
SQL

— Structured Query Language —



Introducción

Introducción

- Los lenguajes son las herramientas a través de las que interactuamos con los modelos de datos
- Los lenguajes se dividen en varios tipos
 - **Lenguajes de Definición de datos (DDL):** permiten expresar la estructura y restricciones del modelo de datos
 - **Lenguajes de Manipulación de datos (DML):** permiten ingresar, modificar, eliminar y consultar datos
 - **Lenguajes de Control de datos (DCL):** permiten definir permisos de acceso a los datos

SQL - Historia

- Su primer versión es de 1974 (SEQUEL), desarrollada por IBM Research
- Implementado en System R de IBM (1977)
- Oracle (1979) fue su primer versión comercial (Relational Software Inc, actual Oracle Corporation)
- SQL: Structured Query Language
- Estándar ANSI desde 1986 e ISO desde 1997, fue incorporando distintas funcionalidades con los años (WITH RECURSIVE, CUBE, XML, JSON, etc)

SQL - Lenguaje

- Es declarativo, no procedural
- Basado en cálculo de tuplas
- Trabajamos con tablas en vez de relaciones

Modelo Relacional	SQL
Relación	Tabla
Tupla	Fila
Atributo	Columna

SQL - Estructura del estándar

- El estándar ISO SQL tiene actualmente 9 partes:

- ISO/IEC 9075-1: Framework (SQL/Framework)
- **ISO/IEC 9075-2: Foundation (SQL/Foundation)** →
- ISO/IEC 9075-3: Call-Level Interface (SQL/CLI)
- ISO/IEC 9075-4: Persistent Stored Modules (SQL/PSM)
- ISO/IEC 9075-9: Management of External Data (SQL/MED)
- ISO/IEC 9075-10: Object Language Bindings (SQL/OLB)
- **ISO/IEC 9075-11: Information and Definition Schemas (SQL/Schemata)** ↗
- ISO/IEC 9075-13: SQL Routing and Types using Java (SQL/JRT)
- ISO/IEC 9075-14: XML Related Specifications (SQL/XML)

CORE SQL

- El estándar SQL es abierto pero no es gratuito. En <http://modern-sql.com/standard> pueden encontrar drafts muy cercanos a la publicación de algunas versiones

Definición de Datos

DDL - Data Definition Language

DDL - Esquemas

- Para crear esquemas se usa el comando **CREATE SCHEMA**

```
CREATE SCHEMA nombre_esquema [ AUTHORIZATION AuthId ] ;
```

```
CREATE SCHEMA empresa AUTHORIZATION lroman;
```

- En un entorno SQL puede haber varios esquemas agrupados en colecciones llamadas catálogos
- Todo catálogo tiene un esquema llamado **INFORMATION_SCHEMA** que describe a los demás esquemas dentro del catálogo

DDL - Tablas

- Para definir estructuras de tablas se usa el comando **CREATE TABLE**

```
CREATE TABLE nombre_tabla (  
    definicion_columna_1 [,  
    definicion_columna_2,  
    ...,  
    definicion_columna_N ] [,  
    definicion_restriccion_1,  
    ...,  
    definicion_restriccion_M ]  
);
```

No es obligatorio definir restricciones, pero **MUY** recomendado siempre definir una de PK

DDL - Definición de columnas

- La definición de columnas tiene el siguiente formato

```
nombre_columna tipo_dato [ restricciones ]
```

```
padron INTEGER NOT NULL CHECK (padron > 10000)
```

DDL - Tipos de dato

```
nombre_columna tipo_dato [ restricciones ]
```



- **Numéricos**

- INTEGER o INT
- SMALLINT
- FLOAT(n)
- DOUBLE PRECISION
- NUMERIC(precision, escala)

- **Fechas y horas**

- DATE
- TIME
- TIMESTAMP (con o sin TZ)
- INTERVAL

- **Strings**

- CHARACTER(n) o CHAR(n)
- CHARACTER VARYING(n) o VARCHAR(n)

- **Otros**

- BOOLEAN
- CLOB
- BLOB
- UUID

DDL - Restricciones (algunas)

```
nombre_columna tipo_dato [ restricciones ]
```



- PRIMARY KEY
 - FOREIGN KEY
 - UNIQUE
 - NULL o NOT NULL
 - DEFAULT *valor_defecto*
 - CHECK *condicion*
- Las restricciones pueden definirse luego de las columnas, y usando **CONSTRAINT** se les puede dar un nombre

```
CONSTRAINT cc_alumno_cuit UNIQUE (cuit)
```

DDL - Ejemplo completo

```
CREATE TABLE Persona (  
    dni INT PRIMARY KEY,  
    nombre VARCHAR(255) NOT NULL,  
    fecha_nacimiento DATE  
);  
  
CREATE TABLE HijoDe (  
    dni_hijo INT,  
    dni_padre INT,  
    PRIMARY KEY (dni_hijo, dni_padre),  
    FOREIGN KEY (dni_hijo) REFERENCES Persona(dni),  
    FOREIGN KEY (dni_padre) REFERENCES Persona(dni)  
);
```

DDL - Restricción de clave foránea

- En una restricción de **FOREIGN KEY** podemos opcionalmente definir estrategias en **DELETE** o **UPDATE** de la tabla referenciada
 - Cada SGBD tiene su default

```
[ CONSTRAINT nombre_restriccion ]  
FOREIGN KEY (columnas)  
REFERENCES tabla_referenciada (columnas_referenciadas)  
[ ON DELETE SET NULL | RESTRICT | CASCADE | SET DEFAULT ]  
[ ON UPDATE SET NULL | RESTRICT | CASCADE | SET DEFAULT ]
```

DDL - Claves sustitutas / Ids secuenciales

- Muy dependientes de cada motor
- PostgreSQL tiene los tipos **SERIAL** y **BIGSERIAL** de 4 y 8 bytes respectivamente
- SQL Server tiene el tipo **IDENTITY**
- MySQL tiene la opción **AUTO_INCREMENT**
- Oracle usa secuencias y triggers **BEFORE INSERT**
- Todo esto puede cambiar en versiones posteriores a estas filminas, consultar la documentación

DDL - Cambios en tablas

- Una vez creadas las tablas pueden cambiar su estructura usando **ALTER TABLE**. Algunos usos posibles son:

```
ALTER TABLE Persona ADD COLUMN direccion VARCHAR(255);
```

```
ALTER TABLE Persona RENAME TO DatosPersonas;
```

