Cápsulas SQL

Glosario SQL

- SQL (Structured Query Language): Lenguaje de programación que sirve para administrar y manipular bases de datos relacionales
- Base de datos: Conjunto organizado de información que se puede acceder y manipular de manera eficiente. La información en una base de datos se organiza en tablas, compuestas por registros distribuidos de forma columnar
- Tabla: Objeto que almacena datos de forma columnar
- Esquema: Colección de objetos que se utiliza para estructurar la información en una base de datos
- Columna: Estructura vertical de una tabla. Esta posee un tipo de dato específico
- Fila: Estructura horizontal de una tabla. Contiene datos relacionados con cada columna de la misma tabla
- Consulta: Solicitud de información a una base de datos

Etapa 1: Sintaxis Básica

Semana 1

https://docs.google.com/presentation/d/1J003pC2bwGklj3kVCj_FuEkmOBlhEcxEvX8TNi2o-vY/edit#slide=id.p

https://drive.google.com/file/d/1So-35N5VZh3f0-gBU5_5TMMNzNFuUnAb/view?usp=share_link

Qué es SQL? 👀

- Lenguaje de programación
- Bases de datos relacionales
- Creación de bases de datos; gestión de bases de datos; creación de tablas; definición de relaciones entre tablas; insertar registros en tablas; actualizar o eliminar registros; **consulta de datos**

Qué son las Cápsulas SQL? 🤔

- Programa de 12 semanas
- Aprender a extraer y transformar datos con SQL
- Desarrollo de un proyecto con SQL y otras herramientas

Qué es BigQuery? 📦

BigQuery

- Servicio de Google de almacenamiento
- Almacenar, explorar y procesar datos
- Compatible con distintas herramientas y plataformas

▼ Ejemplos

```
SELECT
  *
FROM
  `buda-data-science.master_entities.accounts_view`
LIMIT
  10
```

Semana 2

https://docs.google.com/presentation/d/1zLGVNv4G9nUe3CKXiSw0gvvzodyDJYxmFmJFPVtEvHo/edit?usp=share_link

https://drive.google.com/file/d/1Kzyp73wP6CD2OqDKg-0XFvs8zC_RxOV9/view?usp=share_link

Tipos de Datos 🌀

Tipo de valor que se puede almacenar en una columna de una tabla

- NUMERIC
  - INTEGER / INT64: Valores numéricos enteros
  - FLOAT / FLOAT64: Valores numéricos con componentes fraccionarios
- STRING
  - Cadena de caracteres
- DATE / DATETIME
  - DATE: Representan una fecha de calendario
  - DATETIME: Representa una fecha y una hora
- BOOLEAN
  - Tipo de dato lógico que devuelve TRUE, en caso de cumplir con la condición lógica planteada, o FALSE en caso contrario

Proposiciones Lógicas ☑️

Se utilizan con las cláusulas WHERE y HAVING para filtrar datos en una consulta

- Operadores de comparación
  - Igual a: `=`
  - Distinto a: `!= | <>`
  - Mayor que: `>`

    - Menor que: `<`
    - Mayor o igual que: `>=`
    - Menor o igual que: `<=`

- Operadores Lógicos
    - AND: Devuelve los registros que cumple con todas las condiciones planteadas
    - OR: Devuelve los registros que cumple al menos una de las condiciones planteadas
    - NOT: Devuelve registros si la condición propuesta es falsa
    - IN: Devuelve registros que estén dentro de la lista
    - BETWEEN: Devuelve registros que estén dentro de dos valores (números o fechas)
    - LIKE: Devuelve los registros que asemeja al texto seleccionado
    - IS NULL: Devuelve los registros en los que el campo es nulo
    - IS NOT NULL: Devuelve los registros en los que el campo no es nulo

DQL 💻

Data Query Language

Comandos SQL que se utilizan para recuperar datos de una base de datos

- SELECT
    - Para seleccionar columnas específicas de una tabla
    - Las columnas se separan por coma — `,`
    - Para seleccionar todas las columnas de una tabla se utiliza asterisco — `*`
- FROM
    - Para seleccionar la tabla desde donde se extraerán los datos
    - En BigQuery se usa `` `dataset_name.table_name` ``
- WHERE
    - Proposición(es) lógica(s) o condición(es)
    - Si se cumple la(s) condición(es), devuelve los valores que cumplieron esta(s)
    - Para añadir más condiciones se agrega AND o OR, según corresponda
- ORDER BY
    - Forma de ordenar tus resultados después del procesamiento
    - Las columnas por las cuales se quiere ordenar los resultados se separan con coma — `,`

