Laboratorio de Bases de Datos (EBB)

Unidad III – Consultas

Departamento de Electricidad, Electrónica y Computación
Facultad de Ciencias Exactas y Tecnología
Universidad Nacional de Tucumán

Primer Cuatrimestre 2017

Introducción [1 | 3]

- Consultas básicas
  - Recuperación de datos
  - Formato de un *ResultSet*
  - Modificación de datos

Introducción [2 | 3]

- Múltiples tablas
  - Combinación de múltiples tablas
  - Combinación de varios *Resultsets*
  - Creación de tabla desde un *Resultset*

Introducción [3 | 3]

- Técnicas avanzadas
  - Subconsultas
  - Subconsultas correlacionadas
  - Modificación de datos

Recuperación de datos [1 | 22]

- **Sentencia `SELECT`:**
  - Se usa para recuperar los datos

```
SELECT [ALL | DISTINCT] <lista_selección>
FROM {<tabla_fuente>} [,…n]
WHERE <predicado_búsqueda>
```


- lista_selección: especifica las columnas a devolver.
- **`FROM`**: indica la o las tablas origen de datos.
- **`WHERE`**: especifica las filas a devolver.

Recuperación de datos [2 | 22]

- **Sentencia `SELECT`:**
  - La lista de selección puede contener:
    - Columnas
    - Expresiones
    - Variables
    - Constantes

Recuperación de datos [3 | 22]

- **Sentencia `SELECT`:**
  - Sobre la lista de selección:
    - Devuelve los resultados en el orden listado
    - Las columnas se separan con comas
    - El (*) devuelve todas las columnas (evitar su uso)

Recuperación de datos [4 | 22]

- **Sentencia `SELECT`:**
  - Sobre la cláusula `WHERE`:
    - Limita el número de filas devueltas por la sentencia `SELECT` según una condición de búsqueda
    - Sólo se devuelven las filas que cumplen con los criterios especificados en la cláusula

Recuperación de datos [5 | 22]

- **Sentencia `SELECT`:**
  - Sobre la cláusula `WHERE`:
    - SQL Server emplea comillas simples para las cadenas y fechas, mientras que MySQL emplea comillas simples o dobles
    - En lo posible se deben emplear condiciones de búsqueda positivas

- Las condiciones de búsqueda negativa agregan una operación más.

Recuperación de datos [6 | 22]

- **Sentencia `SELECT`:**
  - Ejemplo:
    - Para cada producto, mostrar su identificador, nombre y cantidad por unidad

Consultas básicas – SELECT (1)

Recuperación de datos [7 | 22]

- **Sentencia `SELECT`:**
  - En una cláusula `WHERE` se pueden emplear:
    - Comparadores
    - Operadores lógicos
    - Rango de valores
    - Lista de valores
    - Valores desconocidos

Recuperación de datos [8 | 22]

- **Comparadores**
  - `=`, `>`, `<`, `>=`, `<=`, `<>`, `LIKE`
  - Comparan columnas, expresiones, variables y constantes entre sí
  - Usar condiciones positivas (son más rápidas)

Recuperación de datos [9 | 22]

- **Comparadores**
  - Ejemplo:
    - Para el producto cuyo identificador es 10, mostrar su nombre y cantidad por unidad

Recuperación de datos [10 | 22]

- **Comparadores**
  - Para comparar cadenas de caracteres se puede usar el operador `LIKE` en combinación con caracteres especiales:

Caracter	Descripción
`%`	Cualquier cadena de 0 o más caracteres
`_`	Cualquier caracter
`[]`	Cualquier carácter del conjunto
`[^]`	Cualquier carácter menos los del conjunto

Recuperación de datos [11 | 22]

- **Comparadores**
  - Ejemplos:
    - `LIKE 'BR%'`: todos los nombres que empiecen con BR
    - `LIKE '_er'`: palabras de tres letras terminadas en er
    - `LIKE '[VB]%'`: palabras que empiecen en V o B
    - `LIKE '[0-3][^0]'`: números que comiencen con 0,1,2 o 3 y cuyo segundo número no sea 0