- Y para quitar tablas se usa **DROP TABLE**


```
DROP TABLE DatosPersonas [ CASCADE ];
```


DDL - Índices

- Los índices se pueden utilizar para mejorar la performance de acceso a los datos
 - Aunque su mantenimiento tiene un costo, con lo que pueden impactar negativamente en la performance

```
CREATE [ UNIQUE ] INDEX nombre_indice
    ON nombre_tabla
        ( expresion1 [, ..., expresionN] )
    [ INCLUDE (columna1 [, ..., columnaN] ) ]
    [ WHERE condicion ]
);
```

Se indexa una tabla
por una o más
expresiones



DDL - Índices

- Pueden agilizar consultas por igualdad y por rango

```
CREATE INDEX idx_padron ON alumnos(padron) ;
```

- Para quitar índices también se utiliza **DROP**

```
DROP INDEX nombre_indice;
```

- Distintos motores permiten distintos tipos de índice con sus ventajas y desventajas
 - Revisar documentación

DDL - Índices que incluyen columnas

- Pueden incluirse columnas no indexadas en el índice

```
CREATE INDEX idx_padron ON alumnos (padron)  
    INCLUDE (nombre, apellido);
```

- Esto permite buscar por padrón y devolver nombre y/o apellido sin acceder a la tabla (más rápido)
 - No agiliza búsquedas por nombre ni apellido
- Pero el índice ocupa más que si no tuviera las columnas
- Usar con cuidado, no incluir columnas muy grandes ni muchas columnas “porque sí”

DDL - Índices parciales

- Puede no indexar toda la tabla

```
CREATE INDEX idx_clientes ON clientes(id)  
WHERE anulado = FALSE;
```

- Sólo se indexan las filas que cumplan la condición
 - Índice más pequeño suele tener mejor performance
- Sólo se usará el índice para consultas que tengan incluida esa condición
 - Consultar clientes anulados no aprovechará el índice

Manipulación de Datos

DML - Data Manipulation Language

Caso de ejemplo

- Para ejemplificar las consultas utilizaremos el Stack Exchange Data Explorer de Stack Overflow:
<https://data.stackexchange.com/stackoverflow/query/new>
- Utiliza SQL Server
- Tiene datos de los posts hechos en Stack Overflow, sus respuestas e información adicional como comentarios, etiquetas (tags) y puntajes

StackExchange



question
title

SQL Server String Concatenation with Null

[Ask Question](#)

question
score

64



14

I am creating a computed column across fields of which some are potentially null. **question body**

The problem is that if any of those fields is null, the entire computed column will be null. I understand from the Microsoft documentation that this is expected and can be turned off via the setting SET CONCAT_NULL_YIELDS_NULL. However, there I don't want to change this default behavior because I don't know its implications on other parts of SQL Server.

Is there a way for me to just check if a column is null and only append its contents within the computed column formula if its not null?

[sql-server](#) [null](#) [string-concatenation](#) [calculated-columns](#)

question tags **user display name**

[share](#) [improve this question](#)

asked May 26 '10 at 21:05

Alex
30.9k • 65 • 223 • 317

[add a comment](#)

9 Answers

[active](#)
[oldest](#)
[votes](#)

answer
score

110



You can use `ISNULL(...)` **question body**

```
SET @Concatenated = ISNULL(@Column1, '') + ISNULL(@Column2, '')
```

If the value of the column/expression is indeed NULL, then the second value specified (here: empty string) will be used instead.

[share](#) [improve this answer](#)

**user
display name**

answered May 26 '10 at 21:07

marc_s
532k • 116 • 1034 • 1194

comment
score

13

comment

"Coalesce" is the ANSI-standard function name, but ISNULL is easier to spell. – Philip Kelley May 26 '10 at 21:08 **user display name**

1 And ISNULL seems to be a tad faster on SQL Server, too - so if you want to use it in a function that concatenates strings into a computed column, you might forgo the ANSI standard and opt for speed (see Adam Machanic: sqlblog.com/blogs/adam_machanic/archive/2006/07/12/...) – marc_s May 26 '10 at 21:15

asked 7 years, 10 months ago

viewed 102,669 times **question view count**

active 9 months ago

BLOG



Quantum Computing Site Launches with the Help of Strangeworks

Looking for a job?

Senior 3D Engineer

Envelope • New York, NY

\$100K - \$150K • REMOTE

[sql](#) [javascript](#)

Java Developer

Wallethub • Washington, DC

REMOTE

[java](#) [spring](#)

Java Developer - Best practices oriented

Almundo • Retiro, Argentina

RELOCATION

[mongodb](#) [java](#)