- Los resultados se pueden ordenar, para cada columna, de forma ascendente (`ASC`) o descendente (`DESC`)

▼ Ejemplos

```
SELECT
  64+87 AS integer_field,
  63.3+87.7 AS float_field,
  'a3%' AS string_field,
  DATE("2022-07-15") AS date_field,
  DATETIME("2022-07-15") AS datetime_field,
  TRUE AS boolean_field
```

```
SELECT
  patabit_account_id,
  user_document_country_code,
  account_verification_level_flag
FROM
  `buda-data-science.master_entities.accounts_view`
WHERE
  user_document_country_code = 'cl'
  OR account_verification_level_flag = 1
ORDER BY
  user_document_country_code ASC,
  account_verification_level_flag DESC
LIMIT
  10
```

```
SELECT
  *
FROM
  `buda-data-science.master_entities.movements`
WHERE
  movement_state_ind = 'confirmed'
  AND movement_action_ind = 'deposit'
  AND movement_confirmation_dt BETWEEN "2023-01-01 00:00:00" AND "2023-03-31 23:59:59"
  AND (movement_origin_account_name_txt LIKE "%banco%" OR movement_origin_account_name_txt LIKE "%scotiabank%")
```

SQL SINTAXIS ATM

```
SELECT
  { * | fields_list}
FROM
  {schema_name.table_name}
WHERE
  {logical_statement}
```

```
ORDER BY
  {fields_list}
```

Semana 3

https://docs.google.com/presentation/d/1RZeBbFJ9E3THXiCJfJYGOGDRMkckg_Ztn0gWl_XO1Bw/edit#slide=id.p

https://drive.google.com/file/d/1dw9ROuYLTFsfi4uGctQIK8rxHGK7Gnrf/view?usp=share_link

Qué es una función de agregación? 🧮

- Se utilizan para realizar cálculos con salidas de datos numéricos
- Sumar, contar, promediar, obtener el valor máximo o mínimo
- **Se deben agregar las columnas que no han sido operadas** con `GROUP BY`

Funciones de Agregación 📏

- COUNT(* | field_name): Cuenta el numero de registros
- SUM(field_name): Suma los valores de una columna de tipo numérico
- AVG(field_name): Calcula el promedio de una columna de tipo numérico
- MAX(field_name): Para obtener el valor máximo de una columna de tipo numérico
- MIN(field_name): Para obtener el valor mínimo de una columna de tipo numérico

GROUP BY 👥

- Para agrupar filas de datos de acuerdo a una o más columnas de la tabla
- Genera grupos de datos de acuerdo a las columnas seleccionadas

- Se usan generalmente con las funciones de agregación mencionadas anteriormente

▼ Ejemplos

```
SELECT
  patabit_account_id,
  movement_action_ind,
  COUNT(movement_id) ids_totales,
  SUM(movement_oficial_usd_amt) usd_amt,
  AVG(movement_oficial_usd_amt) usd_avg,
  MAX(movement_oficial_usd_amt) max_usd_amt,
  MIN(movement_oficial_usd_amt) min_usd_amt
FROM
  `buda-data-science.master_entities.movements`
WHERE
  movement_state_ind = 'confirmed'
  AND movement_action_ind IN ('deposit','withdrawal')
  AND movement_confirmation_dt BETWEEN "2023-02-01 00:00:00" AND "2023-02-28 23:59:59"
GROUP BY
  patabit_account_id,
  movement_action_ind
ORDER BY
  1 DESC,
  2 DESC
```

SQL SINTAXIS ATM

```
SELECT
  field_1,
  field_2,
  agg_function(field_3),
  agg_function(field_4)
FROM
  {dataset_name.table_name}
WHERE
  {logical_statement}
GROUP BY
  field_1,
  field_2
ORDER BY
  {fields_list}
```

Semana 4

https://docs.google.com/presentation/d/1Am6Vn9XCQPx3PJ-4X3YikdbgjHPgDv-jnPlOUPE38Yg/edit#slide=id.p

https://drive.google.com/file/d/1gjlhorNaEvbqNCfR-bJvL-BSAC6vrZrG/view?usp=share_link