Recuperación de datos [12 | 22]

- **Comparadores**
  - Sobre el operador `LIKE`:
    - Todos los caracteres en el patrón son significantes, incluidos los espacios antes y después
    - Se puede usar con cadenas y fechas

Recuperación de datos [13 | 22]

- **Comparadores**
  - Ejemplo:
    - Para cada producto en el que su cantidad se exprese en cajas, mostrar su identificador, nombre y cantidad por unidad

Recuperación de datos [14 | 22]

- **Operadores lógicos**
  - `NOT`, `AND`, `OR`
  - El orden de evaluación es: `NOT`, `AND` y `OR`
  - Combinan expresiones
  - Admiten paréntesis

Recuperación de datos [15 | 22]

- **Operadores lógicos**
  - Ejemplo:
    - Para cada producto en el que su cantidad se exprese en cajas o botellas, mostrar su identificador, nombre y cantidad por unidad

Consultas básicas – SELECT (4)

Recuperación de datos [16 | 22]

- **Rango de valores**
  - Para obtener filas cuyos valores de búsqueda estén dentro de un rango se usa el operador `BETWEEN`
  - Ejemplo:
    - Para cada producto cuyo identificador esté entre 1 y 10, mostrar su identificador, nombre y cantidad por unidad

Consultas básicas – SELECT (5)

Recuperación de datos [17 | 22]

- **Rango de valores**
  - Consideraciones:
    - La inclusión de los extremos del intervalo depende del SGBDR
    - Es mejor usar `BETWEEN` que `<=` y `>=`
    - Evitar el uso de `NOT BETWEEN`
    - En las fechas hay que tener en cuenta la hora

- La conveniencia del uso del operador `BETWEEN` con respecto a los comparadores `<=` y `>=` tiene que ver más que nada con cuestiones de legibilidad. Además `BETWEEN` se puede usar negado (`NOT`).

Recuperación de datos [18 | 22]

- **Lista de valores**
  - Para obtener filas cuyos valores de búsqueda pertenezcan a un conjunto dado se usa la condición `IN`
  - Ejemplo:
    - Para cada empleado cuya ciudad de origen sea Londres o Seattle, mostrar su apellido, nombre y ciudad

Consultas básicas – SELECT (6)

Recuperación de datos [19 | 22]

- **Lista de valores**
  - Consideraciones:
    - Se pueden reemplazar por series de expresiones `OR`
    - No incluir el valor nulo en el conjunto (resultados inesperados)
    - El uso de `NOT IN` disminuye el rendimiento

- La condición `IN` se puede reemplazar por series de expresiones `OR`. Por ejemplo, `City IN ('Bothell', 'Everett')` equivale a `City = 'Bothell' OR City = 'Everett'`.

Recuperación de datos [20 | 22]

- **Valores desconocidos**
  - Una columna tiene el valor nulo si no se ingresó valor alguno
  - Un valor nulo no equivale a la cadena vacía ni a 0
  - Para devolver filas cuyos valores de búsqueda sean nulos se usa `IS NULL`

Recuperación de datos [21 | 22]

- **Valores desconocidos**
  - Ejemplo:
    - Para cada empleado del cual se conozca su región de origen, mostrar su apellido, nombre y región

Recuperación de datos [22 | 22]

- **Valores desconocidos**
  - Consideraciones:
    - Los valores nulos hacen fallar a todas las comparaciones
    - Se define la posibilidad de agregar nulos en las columnas en la sentencia `CREATE TABLE`
    - Usar `IS NOT NULL` para devolver filas con valores conocidos

Formato de un *ResultSet* [1 | 15]

- **Orden de los datos**
  - Para ordenar las filas en orden ascendente (`ASC`) o descendente (`DESC`) se emplea la cláusula `ORDER BY`
  - Ejemplo:
    - Para cada producto, mostrar su identificador, nombre y precio unitario. Ordenarlos descendentemente según el precio

Consultas básicas – Orden de datos

- Un *ResultSet* es un conjunto de filas de una BD, más los metadatos sobre la consulta, como ser los nombres de las columnas, los tipos y tamaño de cada una, etc.