DevOps Engineer - Remote

Peak Games • No office location

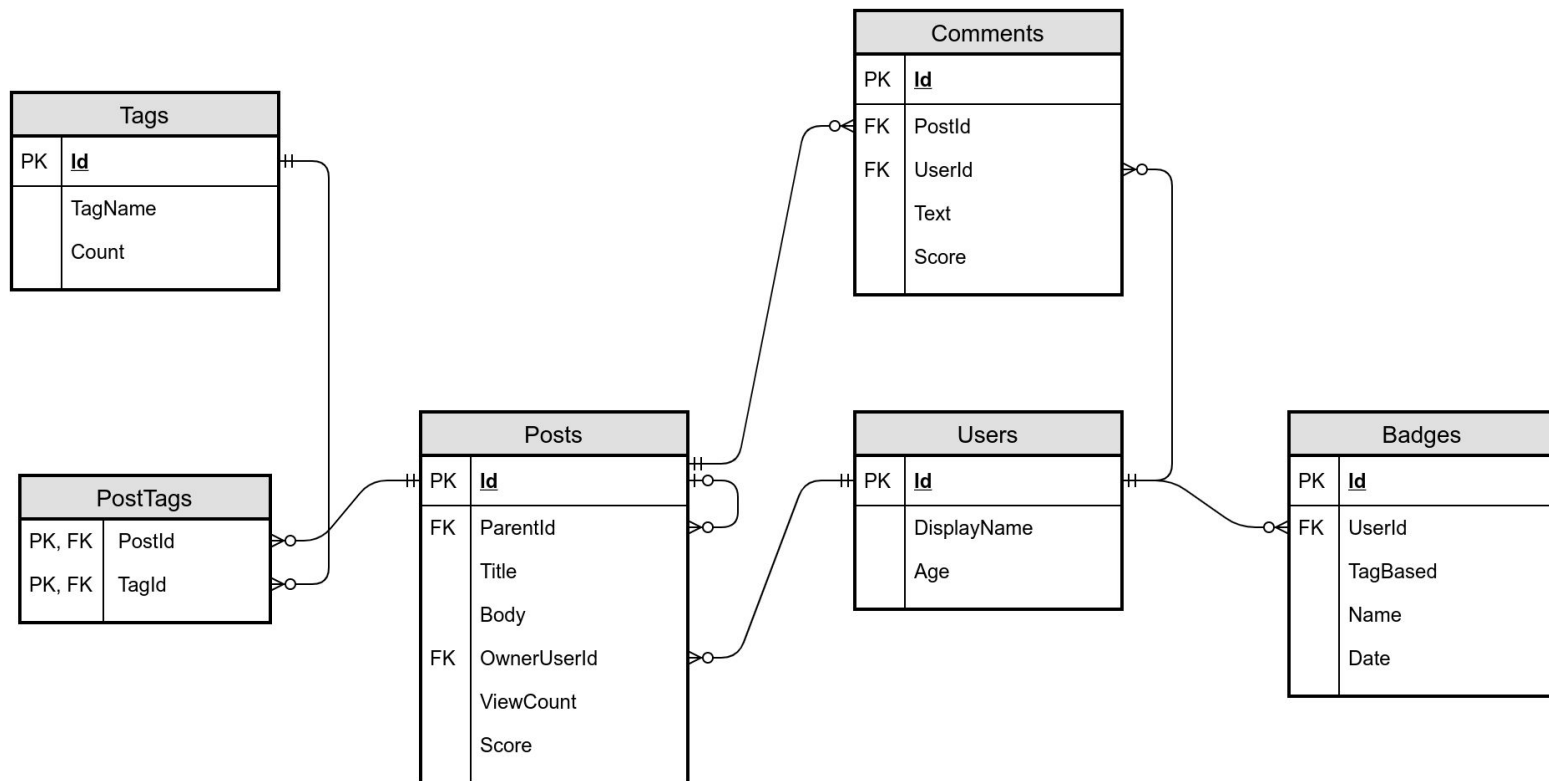
REMOTE

[linux](#) [amazon-web-services](#)

Linked

0 How can I avoid my selected row being a

Caso de ejemplo - Diagrama de tablas



DML - SELECT

- El comando para efectuar consultas en SQL es **SELECT**:

```
SELECT [ DISTINCT ] columna1 [ , ... , columnaN ]
FROM tabla1 [ , ... tablaN ]
[ WHERE condicion ] ;
```

- Es equivalente* a un producto cartesiano, selección y proyección

$\pi_{columna1, \dots, columnaN} (\sigma_{condicion} (tabla1 \bowtie \dots \bowtie tablaN))$

* Si se usa DISTINCT

DML - Espacios y Comentarios

- No hay problemas en poner varios espacios, tabs o incluso saltos de líneas en el comando
- Un comando SQL finaliza con un ; pero no es obligatorio si se ejecuta un único comando en la consulta
- Se pueden poner comentarios de línea con dos guiones
 - No hay bloques de comentario

```
-- Comentario que ayude a la memoria
```

DML - DISTINCT

- El **DISTINCT** hace que no se devuelvan filas repetidas
- Por defecto, se comporta distinto a álgebra relacional
- Aplica a la fila completa, no se puede hacer por solo una de las columnas devueltas

DML - Expresiones

- Se permite devolver una lista de expresiones, que pueden ser:
 - Nombres de columnas: se devuelve el valor para cada fila. Se puede usar nombre_tabla.nombre_columna ante ambigüedad
 - Valores constantes
 - Funciones con o sin parámetros, que son a su vez expresiones
 - Un caso especial son las funciones de agregación
 - Operadores aplicados a expresiones
- Si se pone un asterisco (**SELECT ***) se devuelven todas las columnas de la/s tabla/s

DML - Operadores

- Para textos tenemos el operador de concatenación ||

```
apellido || ', ' || nombre -- Román, Lucas
```

- Para valores numéricos los operadores matemáticos + - * /
 - Las fechas pueden restarse entre sí para obtener diferencias de días, o se les puede sumar días como enteros

```
campo_fecha + 7 -- Sumar una semana
```

- Están los operadores lógicos **AND**, **OR** y **NOT**
- Y los de comparación: =, <>, >, >=, < y <=

DML - Rangos

- Para buscar que un valor esté entre dos rangos está el operador **BETWEEN**

```
expresion BETWEEN rango1 AND rango2
```

- Es equivalente a:

```
expresion >= rango1 AND expresion <= rango2
```

DML - Comparación con nulos

- El nulo se toma como desconocido, entonces no se puede comparar con operadores normales

```
expresion = NULL -- Siempre devuelve falso
```



```
expresion <> NULL -- También siempre devuelve falso
```



- Se debe usar el operador **IS** (o **IS NOT** para el contrario)

```
expresion IS NULL -- Verdadero si expresion es nula
```

```
expresion IS NOT NULL -- Verdadero si no es nula
```

DML - Funciones de conversión de datos

- Existen distintas funciones de conversión: `to_char`, `to_date`, `to_number`, `to_timestamp`

```
to_char(padron, 'fm000000') -- 6 dígitos, 0s a la izq
```

- Se puede extraer un subcampo de una fecha

```
EXTRACT (DAY FROM current_date)
```

```
DAY(current_date) -- En MS SQL, no standard
```


DML - CAST

- Si bien no es una función, se puede castear entre tipos de dato

```
CAST( codigo AS CHAR(2) ) -- no agrega 0s a la izq
```

- Luego del **AS**, se indica un tipo de dato de SQL

DML - Otras funciones

- Numéricas: **ABS**, **ROUND**, **TRUNC**, **LN**, **LOG**, **EXP**, **SQRT**, etc

```
round(0.345, 2) -- 0.35
```

- Strings: **CHARACTER_LENGTH**, **TRIM**, **REPLACE**, **SUBSTR**, **LOWER**, **UPPER**

```
upper('HaCkEr') -- HACKER
```

- De tiempo: **CURRENT_TIME**, **CURRENT_DATE**

DML - Comparación con patrones

- Para comparar un string con patrones se utiliza **LIKE**
 - % se usa para cualquier cadena de 0 o más caracteres
 - _ para un caracter cualquiera

```
nombre LIKE 'A%o' -- Alejandro, Arnaldo, etc...
```

- Si no queremos que cumpla un patrón, usamos **NOT LIKE**
- Otros operadores de patrones:
 - **ILIKE**: Similar al **LIKE** pero case-insensitive (no es SQL standard)
 - **SIMILAR TO**: Más parecido a expresiones regulares