Qué es un Alias? ✏️

- Nombre alternativo que se le da a una columna o tabla en una consulta
- Permite una mayor legibilidad y comprensión de la consulta
- Se define un alias utilizando `AS` seguida del nombre alternativo que se le quiera dar

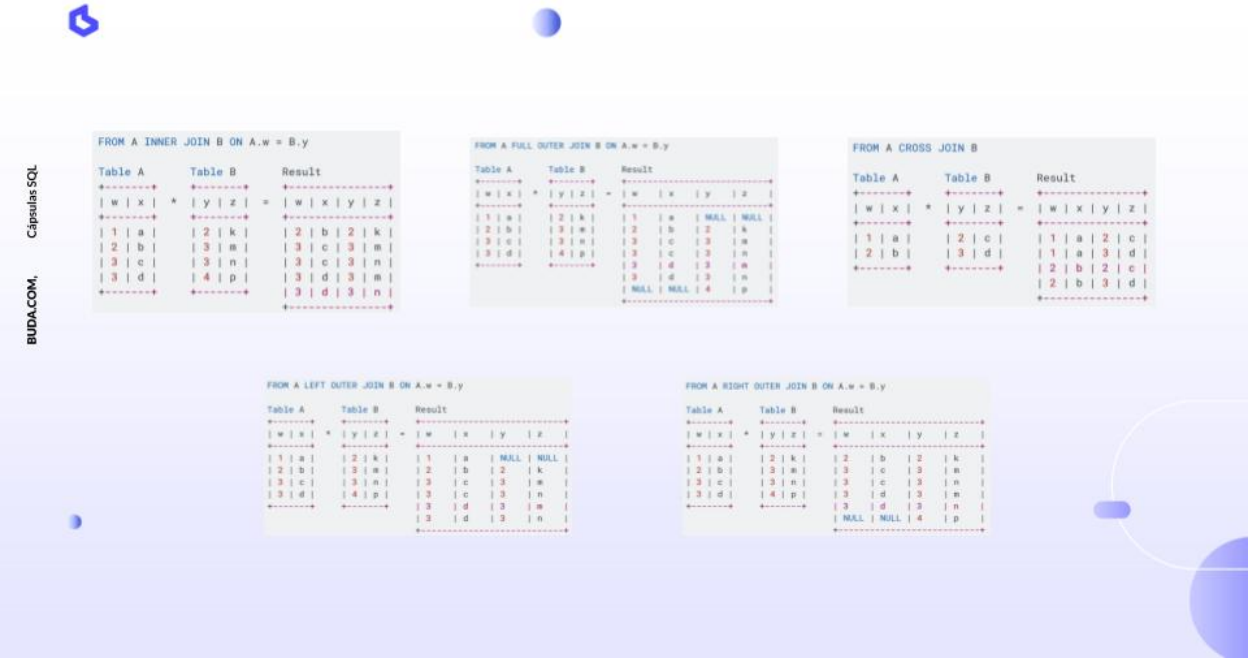
Having 🤲

- Cláusula para filtrar, luego del `GROUP BY`, filas basados en una condición sobre un campo agregado
- La diferencia con `WHERE` es que `HAVING` filtra posterior a la agrupación, no así `WHERE`
- Se utiliza **solo** con funciones de agregación o los alias correspondientes

Combinación de fuentes de datos 🖇️

- Para agregar más de una fuente de datos
- Dos tipos de combinación de fuentes de datos, `JOIN` y `UNION`

JOIN 🔀



- `INNER JOIN`

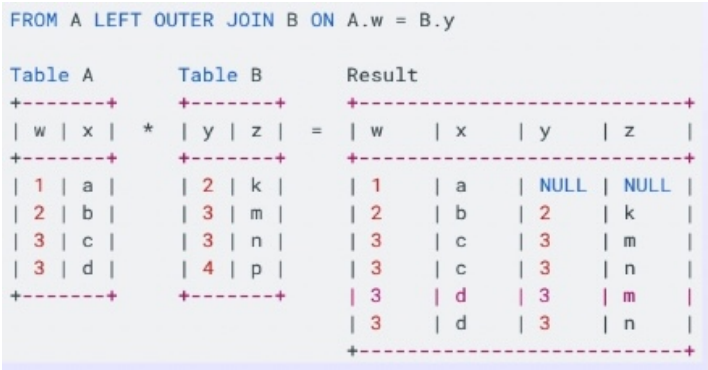
```
FROM A INNER JOIN B ON A.w = B.y

Table A       Table B       Result
+-------+     +-------+     +---------------+
| w | x |  *  | y | z |  =  | w | x | y | z |
+-------+     +-------+     +---------------+
| 1 | a |     | 2 | k |     | 2 | b | 2 | k |
| 2 | b |     | 3 | m |     | 3 | c | 3 | m |
| 3 | c |     | 3 | n |     | 3 | c | 3 | n |
| 3 | d |     | 4 | p |     | 3 | d | 3 | m |
+-------+     +-------+     | 3 | d | 3 | n |
                            +---------------+
```