Formato de un *ResultSet* [2 | 15]

- **Orden de los datos**
  - Consideraciones:
    - Si no se emplea `ORDER BY` no se garantiza un orden determinado
    - Por defecto se ordena ascendentemente (`ASC`)
    - Las columnas por las que se ordenan no necesitan aparecer en la lista de selección
    - Se puede ordenar por columnas o expresiones y referirse a las mismas por su número de orden en la lista de selección

- El orden se especifica cuando se instala el SGBDR.

- Si bien las columnas por las que se ordena no necesitan aparecer en la lista de selección, algunos motores tienen ciertas restricciones. Por ejemplo, en SQL Server estas columnas no deben exceder los 8060 bytes.

Formato de un *ResultSet* [3 | 15]

- **Eliminación de duplicados**
  - Para eliminar filas duplicadas se puede usar la cláusula
  - Ejemplo:
    - Mostrar los clientes que tienen órdenes. Mostrar sólo los identificadores ordenados ascendentemente

Consultas básicas – Eliminación de duplicados

Formato de un *ResultSet* [4 | 15]

- **Eliminación de duplicados**
  - Consideraciones
    - La combinación de valores en la lista de selección determina la desigualdad entre filas
    - Solo se muestra un elemento de todas las filas que tienen la misma combinación de valores de la lista de selección
    - `DISTINCT` no ordena, para ello se usa `ORDER BY`

Formato de un *ResultSet* [5 | 15]

- **Cambio de nombres a las columnas**
  - La cláusula `AS` permite crear alias a las columnas para que resulten más legibles o den nombres a las expresiones

Consultas básicas – Cambio de nombres a las columnas

Formato de un *ResultSet* [6 | 15]

- **Cambio de nombres a las columnas**
  - Consideraciones:
    - Por defecto se muestran los nombres de las columnas
    - Encerrar los alias entre comillas simples si incluyen espacios
    - Se pueden crear alias para expresiones
    - Se puede omitir la cláusula `AS` y dejar un espacio entre la expresión y el alias

- En SQL Server los nombres de los alias pueden tener hasta 128 caracteres.

Formato de un *ResultSet* [7 | 15]

- **Uso de constantes**
  - Las constantes o literales son letras, números o símbolos

Consultas básicas – Uso de constantes

Formato de un *ResultSet* [8 | 15]

- **CASE**
  - Evalúa una lista de condiciones devolviendo una de las varias expresiones de resultado posibles
  - Tiene 2 formatos:
    - **Sencillo**: compara una expresión con un conjunto de expresiones sencillas para determinar el resultado
    - **De búsqueda**: evalúa un conjunto de expresiones booleanas para determinar el resultado

- MySQL también soporta este operador.

Formato de un *ResultSet* [9 | 15]

- **CASE**
  - Formato simple:

```
CASE expresión
  WHEN valor THEN resultado [ ...n ]
  [ELSE resultado ]
END
```

  - Formato de búsqueda:

```
CASE
  WHEN exp_bool THEN resultado [ ...n ]
  [ ELSE resultado ]
END
```

Consultas básicas – CASE simple / CASE búsqueda

Formato de un *ResultSet* [10 | 15]

- **CASE**
  - Consideraciones:
    - Se puede utilizar en cualquier sentencia o cláusula que permita una expresión válida. Ejemplo: sentencias `SELECT`, `UPDATE`, `DELETE` y `SET`, cláusulas `IN`, `WHERE`, `ORDER BY`, `HAVING`, etc
    - SQL Server permite 10 niveles de anidamiento en las expresiones `CASE`

Formato de un *ResultSet* [11 | 15]

- **CASE**
  - Consideraciones (continuación):
    - La expresión `CASE` no se puede utilizar para controlar el flujo de ejecución de bloques de instrucciones, funciones definidas por el usuario, procedimientos almacenados e instrucciones SQL
    - La instrucción `CASE` devuelve las condiciones de forma secuencial y se detiene en la primera condición cuya condición se cumple