DML - Expresiones condicionales

- Con la expresión **CASE** podemos hacer que el valor dependa de condiciones:

```
CASE WHEN condicion1 THEN valor1  
      [ WHEN condicion2 THEN valor2 (...) ]  
      [ ELSE valor_por_defecto ]  
END
```

```
CASE WHEN intercambio = TRUE   THEN 'De intercambio'  
      WHEN intercambio = FALSE THEN 'Propio'  
      ELSE 'Indeterminado'  
END AS "tipo de estudiante"
```

DML - Condición del WHERE

- Es una expresión que debe ser de tipo lógico:

```
WHERE monto < (total / 2) -- menos de la mitad
```

- Se devolverán las filas en las cuales al evaluar la condición, el resultado devuelva el valor VERDADERO
 - No quiere decir que se evalúe la condición fila a fila, el procedimiento puede ser otro, pero con un resultado equivalente
- Si no se pone condición, se devuelven todas las filas

DML - Alias

- Se le puede poner un alias a las columnas devueltas

```
apellido || ', ' || nombre AS "Ape y Nom"
```

- Y también a las tablas

```
FROM tabla [ AS ] t WHERE t.campo = 1
```

- Es necesario usarlo cuando se usa más de una vez una tabla. También se pueden renombrar sus columnas

```
FROM tabla AS t1(a1, b1) , tabla AS t2(a2, b2)
```

DML - Ejercicio 1

Liste los nombres de badges que otorga StackOverflow y que no estén basados en tags (Tagbased = 0)

DML - Ejercicio 1

Liste los nombres de badges que otorga StackOverflow y que no estén basados en tags (Tagbased = 0)

```
SELECT DISTINCT Name  
FROM Badges  
WHERE TagBased = 0;
```


DML - Ejercicio 2

Liste los tags que utilizó el usuario "Jon Skeet"

DML - Ejercicio 2

Liste los tags que utilizó el usuario "Jon Skeet"

```
SELECT DISTINCT t.TagName
FROM Users u, Posts p, PostTags pt, Tags t
WHERE pt.PostId = p.Id AND pt.TagId = t.Id
AND u.Id = p.OwnerUserId
AND u.DisplayName = 'Jon Skeet';
```

DML - Ordenamiento

- Se puede ordenar el resultado de una consulta por una o más expresiones
 - Se ordena por la primera, y si hay empates por la segunda, etc
 - Se puede definir órdenes ascendentes o descendentes de cada expresión

```
ORDER BY expr1 [ ASC | DESC ]  
[ , (...) , exprN [ ASC | DESC ] ]
```

- Si se usan números como expresiones, se ordena por las columnas devueltas en el **SELECT** (numeradas desde 1)

DML - Limitar filas devueltas

- Se puede solicitar que no se devuelvan mas de N filas

```
FETCH FIRST cantidad_filas ROWS ONLY
```

- Distintos motores tienen distintas formas de hacerlo
 - **LIMIT** *cantidad_filas*
 - **SELECT TOP** *cantidad_filas* (...)
 - Otros Ejemplos:
<https://www.jooq.org/doc/3.10/manual/sql-building/sql-statements/select-statement/limit-clause/>
- Debería venir siempre despues de un **ORDER BY**
 - O ejecuciones diferentes podrían tener distintos resultados

DML - Paginado

- Se pueden saltar M filas previo a devolver las N filas

```
OFFSET filas_saltear ROWS  
FETCH FIRST cantidad_filas ROWS ONLY
```

- Esto permite manejar paginado de los datos



DML - Ejercicio 3

Para las preguntas hechas desde el 01/01/2024 en adelante, muestre las 10 que recibieron más visitas, indicando su título, la fecha en la que se hicieron y la cantidad de visitas que recibieron

DML - Ejercicio 3

Para las preguntas hechas desde el 01/01/2024 en adelante, muestre las 10 que recibieron más visitas, indicando su título, la fecha en la que se hicieron y la cantidad de visitas que recibieron

```
SELECT Title, CreationDate, ViewCount
FROM Posts
WHERE CreationDate >= '2024-01-01'
AND ParentId IS NULL
ORDER BY ViewCount DESC
OFFSET 0 ROWS FETCH FIRST 10 ROWS ONLY;
```

DML - Joins

- Para hacer juntas se puede utilizar el operador **JOIN**

```
FROM tabla1 INNER JOIN tabla2 ON condicion_de_join
```

- Si ambas tablas tienen columnas de mismo nombre y se va a comparar por igualdad, se puede usar **USING**

```
FROM tabla1 INNER JOIN tabla2  
    USING (columna1 [, columna2, ... , columnaN ])
```

- También está el **NATURAL JOIN**, con su condición implícita

```
FROM tabla1 NATURAL JOIN tabla2
```


DML - Ejercicio 4

Utilizando el JOIN, devuelva los posts que utilizan tags 'relational' y 'entity-relationship', indicando el id del post, el título y la cantidad de vistas.

DML - Ejercicio 4

Utilizando el JOIN, devuelva los posts que utilizan tags 'relational' y 'entity-relationship', indicando el id del post, el título y la cantidad de vistas.

```
SELECT DISTINCT p.Id, p.Title, p.ViewCount
FROM Tags t1 INNER JOIN PostTags pt1 ON t1.Id = pt1.TagId
INNER JOIN Posts p ON pt1.PostId = p.Id
INNER JOIN PostTags pt2 ON p.Id = pt2.PostId
INNER JOIN Tags t2 ON pt2.TagId = t2.Id
WHERE t1.TagName = 'relational'
AND t2.TagName = 'entity-relationship';
```

DML - Joins externos

- El Join externo devuelve siempre todas las filas de alguna de las tablas, aún cuando no se vincule con ninguna fila de la otra
 - En ese caso, los valores de las columnas de la otra tabla se devolverán siempre como nulos

```
FROM tabla1 LEFT OUTER JOIN tabla2 ON condicion
```

```
FROM tabla1 RIGHT OUTER JOIN tabla2 ON condicion
```

```
FROM tabla1 FULL OUTER JOIN tabla2 ON condicion
```

DML - Operadores de conjuntos

- Los tres operadores de conjunto están presentes en SQL

```
SELECT ... UNION [ ALL ] SELECT ....;
```

```
SELECT ... INTERSECT [ ALL ] SELECT ....;
```

```
SELECT ... EXCEPT [ ALL ] SELECT ....;
```

