUCIÓN DEL LENGUAJE SQL BAJO LA GUÍA DE LOS SGBD

El lenguaje SQL nació con el modelo relacional, propuesto en 1970 por Edgar F. Codd en IBM. El primer SGBD relacional destacado fue System R, desarrollado por IBM como prototipo. Tras demostrar la eficacia del modelo relacional y de SQL, varias empresas compiten por desarrollar sus propios productos:

- *IBM lanzó System/38 (1979), SQL/DS (1981) y DB2 (1983).*
- *Relational Software, Inc. (ahora Oracle Corporation) creó un SGBD relacional para entidades como la Marina de los EE. UU. y la CIA.*

SQL evolucionó de manera desigual hasta que el ANSI publicó el estándar SQL-86 en 1986, ratificado en 1987 por el ISO, estableciendo una versión común del lenguaje para los SGBD relacionales.

2. TIPO DE SENTENCIAS SQL

El lenguaje SQL se organiza en distintos tipos de sentencias según las tareas que realizan en una base de datos relacional. Estas se agrupan en las siguientes categorías:

1. ***DDL (Lenguaje de definición de datos):** Define objetos como tablas, campos, restricciones y reglas de integridad.*
2. ***DCL (Lenguaje de control de datos):** Administrar permisos y controlar el acceso a los objetos de la base de datos.*
3. ***QL (Query language):** Permite realizar consultas para acceder a los datos.*
4. ***DML (Lenguaje de manipulación de datos):** Gestiona modificaciones en la base de datos, como altas, bajas y cambios.*

En algunos SGBD, las consultas (QL) y manipulaciones (DML) se agrupan bajo DML, y a veces las sentencias de control (DCL) se incluyen en DDL. Estas clasificaciones son flexibles y no afectan la funcionalidad del lenguaje.

3. TIPO DE DATOS

1. ***Alfanuméricos**: Para almacenar texto.*
    - ***CHAR**: Cadena de longitud fija (1 a 255 caracteres). Se rellenan con espacios si el texto es más corto.*
    - ***VARCHAR**: Cadena de longitud variable (hasta 65,535 caracteres). No se añaden espacios.*
    - ***BINARY** y **VARBINARY**: Similares a CHAR y VARCHAR, pero almacenan datos en formato binario.*
    - ***TEXT**: Almacenan grandes cantidades de texto, con subtipos (TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT).*
    - ***BLOB**: Para datos binarios grandes (como archivos), con subtipos (TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB).*
    - ***ENUM**: Lista de valores predefinidos. Solo se puede seleccionar un valor.*
    - ***SET**: Lista de valores predefinidos. Se pueden seleccionar múltiples valores.*

2. ***Numéricos**: Para almacenar números enteros y reales.*
    - ***Enteros**:*
        - *TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT: Varían en tamaño y rango de valores soportados.*
    - ***Reales**:*
        - *FLOAT y DOUBLE: Para valores decimales aproximados.*
        - *DECIMAL (NUMERIC): Para valores decimales exactos (útil para datos financieros).*
    - ***Otros**:*
        - ***BIT**: Almacena valores binarios.*
        - ***BOOLEAN**: Sinónimo de TINYINT(1), interpreta 0 como falso y otros valores como verdadero.*

3. ***Momentos temporales**: Para fechas y horas.*
    - ***DATE**: Almacena solo la fecha ('AAAA-MM-DD').*
    - ***DATETIME**: Fecha y hora combinadas ('AAAA-MM-DD HH:MM:SS').*
    - ***TIMESTAMP**: Similar a DATETIME, pero incluye la fecha y hora actuales automáticamente si no se asigna un valor.*
    - ***TIME**: Solo horas ('HH:MM:SS'). También puede representar intervalos de tiempo.*
    - ***YEAR**: Almacena un año (de 1901 a 2155).*

Modificadores:

- ***UNSIGNED**: Permite solo valores positivos.*
- ***ZEROFILL**: Rellena con ceros a la izquierda hasta completar la longitud definida.*
- ***AUTO_INCREMENT**: Incrementa automáticamente el valor de una columna numérica.*

Consideraciones importantes:

- *Es esencial elegir el tipo de dato correcto, ya que define el dominio de valores y optimiza el rendimiento.*
- *La elección del tipo depende de las características específicas del dato (tamaño, precisión, formato, etc.).*