Formato de un *ResultSet* [12 | 15]

- **Función `COALESCE`**
  - `COALESCE ( expresión [ ,...n ] )`
  - Forma abreviada de un tipo especial de `CASE`:

```
CASE
  WHEN (expr1 IS NOT NULL) THEN expr1
  WHEN (expr2 IS NOT NULL) THEN expr2
  ...
  ELSE exprN
END
```

Consultas básicas – coalesce

- La función `COALESCE()` el primer valor no `NULL` de su lista de argumentos, o bien `NULL` si todos los valores son `NULL`.

Formato de un *ResultSet* [13 | 15]

- **Función `COALESCE`**
  - Consideraciones
    - Si todos los argumentos son `NULL` devuelve `NULL`

Formato de un *ResultSet* [14 | 15]

- **Función `ISNULL` (SQL Server)**
  - Reemplaza `NULL` con el valor especificado
    - `ISNULL (expresión , valor)`
- **Función `IFNULL` (MySQL)**
  - Reemplaza `NULL` con el valor especificado
    - `IFNULL (expresión , valor)`

Consultas básicas – isnull (SQL Server) / ifnull (MySQL)

- En MySQL la función análoga a `ISNULL()` en SQL Server se llama `IFNULL()`. En MySQL, la función `ISNULL()` verifica si una expresión es `NULL` o no, devolviendo 1 o 0 respectivamente.

Formato de un *ResultSet* [15 | 15]

- **Función `NULLIF`**
  - Devuelve `NULL` si las 2 expresiones son iguales
  - `NULLIF (expresión1 , expresión2)`

Consultas básicas – nullif

- Si no se hubiera empleado la función `NULLIF` y se calculara el promedio sobre la columna `C1`, el valor sería 170000 (850000 / 5). La función `AVG` al calcular el promedio, no cuenta las filas que valen `NULL`.

Modificación de datos [1 | 12]

- **Inserción de filas**
  - Para agregar filas a una tabla se usa la sentencia `INSERT`

```
INSERT [INTO] tabla [(lista_columnas)]
VALUES (lista_valores) | DEFAULT VALUES
```


- Si no se especifica la lista de columnas, el orden es igual al de la tabla.

Modificación de datos [2 | 12]

- **Inserción de filas**
  - Consideraciones:
    - Usar la lista de columnas para forzar el orden de los valores
    - Especificar los datos a ingresar luego de la palabra `VALUES` (el orden y tipo de datos deben coincidir con los de la tabla)
    - La sentencia `INSERT` falla si se violan las restricciones o reglas

Modificación de datos [3 | 12]

- **Inserción de filas**
  - Consideraciones (continuación):
    - Si la columna tiene valores por defecto o acepta nulos, se puede omitir el nombre en la sentencia `INSERT`
    - Especificar el valor nulo escribiendo `NULL` (sin comillas)

Modificación de datos [4 | 12]

- **Inserción de filas**
  - Cuando se insertan filas a una tabla, se pueden emplear las cláusulas `DEFAULT` o `DEFAULT VALUES` (la última cláusula disponible sólo en SQL Server)

Consultas básicas – Inserción de filas con valores por defecto MySQL/ SQL Server

- En el ejemplo, si la tabla Clientes tuviera una columna IDCliente con la propiedad Identity, otra columna Apellidos con un valor por defecto = 'Perez', otra columna Nombres con un valor por defecto = 'Juan' y una columna Domicilio cuyo valor por defecto es NULL, se agregaría una fila con estos valores en las columnas, y en la que tiene la propiedad Identity el siguiente valor.