- Las consultas deben tener misma cantidad y tipo de datos de columnas (unión compatible)
- La opción **ALL** hace que la respuesta sea un set (igual que álgebra relacional). Si no, es un multiset (con duplicados)

Consultas Anidadas

DML - Consultas anidadas

- El resultado de una consulta SQL es una tabla
- Entonces ese resultado puede ser utilizado por una consulta principal
 - Se las llama subconsultas o consultas anidadas
- Por ejemplo como un escalar

```
SELECT ... FROM ...  
WHERE campo1 = (SELECT campo1 FROM ...);  
                -- Debe devolver solo 1 fila o da error
```

```
SELECT expr1, ..., (SELECT campo1 FROM ...), ...  
FROM ...;        -- Debe devolver solo 1 fila o da error
```

DML - Consultas anidadas

- Pueden utilizarse subconsultas en el **FROM** como si fueran una tabla

```
SELECT ...  
FROM ...  
    , (SELECT ... FROM ...) nombre_tabla  
    -- Puede devolver varias filas y columnas  
    , ...
```

- Se les define un alias para poder luego referenciar en joins, condiciones, columnas a devolver, etc...

DML - Operadores de conjunto

- Con operadores de conjunto se puede ver si una condición se cumple contra todos o contra al menos uno (alguno) de los elementos del conjunto
 - Sólo para operadores de comparación

```
expression operator [ SOME | ANY | ALL ] (SELECT ...)
```

```
WHERE padron = ANY (SELECT padron FROM notas
                     WHERE notas.nota = 10)
```


DML - Operador IN

- Se puede usar para comparar contra un conjunto fijo

```
categoria IN ('A', 'C') -- Similar a un OR
```

- O contra el resultado de una subconsulta

```
expr1 IN (SELECT expr2 FROM ...) -- = ANY
```

```
expr1 NOT IN (SELECT expr2 FROM ...) -- <> ALL
```

DML - Operador IN sobre varias expresiones

- Es un feature opcional pero que suele estar implementado

```
(expr1, expr2) IN (SELECT expr3, expr4 FROM ...)
```

- Revisa que el par de valores esté devuelto en al menos una fila de la subconsulta

DML - Consultas correlacionadas

- Una consulta interna puede hacer referencia a tablas de la consulta externa
 - no al revés!

```
FROM tabla1 t1 WHERE expr1 IN  
    ( SELECT expr2 FROM tabla2 t2  
      WHERE t2.columna2 = t1.columna1 )
```

- Cuando ocurre decimos que las consultas están correlacionadas
- El costo de este tipo de subconsultas suele ser mucho más alto

DML - comparar valores vs comparar conjuntos

- Siempre hay que tener cuidado cuando comparamos valores vs conjuntos de valores
 - Devolver el padrón de los alumnos que no se llamen Lucas

```
SELECT padron FROM alumnos  
WHERE nombre <> 'Lucas';
```



- Devolver el padrón de los alumnos que no tienen nota en materias del departamento de código 75

```
SELECT DISTINCT padron FROM notas  
WHERE codigo <> 75;
```



DML - comparar valores vs comparar conjuntos

- Devolver el padrón de los alumnos que no tienen nota en materias del departamento de código 75

```
SELECT DISTINCT padron FROM notas  
WHERE codigo <> 75;
```



- Esto devuelve los alumnos que tienen al menos una nota de un departamento que no es el 75
- Si tiene una nota del 75 y otra del 71, lo devuelve por esta última, cuando no debería
- Necesitamos que ninguna nota sea del departamento 75, con una sola no alcanza

DML - comparar valores vs comparar conjuntos

- Devolver el padrón de los alumnos que no tienen nota en materias del departamento de código 75

```
SELECT DISTINCT padron FROM notas  
WHERE codigo <> 75;
```



```
SELECT padron FROM alumnos  
WHERE padron NOT IN ( -- o <> ALL  
    SELECT padron FROM notas  
    WHERE codigo = 75  
);
```



DML - Ejercicio 5

Utilizando subconsultas, devuelva los posts que utilizan el tag 'relational' y no utilizan el tag 'entity-relationship', indicando el id del post, el título y la cantidad de vistas.

DML - Ejercicio 5

Utilizando subconsultas, devuelva los posts que utilizan el tag 'relational' y no utilizan el tag 'entity-relationship', indicando el id del post, el título y la cantidad de vistas.

```
SELECT DISTINCT p.Id, p.Title, p.ViewCount
FROM Tags t1 INNER JOIN PostTags pt1 ON t1.Id = pt1.TagId
      INNER JOIN Posts p ON pt1.PostId = p.Id
WHERE t1.TagName = 'relational'
AND p.Id NOT IN (SELECT pt2.PostId FROM PostTags pt2
      INNER JOIN Tags t2 ON pt2.TagId = t2.Id
      AND t2.TagName = 'entity-relationship');
```


DML - Ejercicio 6

Encuentre para cada Tag la/s pregunta/s que haya/n tenido la mayor cantidad de vistas, indicando el nombre del tag y el título y cantidad de vistas de las preguntas.

Sólo muestre tags de preguntas con al menos 2 millones de vistas

DML - Ejercicio 6

Encuentre para cada Tag la/s pregunta/s que haya/n tenido la mayor cantidad de vistas, indicando el nombre del tag y el título y cantidad de vistas de las preguntas.

Sólo muestre tags de preguntas con al menos 2 millones de vistas

```
SELECT t.TagName, preg.Title, preg.ViewCount
FROM Tags t INNER JOIN PostTags pt ON t.Id = pt.TagId
      INNER JOIN Posts preg ON pt.PostId = preg.Id
WHERE preg.ParentId IS NULL AND preg.ViewCount >= 2000000
AND preg.ViewCount >= ALL (SELECT ViewCount
      FROM PostTags pt2 INNER JOIN Posts p2
      ON pt2.PostId = p2.Id
      WHERE pt2.TagId = t.Id AND pt2.ParentId IS NULL);
```

DML - Operador EXISTS

- El operador **EXISTS** devuelve verdadero cuando la subconsulta devuelve al menos una fila
 - Se suele usar en consultas correlacionadas

```
[NOT] EXISTS (SELECT ...)
```

```
FROM alumnos a WHERE NOT EXISTS (  
    SELECT 1  -- No importa lo que devuelva, puede ser *  
    FROM notas n  
    WHERE n.padron = a.padron  
    AND n.nota = 10)
```