  - Retorna los registros de las tablas que cumplen con la condición en ambas tablas
  - ▼ SQL SINTAXIS

```
SELECT
  {fields_list}
FROM
  {dataset_name_1.table_name_1}
INNER JOIN
```

```
  {dataset_name_2.table_name_2}
  ON table_name_1.join_field_table_1 = table_name_2.join_field_table_2
```

- `LEFT JOIN`

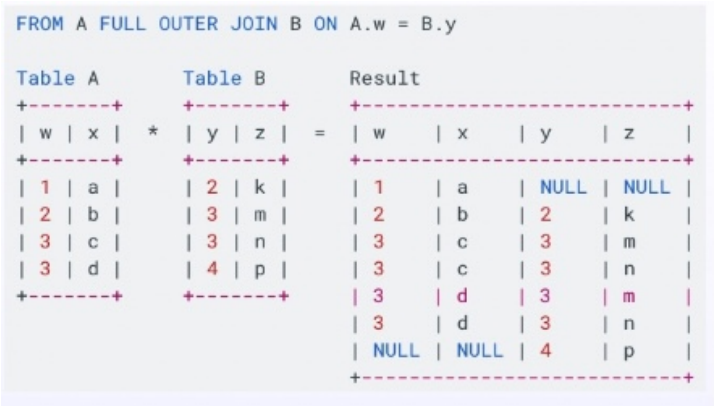


  - Retorna todos los registros de la tabla a la izquierda y, si cruzan los datos de acuerdo a la condición de cruce, agrega las filas de la tabla a la derecha
  - Si no se cumple la condición en la tabla de la derecha devuelve `NULL` en los campos de esa misma tabla

  ▼ SQL SINTAXIS

```
SELECT
  {fields_list}
FROM
  {dataset_name_1.table_name_1}
LEFT JOIN
  {dataset_name_2.table_name_2}
  ON table_name_1.join_field_table_1 = table_name_2.join_field_table_2
```

- `FULL OUTER JOIN`

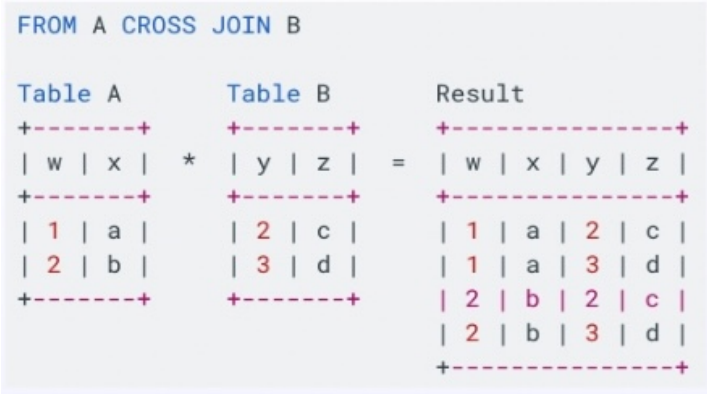


- Retorna todos los registros ambas tablas y retorna NULL en los campos donde no se cumple la condición de cruce

▼ SQL SINTAXIS

```
SELECT
  {fields_list}
FROM
  {dataset_name_1.table_name_1}
FULL OUTER JOIN
  {dataset_name_2.table_name_2}
  ON table_name_1.join_field_table_1 = table_name_2.join_field_table_2
```

- `CROSS JOIN`



- Realiza el producto cruz de los registros de ambas tablas
- No requiere condición de cruce

▼ SQL SINTAXIS

```
SELECT
  {fields_list}
FROM
  {dataset_name_1.table_name_1}
CROSS JOIN
  {dataset_name_2.table_name_2}
```