Modificación de datos [5 | 12]

- **Inserción de filas**
  - `DEFAULT`
    - Inserta un valor nulo en las columnas que no tengan definido un valor por efecto y admitan nulos
    - Si la columna no admite nulos ni tiene valor por defecto, falla la sentencia
    - No se puede usar para columnas `IDENTITY`

Modificación de datos [6 | 12]

- **Inserción de filas**
  - `DEFAULT VALUES`:
    - Permite insertar toda una fila por defecto en la tabla
    - En columnas timestamp o `IDENTITY` inserta el siguiente valor apropiado
    - Permite generar datos de prueba y poblar tablas de manera sencilla
    - No disponible en MySQL

Modificación de datos [7 | 12]

- **Borrado de filas**
  - `DELETE`: borra una o más filas de una tabla
  - `TRUNCATE`: borra **todas** las filas de una tabla

- Ambos comandos sólo borran filas de una tabla, no la tabla en sí misma.

Modificación de datos [8 | 12]

- **Borrado de filas**
  - `DELETE`
    - Se pueden limitar las filas a borrar (`WHERE`)
    - Cada fila borrada se registra en el registro de transacciones como una transacción

- Si no se especifica la cláusula `WHERE` se borran todas las filas de la tabla.
- Como las filas borradas se registran en el registro de transacciones, las mismas se pueden volver a agregar (roll back).

Modificación de datos [9 | 12]

- **Borrado de filas**
  - `TRUNCATE`: borra **todas** las filas de una tabla
  - Si la tabla tiene una columna `IDENTITY`, vuelve el valor de la semilla al valor inicial
  - Se ejecuta mucho más rápido que `DELETE` ya que no se registra en el registro de transacciones

Consultas básicas – Borrado de filas en MySQL/SQL Server

- Como las filas que se borran no se registran en el registro de transacciones, las mismas no se pueden volver a aplicar.

- La operación `TRUNCATE` se ejecuta mucho más rápido que `DELETE` ya que la primera en realidad borra y vuelve a recrear la tabla, lo cual es más rápido que borrar todas las filas una a una.

Modificación de datos [10 | 12]

- **Modificación de filas**
  - Para agregar filas a una tabla se usa la sentencia `UPDATE`

Consultas básicas – Modificación de filas

- Si no se especifica la cláusula `WHERE`, se modifican todas las filas.

Modificación de datos [11 | 12]

- **Modificación de filas**
  - Consideraciones:
    - Las filas a modificar se especifican en la cláusula `WHERE`
    - Los valores nuevos se especifican con la cláusula `SET`
    - Se pueden cambiar los datos de una tabla a la vez

Modificación de datos [12 | 12]

- **Modificación de filas**
  - Consideraciones (continuación):
    - Los valores ingresados deben ser del mismo tipo de las columnas a modificar y no deben violar las restricciones y reglas
    - Se pueden usar expresiones basadas en columnas de la tabla u otras tablas, variables y constantes

Combinación de múltiples tablas [1 | 17]

- Para producir un resultado que incorpore filas de 2 o más tablas se utiliza el operador `JOIN`

- Tipos de `JOIN`
  - `INNER`
  - `OUTER`
  - `CROSS`

Combinación de múltiples tablas [2 | 17]

```
SELECT lista_selección
FROM t1 [INNER | LEFT | RIGHT | OUTER] JOIN t2
ON condición
```

- `JOIN`: especifica qué tablas son las que se unen y cómo se unen
- `ON`: especifica las columnas en común que tienen las tablas

Combinación de múltiples tablas [3 | 17]

- Consideraciones:
  - Especificar la condición según las PKs y FKs de las tablas
  - Si la tabla tiene una clave compuesta, se deben referenciar todas las columnas luego de la palabra ON
  - Si 2 o más tablas tienen las mismas columnas, anteponer el nombre de la tabla

Combinación de múltiples tablas [4 | 17]

- Consideraciones (continuación):
  - Limitar el número de tablas ya que aumenta el tiempo de proceso de la consulta

Combinación de múltiples tablas [5 | 17]

- **`INNER JOIN`:**
  - Combina tablas comparando los valores en las columnas comunes y devuelve sólo las filas que cumplan con la condición

Combinación de múltiples tablas [6 | 17]