Agregación

DML - Funciones de Agregación

- Ciertas consultas, como el promedio de notas de un alumno, precisan los valores de un conjunto de filas
- Para resolverlas se utilizan funciones de agregación, que operan sobre un grupo de filas y para el grupo devuelven un único valor
 - **MAX**(expr) y **MIN**(expr): Devuelven el valor máximo o mínimo de la expresión entre las filas del grupo
 - **SUM**(expr) y **AVG**(expr): Devuelven la suma o el promedio del valor de la expresión entre las filas del grupo
 - **COUNT**([**DISTINCT**] expr | *): Con * devuelve la cantidad de filas del grupo. Si no, devuelve la cantidad de filas en las que la expresión no es nula. Si se utiliza la opción **DISTINCT** no cuenta más de una vez valores repetidos

DML - Ejercicio 7 y 8

Cuenta la cantidad de usuarios existentes en la base

Cuenta la cantidad de Posts que son preguntas

DML - Ejercicio 7 y 8

Cuenta la cantidad de usuarios existentes en la base

```
SELECT COUNT(*) FROM Users;
```

Cuenta la cantidad de Posts que son preguntas

```
SELECT COUNT(*) FROM Posts p  
WHERE p.ParentId IS NULL;
```

DML - Funciones de Agregación

- Todas las filas que hubieran sido devueltas forman un grupo, y se devuelve un único valor para el grupo
- Usar una función de agregación entonces modifica la cantidad de filas devueltas
- No pueden utilizarse como condición del **WHERE**, ya que esta se evalúa fila a fila

DML - Agrupamiento

- Se puede querer usar funciones de agregación pero devolviendo más de un valor en la consulta

Torneo	Año	Ganador	Premio
Australian Open	2023	Novak Djokovic	2,975,000
French Open	2023	Novak Djokovic	2,300,000
Wimbledon	2023	Carlos Alcaraz	2,350,000
US OPeN	2023	Novak Djokovic	3,600,000
Australian Open	2024	Jack Sinner	3,150,000
French Open	2024	Carlos Alcaraz	2,400,000
Wimbledon	2024	Carlos Alcaraz	2,700,000

- ¿Cuántas veces ganó cada tenista?
- ¿Cuánto dinero acumuló cada uno?
- ¿Cuántos tenistas ganaron cada torneo?

DML - Agrupamiento

- Se debe definir cómo se van a agrupar las filas y devolver resultados de agregación por cada grupo
 - Si quiero datos por tenista, agrupo por “ganador” y se formarán tantos grupos como tenistas ganadores haya
 - Si quiero datos por torneo, agrupo por torneo
- Las filas se agrupan según el valor de una o más expresiones utilizando **GROUP BY**, luego del **WHERE** y antes del **ORDER BY**

```
GROUP BY expresion1 [ , expresion2 ... , expresionN ]
```

- Filas con mismo valor en todas las expresiones estarán en un mismo grupo

DML - Agrupamiento

- Agrupando por ganador, se forman tres grupos

Torneo	Año	Ganador	Premio
Australian Open	2023	Novak Djokovic	2,975,000
French Open	2023	Novak Djokovic	2,300,000
US OPeN	2023	Novak Djokovic	3,600,000
Wimbledon	2023	Carlos Alcaraz	2,350,000
French Open	2024	Carlos Alcaraz	2,400,000
Wimbledon	2024	Carlos Alcaraz	2,700,000
Australian Open	2024	Jack Sinner	3,150,000

Ganó 3 veces, acumuló
8,875,000

Ganó 3 veces, acumuló
7,450,000

Ganó 1 vez, acumuló
3,150,000

- Al haber 3 grupos, luego de agrupar se devolverán 3 filas

DML - Agrupamiento

- Agrupando por ganador, se forman tres grupos

```
SELECT Ganador, COUNT(*) AS "Ganados"  
      , SUM(Premio) AS "Acumulado"  
FROM Torneos  
GROUP BY Ganador;
```

Ganador	Ganados	Acumulado
Novak Djokovic	3	8,875,000
Carlos Alcaraz	3	7,450,000
Jack Sinner	1	3,150,000

DML - Agrupamiento - acceso a columnas

- Al estar devolviendo una fila por grupo, no se puede devolver directamente columnas por las que no se haya agrupado

```
SELECT Ganador, Año  
FROM Torneos GROUP BY Ganador;
```



- Las filas de un mismo ganador podrían tener distintos años, como pasa con Carlos Alcaraz
- Importante: Sí se pueden acceder mediante una función de agregación, ya que el valor devuelto dependera de todos los valores de las columnas entre las filas del mismo grupo

DML - Agrupamiento por clave primaria

- Algunos motores permiten acceder a todas las columnas de una tabla si se agrupó por la clave primaria de la tabla
 - Todas las filas del grupo tendrán el mismo valor en esas columnas

```
SELECT a.padron, a.nombre, ...  
FROM alumnos a, ....  
GROUP BY a.padron, ...
```

Se arma un grupo por cada padrón y todas las filas de cada grupo tendrán el mismo nombre ya que padrón es PK de alumnos

- Si el motor no lo permite, agrupar por la clave y también por las otras columnas que se quieren devolver

DML - Agrupamiento - Otro ejemplo

- Agrupando por torneo, se forman cuatro grupos

```
SELECT Torneo, COUNT(DISTINCT Ganador) AS "Cant. Ganadores"  
FROM Torneos GROUP BY Torneo;
```

Torneo	Cant. Ganadores
Australian Open	2
French Open	2
US OPen	1
Wimbledon	1



Las dos veces lo ganó el mismo tenista, Carlos Alcaraz

DML - Filtrado de grupos

- Se puede querer no devolver todos los grupos, sino sólo algunos que cumplan cierta condición
- Se utiliza la cláusula **HAVING** luego del **GROUP BY**, que de modo similar al **WHERE** tiene una condición a cumplir por el grupo
 - En la condición se tiene acceso a los datos del grupo, pero no de las filas
 - Se devuelven grupos cuya condición se evalúe como VERDADERA

```
GROUP BY expresion1 [ , expresion2 ... , expresionN ]  
HAVING condicion
```


DML - Cláusula HAVING

- Si busco los que ganaron más de un torneo, Sinner no es devuelto

```
SELECT Ganador, COUNT(*) AS "Ganados"  
      , SUM(Premio) AS "Acumulado"  
FROM Torneos  
GROUP BY Ganador  
HAVING COUNT(*) > 1;
```

Ganador	Ganados	Acumulado
Novak Djokovic	3	8,875,000
Carlos Alcaraz	3	7,450,000

DML - Ejercicio 9

Muestre los nombres de los 10 usuarios cuyas respuestas a preguntas taggeadas con 'C#' acumulan mayor puntaje