UNION 🤝

- Se utiliza para combinar resultados de dos o más consultas
- El resultado es una sola tabla con ambos resultados insertados hacia abajo de forma secuencial

- **Deben tener el mismo schema o estructura de columnas**

▼ SQL SINTAXIS

```
SELECT
  field_1,
  field_2,
  field_3
FROM
  {dataset_name_1.table_name_1}
UNION ALL
SELECT
  field_1,
  field_2,
  field_3
FROM
  {dataset_name_2.table_name_2}
```

▼ Ejemplos

```
WITH
  table_1 AS (
  SELECT
    records.numbers,
    records.letters
  FROM (
    SELECT
      STRUCT(1 AS numbers,
        'a' AS letters) AS records
    UNION ALL
    SELECT
      STRUCT(2 AS numbers,
        'b' AS letters) AS records
    UNION ALL
    SELECT
      STRUCT(3 AS numbers,
        'c' AS letters) AS records
    UNION ALL
    SELECT
      STRUCT(3 AS numbers,
        'd' AS letters) AS records ) ),
  table_2 AS (
  SELECT
    records.numbers,
    records.letters
  FROM (
    SELECT
      STRUCT(2 AS numbers,
        'k' AS letters) AS records
    UNION ALL
    SELECT
      STRUCT(3 AS numbers,
        'm' AS letters) AS records
    UNION ALL
    SELECT
      STRUCT(3 AS numbers,
        'n' AS letters) AS records
    UNION ALL
```

```
    SELECT
      STRUCT(4 AS numbers,
        'p' AS letters) AS records ) )


SELECT
  *
FROM
  table_1 AS tabla1
INNER #[LEFT | FULL OUTER | CROSS]
JOIN
  table_2 AS tabla2
  ON tabla1.numbers = tabla2.numbers
```

```
SELECT
  patabit_account_id AS id_patabit,
  SUM(trade_order_oficial_usd_amt) AS volumen_operado,
  COUNT(trade_order_id) AS orders_qty
FROM
  `buda-data-science.master_entities.trade_orders` AS t
WHERE
  trade_order_oficial_usd_amt >= 1000000
GROUP BY
  patabit_account_id
HAVING
  volumen_operado >= 1000000
  AND orders_qty > 1000
```

```
SELECT
  cuentas.patabit_account_id,
  MAX(transaction_creation_dt) AS max_date
FROM
  `buda-data-science.master_entities.accounts_view` AS cuentas
INNER JOIN
  `buda-data-science.master_entities.trade_orders` AS ordenes
ON
  cuentas.patabit_account_id = ordenes.patabit_account_id
WHERE
  user_profession_txt LIKE "%periodista%"
GROUP BY
  1
HAVING
  DATETIME_DIFF(current_datetime,max_date, year) < 1
ORDER BY
  2 ASC
```

```
SELECT
  DISTINCT cuentas.patabit_account_id,
  user_document_country_code
FROM
  `buda-data-science.master_entities.accounts_view` AS cuentas
INNER JOIN
  `buda-data-science.master_entities.movements` AS movimientos
ON
  cuentas.patabit_account_id = movimientos.patabit_account_id
WHERE
  movement_action_ind = 'deposit'
```

```
  AND movement_state_ind = 'confirmed'
  AND movement_currency_code = 'clp'
  AND user_document_country_code != 'cl'
```

SQL SINTAXIS ATM

```
SELECT
  field_1 AS campo_1,
  field_2 AS campo_2,
  agg_function(field_3) AS campo_agregado_1,
  agg_function(field_4) AS campo_agregado_2
FROM
  {dataset_name_1.table_name_1} AS tabla_1
[INNER | LEFT | FULL] JOIN
  {dataset_name_2.table_name_2} AS tabla_2
  ON {join_condition_logical_statement}
WHERE
  {logical_statement}
GROUP BY
  field_1,
  field_2
HAVING
  {logical_statements_with_agg_functions}
ORDER BY
    {fields_list}
```