- **`INNER JOIN`:**
  - Ejemplo:
    - Para cada producto, mostrar su identificador, nombre y categoría

Múltiples tablas – INNER JOIN

Combinación de múltiples tablas [7 | 17]

- **`INNER JOIN`:**
  - Consideraciones:
    - Es el `JOIN` por defecto (se puede omitir `INNER`)
    - Si se incluye `WHERE` se restringe la cantidad de filas devueltas
    - No usar `NULL` como condición
    - No se garantiza un orden en el resultado (usar `ORDER BY`)

Combinación de múltiples tablas [8 | 17]

- **`OUTER JOIN`:**
  - Un `OUTER JOIN` puede ser:
    - `LEFT`: combina filas de 2 tablas que satisfacen la condición del `JOIN` más las filas de la tabla izquierda que no la satisfacen
    - `RIGHT`: combina filas de 2 tablas que satisfacen la condición del `JOIN` más las filas de la tabla derecha que no la satisfacen

Combinación de múltiples tablas [9 | 17]

- **`OUTER JOIN:`**
  - Ejemplo:
    - Para cada cliente, mostrar su identificador, orden y fecha. Contemplar el caso de los clientes que no tuvieran órdenes

Múltiples tablas – OUTER JOIN

Combinación de múltiples tablas [10 | 17]

- **`OUTER JOIN`:**
  - Consideraciones:
    - `A LEFT JOIN B = B RIGHT JOIN A`
    - No usar `NULL` en la condición
    - Usarlo entre 2 tablas preferentemente
    - Se puede omitir `OUTER`

Combinación de múltiples tablas [11 | 17]

- **`CROSS JOIN`:**
  - Realiza el producto cartesiano de 2 tablas (no requiere columnas en común)
  - Generalmente se lo usa para poblar rápidamente tablas
  - Cardinalidad(`A CROSS JOIN B`) = Cardinalidad(`A`) * Cardinalidad(`B`)

Combinación de múltiples tablas [13 | 17]

- **`CROSS JOIN`:**
  - Ejemplo:
    - Mostrar el identificador del empleado combinado con todas las órdenes

Múltiples tablas – CROSS JOIN

Combinación de múltiples tablas [14 | 17]

- **Combinación de más de 2 tablas:**
  - Si varias tablas están relacionadas con FKs, se las puede combinar
  - Ejemplo:
    - Mostrar el identificador de producto, su nombre, nombre de la compañía del proveedor y categoría del producto

Múltiples tablas - Combinación de más de 2 tablas

Combinación de múltiples tablas [15 | 17]

- **Combinación de más de 2 tablas:**
  - Consideraciones:
    - `A JOIN B JOIN C = (A JOIN B) JOIN C`
    - `A JOIN B JOIN C = (B JOIN C) JOIN A`
    - Las tablas deben estar relacionadas con FKs
    - Se deben considerar las claves compuestas en las condiciones
    - Los `JOIN`s son conmutativos

Combinación de múltiples tablas [16 | 17]

- **`JOIN` de una tabla con sí misma:**
  - Se realiza para encontrar filas que tengan valores en común con otras filas de la misma tabla
  - Ejemplo:
    - Para cada empleado, mostrar su identificador, apellido, nombre y a quién reporta (de este último también mostrar su identificador, apellido y nombre)

Múltiples tablas – JOIN de una tabla con sí misma

- Para entender el funcionamiento de la consulta, tomar como que hubiera 2 tablas iguales, E1 y E2. De E1 se obtiene el identificador de empleado, apellido, nombre y a quién reporta. Ahora para saber el apellido y nombre de a quién se reporta, se hace un join con la tabla E2, donde la condición será E1.ReportsTo = E2.ReportsTo.