DML - Ejercicio 9

Muestre los nombres de los 10 usuarios cuyas respuestas a preguntas taggeadas con 'C#' acumulan mayor puntaje

```
SELECT u.DisplayName
FROM Users u INNER JOIN Posts resp
  ON u.Id = resp.OwnerUserId
  INNER JOIN Posts preg ON resp.ParentId = preg.Id
  INNER JOIN PostsTags pt ON pt.PostId = preg.Id
  INNER JOIN Tags t ON t.Id = pt.TagId
WHERE t.TagName = 'C#'
GROUP u.Id, u.DisplayName
ORDER BY SUM(resp.Score) DESC
OFFSET 0 ROWS FETCH FIRST 10 ROWS ONLY;
```

DML - Ejercicio 10

Encuentre el Id, el título y la cantidad de vistas de la/s pregunta/s que haya/n tenido la mayor cantidad de vistas.

DML - Ejercicio 10

Encuentre el Id, el título y la cantidad de vistas de la/s pregunta/s que haya/n tenido la mayor cantidad de vistas.

```
SELECT preg.Id, preg.Title, preg.ViewCount
FROM Posts preg
WHERE preg.parentId IS NULL
AND preg.ViewCount = ( SELECT MAX(preg2.ViewCount)
                       FROM Posts preg2
                       WHERE preg2.parentId IS NULL );
```

DML - Ejercicio 11

Liste los tags cuyo primer uso ocurrió después del 01/01/2018.

DML - Ejercicio 11

Liste los tags cuyo primer uso ocurrió después del 01/01/2018.

```
SELECT t.TagName
FROM Tags t, PostTags pt, Posts p
WHERE t.Id = pt.TagId
AND pt.PostId = p.Id
GROUP BY t.TagName
HAVING MIN(p.CreationDate) >= '2018-01-01';
```

ABMs

ABM - Inserción

- El comando **INSERT** se utiliza para insertar filas en una tabla, pudiendo insertarse varias filas a la vez

```
INSERT INTO nombre_tabla  
    [ ( nombre_col1 , ... , nombre_colN ) ]  
VALUES ( valor1, valor2, ... , valorN )  
    [, ( valor1, valor2, ... , valorN ) ... ]  
;
```

- Se revisan todas las reglas de integridad
- Si alguna inserción falla, se cancela el total de las inserciones

ABM - Inserción

- También puede insertarse el resultado de una consulta
 - Si devuelve varias filas, se intenta insertar todas

```
INSERT INTO nombre_tabla  
  [ ( nombre_col1 , ... , nombre_colN ) ]  
SELECT ...;
```

- Con la opción **RETURNING** (no standard) se permite devolver valores de las filas insertadas
 - Útil para ver qué ids secuenciales se generaron!

ABM - Modificación

- Para actualizar fila/s de la tabla se utiliza **UPDATE**

```
UPDATE nombre_tabla SET  
    columna1 = expresion1  
    [ , columna2 = expresion2 ... ]  
    [ WHERE condicion ];
```

- Sólomente se actualizan las filas que cumplan la condición
- En caso de no indicarse condición se actualiza toda la tabla

ABM - Borrado

- El borrado se hace con el comando **DELETE**

```
DELETE FROM nombre_tabla  
[ WHERE condicion ];
```

- Siempre tener cuidado con poner la condición!

ABM - Transacciones

- Cuando queremos que varias operaciones se ejecuten únicamente en su totalidad, usamos transacciones, indicando con comandos el inicio y fin de la transacción

```
BEGIN TRANSACTION;  
comando1;  
[ comando2; ... comandoN ]  
COMMIT;
```


- Se pueden usar niveles de aislamiento (lo veremos más adelante)
- El comando **ROLLBACK** permite volver atrás los cambios
- Se pueden definir **SAVEPOINTS** hacia los que volver

Vistas

Vista

- Las vistas son tablas virtuales
 - Se las puede acceder como si fueran una tabla más
 - No existe físicamente como una tabla, sus datos surgen como una consulta sobre una o más tablas

```
CREATE VIEW nombre_vista  
  [ (columna1, columna2, ... , columnaN) ]  
AS consulta_SQL;
```

- 
- La consulta no debería tener ORDER BY, aunque algunos SGBD lo permiten como orden por defecto

Usos de vista

- Seguridad
 - No mostrar ciertas filas / columnas
- Reducción de complejidad
 - Representar consultas complejas
- Cambios de representación de datos
 - Usar vistas con versión vieja

Actualización de datos en vistas

- Algunos gestores permiten actualizaciones en vistas, siguiendo algunas restricciones:
 - La vista tiene una única tabla en el **FROM**
 - La vista no usa operaciones de conjunto
 - La vista no usa agrupamiento
 - La vista no limita la cantidad de resultados
 - Sólo se pueden actualizar valores de columnas accedidas directamente
 - Las restricciones de la tabla se deben cumplir, o la actualización fallará

Problemas con filas que quedan fuera de la vista

- Si la consulta de la vista tiene un **WHERE** y se crean o modifican filas, pueden quedar sin cumplir la condición y quedar fuera de la vista
- PostgreSQL permite impedir este tipo de operaciones si se define la vista con la opción **CHECK** (no standard)

```
CREATE VIEW nombre_vista  
  [ (columna1, columna2, ... , columnaN) ]  
  AS consulta_SQL  
  WITH CHECK OPTION;
```

WITH y WITH RECURSIVE

WITH

- La cláusula WITH (opcional en el estándar SQL) permite definir una consulta auxiliar temporal a utilizar durante una consulta
- Se puede pensar como una tabla temporal, que existe únicamente en la ejecución de la consulta
- Permite evitar repetir una estructura varias veces y asociar a un nombre para tener una consulta más clara
- Hay que usarlo con cuidado porque exceso de WITHs puede complejizar una consulta sencilla

WITH - Sintaxis y ejemplo

```
WITH alias1 [(col1,...,colN)] AS (subconsulta)  
    [, alias2 [(col1b,...,colNb)] AS (subconsulta2) ]  
consulta_principal
```

```
WITH aprobados AS (SELECT * FROM notas WHERE nota >= 4)  
SELECT padron  
FROM aprobados  
GROUP BY padron  
HAVING AVG(nota) >= ALL (SELECT AVG(nota)  
                        FROM aprobados  
                        GROUP BY padron);
```

WITH RECURSIVO

- La opción **RECURSIVE** permite que una subconsulta se referencie a si misma
- Esto es útil para resolver consultas iterativas

codigo	desde	hasta
444	París	Madrid
510	París	Roma
610	Brasilia	París
690	Brasilia	Montevideo
880	Roma	Amsterdam
900	Amsterdam	Oslo

- Teniendo la tabla “vuelos” que indican ciudades origen y destino, ¿a qué ciudades puedo llegar desde una en particular, haciendo todas las escalas que se necesiten?