4. Consultas simples

Introducción a las Consultas Simples

- *Para gestionar datos en una base de datos, es necesario:*
    1. *Definir previamente las tablas.*
    2. *Insertar datos en ellas.*
- *El aprendizaje del lenguaje SQL se realiza en orden inverso, comenzando por la consulta de datos.*

Esquema EMPRESA

*El esquema **EMPRESA** consta de **seis entidades** (tablas principales) y sus relaciones:*

1. ***Entidades principales:***
    - ***DEPARTAMENTO**: Contiene datos sobre los departamentos.*
    - ***EMPLEADO**: Almacena información de los empleados.*
    - ***CLIENTE**: Registra a los clientes de la empresa.*
    - ***PRODUCTO**: Representa los productos disponibles.*
    - ***PEDIDO**: Incluye pedidos realizados por los clientes.*
    - ***DETALLE**: Representa los detalles individuales de cada pedido.*

2. ***Relaciones entre las entidades:***
    - ***DEPARTAMENTO** y **EMPLEADO**: Relación 1:N, donde un departamento tiene varios empleados, pero cada empleado pertenece a un solo departamento.*
    - ***EMPLEADO** (reflexiva): Relación 1:N, donde un empleado puede ser jefe de otros empleados.*
    - ***EMPLEADO** y **CLIENTE**: Relación 1:N, donde un empleado puede ser representante de varios clientes.*

- ***CLIENTE** y **PEDIDO**: Relación 1:N, donde un cliente puede tener varios pedidos.*
- ***PEDIDO** y **DETALLE**: Relación fuerte-débil 1:N, ya que cada pedido tiene varios detalles (líneas de pedido).*
- ***DETALLE** y **PRODUCTO**: Relación N:1, donde cada detalle corresponde a un producto.*

Razón de la entidad DETALLE

- *La entidad **DETALLE** evita errores comunes en el diseño:*
  - *En una relación N:N directa entre **PEDIDO** y **PRODUCTO**, un producto no podría aparecer más de una vez en un pedido.*
  - *La entidad **DETALLE** permite que un mismo producto aparezca en un pedido con diferentes condiciones (precio, descuentos, fechas de envío, etc.).*

Esquema relacional y claves

- *Atributos **subrayados**: claves primarias.*
- *Atributos con **(VN)**: permiten valores nulos.*

Tablas principales:

1. ***DEPARTAMENTO***
   (Dept_no, Dnom, Loc (VN))
2. ***EMPLEADO***
   (Emp_No, Nombre, Cargo (VN), Jefe_De (VN), Fecha_Alta (VN), Salario (VN), Comisión (VN), Dept_No)
   *Relación: `Dept_No` referencia a **DEPARTAMENTO**.*
3. ***CLIENTE***
   (Client_Codigo, Nombre, Direccion, Ciudad, Estado (VN), CP, Área (VN), Teléfono (VN), Repr_Cod (VN), Límite_Credito (VN), Observaciones (VN))
   *Relación: `Repr_Cod` referencia a **EMPLEADO**.*
4. ***PRODUCTO***
   (Producto_Numero, Descripción)
5. ***PEDIDO***
   (Pedido_Numero, Pedido_Fecha (VN), Pedido_Tipo (VN), Cliente_Codigo, Fecha_Envío (VN), Total (VN))
   *Relación: `Cliente_Codigo` referencia a **CLIENTE**.*
6. ***DETALLE***
   (Pedido_Numero, Detalle_Numero, Producto_Numero, Precio_Venta (VN), Cantidad (VN), Importe (VN))
   Relaciones:
   - *`Pedido_Numero` referencia a **PEDIDO**.*
   - *`Producto_Numero` referencia a **PRODUCTO**.*

Esquema SANIDAD

*El esquema **SANIDAD** se centra en la gestión de hospitales, salas, doctores, empleados, pacientes y sus ingresos, con el objetivo de proporcionar un contexto para aprender SQL.*

Entidades principales:

1. ***HOSPITAL**: Contiene información sobre los hospitales.*
2. ***SALA**: Almacena datos de las salas de los hospitales.*
3. ***DOCTOR**: Representa a los doctores asociados a cada hospital.*
4. ***PLANTILLA**: Empleados de las salas de los hospitales.*
5. ***ENFERMO**: Pacientes registrados en el sistema.*
6. ***INGRESOS**: Pacientes que actualmente están ingresados en una sala.*