▼ Ejercicio práctico 1

```
# Pregunta 1

SELECT
  COUNT(patabit_account_id) AS accounts_qty
FROM
  `buda-data-science.master_entities.accounts_view`
WHERE
  account_creation_dt BETWEEN "2023-01-01 00:00:00"
  AND "2023-03-31 23:59:59"
  AND account_category_ind = 'personal'
  AND account_verification_level_flag >= 1
```

```
# Pregunta 2

SELECT
  patabit_account_id,
  COUNT(trade_order_id) AS orders_qty
FROM
  `buda-data-science.master_entities.trade_orders`
WHERE
  transaction_creation_dt BETWEEN "2022-01-01 00:00:00"
  AND "2022-12-31 23:59:59"
```

```
  AND trade_order_action_ind = 'bid'
GROUP BY
  1
ORDER BY
  2 DESC
LIMIT
  1
```

```
# Pregunta 3

SELECT
  COUNT(movement_id) AS deposits_qty
FROM
  `buda-data-science.master_entities.movements`
WHERE
  (movement_confirmation_dt BETWEEN "2022-01-01 00:00:00"
    AND "2022-03-31 23:59:59"
    OR movement_confirmation_dt BETWEEN "2022-10-01 00:00:00"
    AND "2022-12-31 23:59:59")
  AND movement_action_ind = 'withdrawal'
  AND movement_state_ind = 'confirmed'
  AND movement_currency_code = 'btc'
  AND movement_oficial_usd_amt > 10000
```

```
# Desafío

SELECT
  account_category_ind,
  SUM(trade_order_oficial_usd_amt) operated_volume
FROM
  `buda-data-science.master_entities.trade_orders` AS t
INNER JOIN
  `buda-data-science.master_entities.accounts_view` AS a
  ON t.patabit_account_id = a.patabit_account_id
WHERE
  user_document_country_code = 'cl'
  AND account_basic_verification_dt < "2022-01-01 00:00:00"
  AND transaction_creation_dt <= "2023-04-30 23:59:59"
  AND product_source_ind IN ('trading_view','simple_view')
  AND trade_order_action_ind = 'ask'
GROUP BY
  1
```

Etapa 2: Funciones y Operadores

Semana 5

https://docs.google.com/presentation/d/1KF3XOWLBcMxyCnawVFLmlJbaxI7JjsEi2XLuIOnHGU0/edit#slide=id.p

https://drive.google.com/file/d/1xLRxvJ7u_f7aRoBYI9ZTWZxIdOOAGAII/view?usp=share_link

CAST

```
CAST(field_name AS field_type)
```

- Función para convertir el tipo de dato de una columna en otro tipo de dato
- La conversión debe ser compatible
- La precisión puede verse afectada

COALESCE

```
COALESCE([field_name_1 | value_1], [field_name_2 | value_2], ..., [field_name_n | value_n])
```

- Función que entrega el primer valor no `NULL` de una lista de valores o campos
- Seleccionar un valor por defecto
- Seleccionar un valor entre varios campos o valores

CASE WHEN

```
CASE WHEN logical_proposition THEN return_true_value ELSE return_false_value END
```

- Para realizar una operación condicional en una consulta
- Realizar operaciones de selección condicional
- Crear nuevas categorías o columnas calculadas

Semana 6

https://docs.google.com/presentation/d/1EIDAY3uLp3tqaneyUPe5fFQ1ojhc0CWbxZKasQbxujA/edit#slide=id.p

https://drive.google.com/file/d/1jSjuA4Rl2pyIetg9L0M11CbHl_pBTvSb/view?usp=share_link

DATETIME STRUCTURE

```
YYYY-[M]M-[D]D { |T|t} [H]H:[M]M:[S]S[.F]
```

- YYYY : Año de cuatro dígitos
- [M]M : Mes de uno o dos dígitos
- [D]D : Día de uno o dos dígitos
- { | T | t} : Un separador como espacio o una T o una t. Esto es un indicador de tiempo
- [H]H : Hora de uno o dos dígitos
- [M]M : Minutos de uno o dos dígitos
- [S]S : Segundos de uno o dos dígitos
- [.F] : Parte fraccional de hasta 6 dígitos (Precisión de microsegundos)

CURRENT_DATETIME

```
CURRENT_DATETIME([TimeZone])
```

- Devuelve el valor datetime actual, al momento de ejecutar la consulta, para la zona horaria definida

DATE_TRUNC

```
[DATE_TRUNC | DATETIME_TRUNC](field_name, datetime_part)
```

- Para obtener una fecha (DATE o DATETIME, dependiendo de cual de ambas funciones se utilice) truncada a la parte de la fecha que se especifique

EXTRACT

```
EXTRACT(datetime_part FROM field_name)
```

- Para obtener el valor numérico de la parte del registro de fecha que se especifica