WITH RECURSIVO

- Desde París podemos llegar a Madrid y Roma
- Desde Roma también a Amsterdam
- Y desde Amsterdam también a Oslo

codigo	desde	hasta
444	París	Madrid
510	París	Roma
610	Brasilia	París
690	Brasilia	Montevideo
880	Roma	Amsterdam
900	Amsterdam	Oslo

- La cantidad de iteraciones está dada por los datos, y no podemos saberla de antemano

WITH RECURSIVO - Estructura

```
WITH RECURSIVE alias [(col1, ..., colN)]  
AS (  
    SELECT ... FROM ...  
    UNION  
    SELECT ... FROM ...  
)  
SELECT ... FROM ... ;
```

Subconsulta inicial
fija

Subconsulta recursiva,
que utiliza a *alias*,
referenciandose

- Se comienza con la subconsulta fija y se ejecuta iterativamente la unión con la subconsulta recursiva hasta que no se agreguen nuevas filas en una iteración

WITH RECURSIVO - Ejemplo

```
WITH RECURSIVE alcanzables AS (  
    SELECT hasta FROM vuelos  
        WHERE desde = 'París'  
    UNION  
    SELECT v.hasta  
        FROM alcanzables a, vuelos v  
        WHERE a.hasta = v.desde  
)  
SELECT * FROM alcanzables;
```

hasta
Madrid
Roma
Amsterdam
Oslo

Control de acceso a los datos

DCL - Data Control Language

DCL - Necesidades de control

- En base de datos multiusuarios, muchas veces es necesario restringir el acceso a los datos
- Distintos usuarios deben tener acceso a distintos subconjuntos de los datos
 - No todas las tablas
 - No todas las columnas
 - No todas las filas
- También se debe poder restringir operaciones a realizar
 - No todos deberian poder modificar / crear / insertar

DCL - Gestión de accesos

- Bastantes diferencias entre SGBD
 - Ej: MSSQL: puede usar Active Directory
- En general se puede restringir desde qué redes se aceptan conexiones, en algún archivo de configuración
- Una vez aceptada la conexión, los accesos se gestionan por usuario y por roles a los que pertenecen los usuarios
- En PostgreSQL el usuario es un caso particular de un rol

DCL - Creación de usuarios en PostgreSQL

- Crear un usuario es crear un rol con login

```
CREATE USER nombre_usuario WITH PASSWORD 'pass';
```

```
CREATE ROLE nombre_usuario WITH LOGIN PASSWORD 'pass';
```

- Se pueden quitar usuarios/roles con el comando **DROP**

```
DROP ROLE nombre_usuario;
```

DCL - Creación y asignación de roles

- Se recomienda gestionar los permisos en roles y asignarle roles a usuarios, para agrupar perfiles comunes

```
CREATE ROLE nombre_rol;
```

```
GRANT nombre_rol TO nombre_usuario;
```

- Se pueden desasignar roles a los usuarios

```
REVOKE nombre_rol FROM nombre_usuario;
```

DCL - Privilegios sobre tablas

- El comando **GRANT** permite dar privilegios a roles:

- SELECT
- INSERT
- UPDATE
- DELETE
- TRIGGER
- CREATE



```
GRANT privilegio1 [, privilegio2, ... , privilegioN ]  
ON [ nombre_tabla | DATABASE nombre_base_de_datos ]  
TO nombre_rol  
[ WITH GRANT OPTION ];
```

- La opción **GRANT OPTION** permite que el rol también pueda dar los privilegios a otros roles utilizando **GRANT**

DCL - Quita de privilegios

- Para quitar privilegios se utiliza el comando **REVOKE**

```
REVOKE [ GRANT OPTION FOR ]  
    privilegio1 [, privilegio2, ... , privilegioN ]  
    ON [ nombre_tabla | DATABASE nombre_base_de_datos ]  
    FROM nombre_rol;
```

- La opción **GRANT OPTION** no quita los privilegios, sino la capacidad de otorgarlos a otros roles

DCL - Uso de vistas para restringir acceso

- Lo anterior da privilegios a todos los datos de las tablas
- Para restringir a ciertas columnas o ciertas filas, se pueden utilizar vistas de SQL y dar permisos sobre ellas
 - No incluir todas las columnas
 - No incluir todas las filas

SQL Avanzado

Cosas que nos quedaron en el tintero

SQL Avanzado - Cosas que quedan fuera del alcance

- Stored Procedures y Triggers
- Funciones de ventana
- Vistas materializadas
- EXPLAIN - plan de ejecución de una consulta
 - En la clase de costos lo veremos un poco!
- Tipos de dato y funciones custom (incluso de agregación!)

Bibliografía

Bibliografía

[ELM16] Fundamentals of Database Systems, 7th Edition.

R. Elmasri, S. Navathe, 2016.

Capítulo 6, Capítulo 7

[SILB19] Database System Concepts, 7th Edition.

A. Silberschatz, H. Korth, S. Sudarshan, 2019.

Capítulo 3, Capítulo 4

[CONN15] Database Systems, a Practical Approach to Design, Implementation and Management, 6th Edition.

T. Connolly, C. Begg, 2015.

Capítulo 6, Capítulo 7, Capítulo 8

[GM09] Database Systems, The Complete Book, 2nd Edition.

H. García-Molina, J. Ullman, J. Widom, 2009.

Capítulo 6, Capítulo 7

Bibliografía

[SQL] ISO/IEC 9075:2011 Standard

Estándar ISO, 2011

Versión draft en

<http://www.wiscorp.com/sql20nn.zip> a la que se puede acceder también desde <http://modern-sql.com/standard>.

[SQLGRAM] SQL::2011 Foundation Grammar

Gramática de la Parte 2 del estándar

<https://jakewheat.github.io/sql-overview/sql-2011-foundation-grammar.html>.

[SQLDR] SQL Dialects Reference

Wikibooks

https://en.wikibooks.org/wiki/SQL_Dialects_Reference.

[SQLCOMP] Comparison of different SQL implementations

T. Arvin

<http://troels.arvin.dk/db/rdbms/>