Relaciones entre las entidades:

1. ***HOSPITAL** y **SALA**: Relación fuerte-débil (1:N).*
   Cada hospital tiene varias salas identificadas por un código único dentro de ese hospital.
2. ***HOSPITAL** y **DOCTOR**: Relación fuerte-débil (1:N).*
   Cada hospital tiene varios doctores identificados por un código único dentro del hospital.
3. ***SALA** y **PLANTILLA**: Relación fuerte-débil (1:N).*
   Los empleados se identifican por un código único dentro de cada sala.
4. ***ENFERMO** e **INGRESOS**: Relación fuerte-débil (1:1).*
   Un paciente puede estar ingresado o no.
5. ***SALA** e **INGRESOS**: Relación (1:N).*
   Una sala puede tener múltiples pacientes ingresados, pero cada paciente sólo puede estar en una sala a la vez.

Observaciones del diseño:

1. ***Identificación de empleados:***
   Es inusual identificar a los empleados únicamente dentro de las salas, ya que sería más lógico hacerlo a nivel de hospital (como en la entidad DOCTOR).
2. ***Relación ENFERMO-SALA:***
   *La tabla **ENFERMO** contiene referencias a sala y hospital (campos `Hosp_Ingreso` y `Sala_Ingreso`) para pacientes ingresados, pero muchos registros tendrán estos valores vacíos, generando ineficiencia.*
3. ***Creación de la entidad INGRESOS:***
   *Para evitar pérdida de espacio, se crea la entidad **INGRESOS** que solo almacena información de los pacientes ingresados actualmente.*
4. ***Especialidades y funciones:***
   No existen entidades específicas para las especialidades de los doctores o las funciones del personal, por lo que esta información se almacena como valores alfanuméricos.

Esquema relacional y claves

- ***Atributos subrayados**: claves primarias.*
- ***(VN)**: atributos que permiten valores nulos.*

Tablas relacionales:

1. ***HOSPITAL***
   (Hospital_Cod, Nombre, Dirección (VN), Teléfono (VN), Num_Camas (VN))
2. ***SALA***
   (Hospital_Cod, Sala_Cod, Nombre, Num_Camas (VN))
   Relaciones:
   - `Hospital_Cod` *referencia a* ***HOSPITAL****.*
3. ***PLANTILLA***
   (Hospital_Cod, Sala_Cod, Empleado_No, Apellido, Función (VN), Turno (VN), Salario (VN))
   Relaciones:
   - `{Hospital_Cod, Sala_Cod}` *referencia a* ***SALA****.*
4. ***ENFERMO***
   (Inscripción, Apellido, Dirección (VN), Fecha_Nace (VN), Sexo, Nss (VN))
5. ***INGRESOS***
   (Inscripción, Hospital_Cod, Sala_Cod, Cama (VN))
   Relaciones:
   - `Inscripción` *referencia a* ***ENFERMO****.*
   - `{Hospital_Cod, Sala_Cod}` *referencia a* ***SALA****.*
6. ***DOCTOR***
   (Hospital_Cod, Doctor_No, Apellido, Especialidad)
   Relaciones:
   - `Hospital_Cod` *referencia a* ***HOSPITAL****.*

Puntos a considerar para el aprendizaje SQL:

- *El diseño presenta relaciones fuertes y débiles, ideales para practicar:*
  - ***JOINs** entre tablas con claves compuestas.*
  - *Gestión de claves primarias y foráneas.*
  - *Consultas para datos con valores nulos.*
- *Aunque no es el diseño óptimo para un sistema de gestión hospitalaria, se mantiene para aprender a resolver desafíos comunes en bases de datos relacionales.*

4.1. CLAUSULAS SELECT Y FROM

Cláusula SELECT

- ***Propósito:*** *Especificar las columnas o expresiones que deseas visualizar en el resultado de la consulta.*

Sintaxis básica:
sql
Copiar código
```
SELECT <columna1>, <columna2>, ... FROM <tabla>;
```

-

Ejemplo simple:
sql
Copiar código
```
SELECT emp_no, nombre, cargo FROM empleado;
```

- ***Resultado:*** *Muestra el código, nombre y cargo de los empleados.*

Uso de expresiones: *Permite realizar cálculos sobre los datos, por ejemplo, calcular el salario anual:*
sql
Copiar código
```
SELECT emp_no, nombre, salario * 14 AS "Salario Anual" FROM empleado;
```