Combinación de múltiples tablas [17 | 17]

- **`JOIN` de una tabla con sí misma:**
  - Consideraciones:
    - Se debe usar un alias para referenciar 2 copias de la misma tabla

Combinación de varios *ResultSets* [1 | 2]

- Para combinar los resultados de 2 o más sentencias `SELECT` en un solo *ResultSet* se usa el operador `UNION`

```
sentencia_select UNION [ALL] sentencia_select...
```

- Se usa cuando los datos residen en diferentes ubicaciones y no se pueden acceder en una sola consulta

Múltiples tablas - Combinación de varios ResultSets

- En los ejemplos, se combinan clientes y empleados.
- En la primera sentencia SELECT se pueden usar alias.

Combinación de varios *ResultSets* [2 | 2]

- Consideraciones:
  - Los resultados deben tener el mismo tipo de datos, número y orden de columnas en la lista de selección
  - Automáticamente se remueven las filas duplicadas, a menos que se emplee la cláusula `ALL`
  - El resultado se devuelve en cualquier orden. Para que salga ordenado se debe emplear `ORDER BY`

Creación de tabla desde un *ResultSet* [1 | 3]

- Se puede crear una nueva tabla en base al resultado de una consulta usando `SELECT INTO` (SQL Server) o `CREATE SELECT` (MySQL)
- MySQL también permite clonar una tabla

Múltiples tablas – Creación de una tabla desde un ResultSet en MySQL / SQL Server

- En la clonación de la tabla, sólo se clona la estructura.

Creación de tabla desde un *ResultSet* [2 | 3]

- Consideraciones:
  - Siempre se crea una tabla y se inserta el *Resultset* en la misma
  - Todas las columnas deben tener nombre (usar alias)
  - En SQL Server se debe tener el permiso select into/bulk copy
  - Al crear la tabla se puede especificar que la misma sea temporal. Una tabla temporal brinda un espacio de trabajo para resultados intermedios.

•Una tabla temporal brinda un espacio de trabajo para resultados intermedios.

Creación de tabla desde un *ResultSet* [3 | 3]

- Consideraciones:
  - En SQL Server se pueden tener tablas temporales:
    - **Locales** (#): se libera el espacio que ocupan cuando se termina la sesión que las creó
    - **Globales** (##): se libera el espacio que ocupan cuando termina la última sesión abierta en el servidor
  - En MySQL las tablas temporales sólo pueden ser locales

Múltiples tablas – Creación de una tabla temporal en MySQL/SQL Server

- Las tablas temporales locales implican que 2 sesiones diferentes pueden usar el mismo nombre para estas tablas sin que se produzcan conflictos.

- En el caso de MySQL, para crear tablas temporales se debe tener el permiso `CREATE TEMPORARY TABLES`.

Subconsultas [1 | 6]

- **Subconsulta**: consulta anidada dentro de otra

```
SELECT * FROM t1
WHERE col1 = (SELECT col1 FROM t2);
```

- `SELECT * FROM t1`: consulta o sentencia externa
- `(SELECT column1 FROM t2)`: subconsulta (anidada dentro de la consulta externa)

Subconsultas [2 | 6]

- Consideraciones:
  - La subconsulta siempre va entre paréntesis
  - Se pueden anidar subconsultas dentro de otras
  - Una subconsulta puede devolver un único valor (escalar), una sola fila, una sola columna o una tabla

Subconsultas [3 | 6]

- Consideraciones (continuación):
  - Una subconsulta puede contener casi cualquier cláusula: `DISTINCT`, `GROUP BY`, `ORDER BY`, joins, `UNION`, comentarios, functions, etc
  - No se puede modificar una tabla y seleccionar la misma tabla en una subconsulta

Subconsultas [4 | 6]

- Ventajas:
  - Permiten aislar cada parte de una sentencia
  - Brindan formar alternativas de realizar operaciones que requerirían uniones y combinaciones (JOIN) complejas

Subconsultas [5 | 6]

- Ejemplo:
  - Mostrar las filas en la tabla t1 donde el valor de la columna col1 sea igual al máximo valor de la columna col2 en la tabla t2

```
SELECT * FROM t1
WHERE col1 = (SELECT MAX(col2) FROM t2);
```

Subconsultas [6 | 6]