DATE_ADD / DATE_SUB

```
[DATE_ADD | DATETIME_ADD](field_name, INTERVAL interval_qty datetime_part)
```

- Para añadir o quitar la cantidad especificada de la parte definida sobre la fecha del registro

DATE_DIFF

```
[DATE_DIFF | DATETIME_DIFF](field_name_1, field_name_2, datetime_part)
```

- Para conocer la diferencia entre dos campos de fechas según la parte de fecha que se indique

Semana 7

https://docs.google.com/presentation/d/1N1YuaHLIX-cqjWij92l_kMpmDQYd_CQqDnjkuaX7cSY/edit#slide=id.p

https://drive.google.com/file/d/1ihH01uRTidmAZ5BFL0TsBD5ZYyZ-qX7Q/view?usp=sharing

Consultas Anidadas

```
SELECT
  *
FROM (
  SELECT
    *
  FROM
    table_name
)
```

- Consultas que se encuentran dentro de otra principal
- Se usa para operaciones más complejas y obtener resultados basados en la sub consulta
- La sub consulta, o consulta anidada, se ejecuta primero y el resultado se usa como entrada para la siguiente consulta

CTEs

```
WITH cte_name AS (
  SELECT
    *
  FROM
    table_name
)

SELECT
  *
FROM
  cte_name
```

- Se definen y nombran fuera de la consulta principal
- Se definen con un WITH al principio seguido por el nombre de la CTE. Una siguiente CTE se remplaza el WITH por una coma y se repite el proceso
- Se utilizan para mayor legibilidad de la consulta, dividiendo una consulta compleja en partes más manejables y, sobre todo, reutilizables
- Se pueden definir las cantidades de CTEs que uno estime conveniente

Semana 8

Particiones

```
PARTITION BY [fields_list]
```

- Sirve para dividir grandes conjuntos de datos en secciones más pequeñas
- Facilita la gestión y mejora el rendimiento de nuestras consultas
- Se generan las agrupaciones de acuerdo a uno o más campos de una tabla

Funciones de Numeración

```
[numbering_function] OVER (PARTITION BY [fields_list] ORDER BY [order_criteria])
```

- `ROW_NUMBER()`
  - Número secuencial para cada registro en una partición y en el orden especificado
- `RANK()`
  - Rango de filas con valores iguales se les asigna el mismo rango y al cambiar de rango se omiten los rangos siguientes
- `DENSE_RANK()`
  - Rango de filas con valores iguales se les asigna el mismo rango y al cambiar de rango no se omiten los rangos siguientes

Funciones de Navegación

```
[navigation_function] OVER (PARTITION BY [fields_list] ORDER BY [order_criteria])
```

- `LAG()`
  - Entrega el anterior valor en una partición de acuerdo al orden especificado
- `LEAD()`
  - Entrega el siguiente valor en una partición de acuerdo al orden especificado
- `FIRST_VALUE()`
  - Entrega el primer valor en una partición de acuerdo al orden especificado
- `LAST_VALUE()`
  - Entrega el último valor en una partición de acuerdo al orden especificado

▼ Ejemplos

```
WITH
  numbers_different_partition AS (
  SELECT
    1 AS partition_id,
    number
  FROM
    UNNEST([1,2,3,4,4,4,5,5,6,7,8,9]) AS number
  UNION ALL
  SELECT
    2 AS partition_id,
    number
  FROM
    UNNEST([11,21,31,41,41,41,51,51,61,71,81,91]) AS number
```

```
)

SELECT
  partition_id,
  number,
  ROW_NUMBER() OVER (PARTITION BY partition_id ORDER BY number) AS row_number_function,
  RANK() OVER (PARTITION BY partition_id ORDER BY number) AS rank_function,
  DENSE_RANK() OVER (PARTITION BY partition_id ORDER BY number) AS dense_rank_function,
  LAG(number) OVER (PARTITION BY partition_id ORDER BY number) AS lag_function,
  LEAD(number) OVER (PARTITION BY partition_id ORDER BY number) AS lead_function,
  FIRST_VALUE(number) OVER (PARTITION BY partition_id ORDER BY number) AS first_value_function,
  LAST_VALUE(number) OVER (PARTITION BY partition_id ORDER BY number) AS last_value_function,
FROM
  numbers_different_partition
ORDER BY
  number
```