-

Alias (sobrenombres): *Ayudan a dar nombres descriptivos a columnas o expresiones. Los alias pueden usar la palabra clave* `AS` *o escribirse directamente:*
sql
Copiar código
```
SELECT emp_no, nombre, salario * 14 AS "Salario Anual" FROM empleado;
SELECT emp_no, nombre, salario * 14 "Salario Anual" FROM empleado;
```

-

Uso del asterisco (*): *Selecciona todas las columnas de una tabla:*
sql
Copiar código
```
SELECT * FROM departamento;
```

-

---

2. Cláusula FROM

- ***Propósito:*** *Indica las tablas donde se encuentran los datos que se desean consultar.*

Permite especificar tablas de otras bases de datos utilizando la sintaxis `<esquema>.<tabla>` *si el usuario tiene permisos adecuados:*
sql
Copiar código

```
SELECT * FROM sanidad.hospital;
```

-

---

3. Consultas avanzadas con varias tablas

- *SQL permite trabajar con varias tablas, obteniendo combinaciones de datos. Esto se denomina* ***producto cartesiano****, y puede restringirse usando alias o filtros.*

Sobrenombres para tablas: *Útiles para simplificar consultas o resolver ambigüedades:*
sql
Copiar código

```
SELECT d.dept_no, s.hospital_cod, s.sala_cod
FROM departamento d, sanidad.sala s;
```

-

---

4. Comandos útiles para explorar tablas

DESC: *Muestra la estructura de una tabla, es decir, sus columnas y propiedades:*
sql
Copiar código

```
DESC departamento;
```

Resumen:

1. ***Cálculos Matemáticos:***
   - *Utiliza operadores aritméticos directamente en las consultas.*
     Ejemplo: `SELECT 4*3-8/2 AS "resultado";`
2. ***Obtención de Fechas del Sistema:***
   - *La función* `SYSDATE()` *permite obtener la fecha actual del sistema.*
     Ejemplo: `SELECT SYSDATE();`
3. ***Ordenación con*** `ORDER BY`***:***
   - *Ordenar resultados según una o más columnas, en orden ascendente (por defecto) ó descendente (*`DESC`*).*
   - *Ejemplo:* `SELECT * FROM departamento ORDER BY loc;`
4. ***Filtrado de Resultados con*** `WHERE`***:***
   - *Usa operadores aritméticos, de comparación y lógicos para establecer condiciones.*
   - *Ejemplos:*
     - *Filtrar por salario:* `SELECT * FROM empleado WHERE salario >= 200000;`
     - *Filtrar por fechas:* `SELECT * FROM empleado WHERE fecha_alta >= '1981-06-01';`
5. ***Operadores Avanzados en*** `WHERE`***:***
   - `LIKE`*: Filtrar cadenas con comodines (*`%` *y* `_`*).*
     - *Ejemplo:* `SELECT * FROM empleado WHERE nombre LIKE 'A%';`
   - `BETWEEN`*: Filtrar valores en un rango.*
     - *Ejemplo:* `SELECT * FROM empleado WHERE salario BETWEEN 100000 AND 200000;`
   - `IN`*/*`NOT IN`*: Filtrar por pertenencia a un conjunto de valores.*
     - *Ejemplo:* `SELECT * FROM empleado WHERE dept_no IN (10, 30);`
6. ***Límite de Filas con*** `LIMIT`***:***
   - *Controla cuántas filas se devuelven y desde qué posición iniciar.*
   - *Ejemplo:* `SELECT * FROM empleado LIMIT 5;` *(Primeras 5 filas)*

Aclaraciones y Consejos:

- ***Alias de Columnas:*** *Usa alias (*`AS`*) para que los resultados sean más legibles.*
  Ejemplo: `SELECT emp_no AS "Código", nombre AS "Empleado";`
- ***Filtrado Combinado:*** *Combina condiciones con* `AND` *y* `OR`*.*
  Ejemplo: `SELECT * FROM empleado WHERE salario >= 200000 AND fecha_alta >= '1981-06-01';`
- ***Optimización con Índices:*** *Si las consultas con* `WHERE` *son lentas, asegúrate de que las columnas utilizadas tengan índices.*