- `EXISTS` y `NOT EXISTS`:
  - Si una subconsulta devuelve al menos una fila, por más que toda tenga valores `NULL`, `EXISTS` devuelve TRUE y `NOT EXISTS` FALSE

Subconsultas correlacionadas [1 | 2]

- **Subconsulta correlacionada**: subconsulta que contiene una referencia a una tabla que también aparece en la consulta externa

```
SELECT *
FROM t1
WHERE col1 = (SELECT col1 FROM t2
                        WHERE t2.col2 = t1.col2)
```


- Observar que la subconsulta tiene una referencia a una columna de t1, aunque la cláusula `FROM` no la mencione. El SGBDR busca entonces fuera de la subconsulta, y la encuentra en la consulta externa.

Subconsultas correlacionadas [2 | 2]

- Consideraciones:
  - La evaluación se hace desde adentro hacia afuera:

```
SELECT col1 FROM t1 AS x
WHERE x.col1 = (SELECT col1 FROM t2 AS x
     WHERE x.col1 = (SELECT col1 FROM t3
                          WHERE x.col2 = t3.col1))
```


- Observar que la subconsulta tiene una referencia a una columna de t1, aunque la cláusula `FROM` no la mencione. El SGBDR busca entonces fuera de la subconsulta, y la encuentra en la consulta externa.

Modificación de datos [1 | 6]

- **Agregado de filas a una tabla existente:**
  - Se puede emplear `INSERT ... SELECT`:

Técnicas avanzadas - Inserción de filas basadas en otras tablas

Modificación de datos [2 | 6]

- **Agregado de filas a una tabla existente:**
  - Consideraciones:
    - Se agregan todas las filas que satisfagan la condición del `SELECT`
    - Las columnas de la tabla destino deben tener tipos de datos compatibles a la lista de selección
    - Siempre se agregan filas a una única tabla

Modificación de datos [3 | 6]

- **Agregado de filas a una tabla existente:**
  - Consideraciones (continuación):
    - Se debe determinar la existencia de valores por defecto o valores nulos en caso de omitir alguna columna

Modificación de datos [4 | 6]

- **Borrado de filas basadas en otras tablas:**
  - Se puede usar `DELETE` con `JOIN`s (sólo SQL Server) o subconsultas

Borrado de filas basadas en otras tablas (DELETE con JOIN en SQL Server)

Borrado de filas basadas en otras tablas (subconsultas)

- En el caso de SQL Server, el primer `FROM` indica la tabla donde se borrarán las filas, y el segundo `FROM` hace referencia al criterio restrictivo y en él se pueden introducir joins o subconsultas.

- El `DELETE` con `JOIN`s no es soportado por MySQL.

Modificación de datos [5 | 6]

- **Modificación de filas basadas en otras tablas:**
  - Se puede usar `UPDATE` con `JOIN`s (sólo SQL Server) o subconsultas

Modificación de filas basadas en otras tablas en SQL Server (UPDATE con JOINs)

Modificación de filas basadas en otras tablas (subconsultas)

- Esta forma no es soportada por MySQL.

Modificación de datos [6 | 6]

- **Modificación de filas basadas en otras tablas:**
  - Consideraciones:
    - Nunca se modifica la misma fila dos veces en una sentencia `UPDATE`. Esta restricción permite el ahorro de mucho tiempo
    - Usar la cláusula `FROM` en la sentencia `UPDATE`

Resumen [1 | 2]

- Uso de la sentencia `SELECT`
- Filtrado: `WHERE`
- Ordenamiento: `ORDER BY`
- Eliminación de filas duplicadas
- Uso de alias y constantes
- Inserción, borrado y modificación de filas
- Consulta de varias tablas: operador `JOIN`
- Tipos de `JOIN`: `INNER`, `OUTER` y `CROSS`
- Combinación de múltiples *ResultSets*

Resumen [2 | 2]

- Creación de una tabla a partir de un *ResultSet*
- Uso de `EXISTS` y `NOT EXISTS`
- Inserción, borrado y modificación de filas a partir de otras