🦆 Sesi 2

DuckDB untuk Query Analitik

HARI 1: Fundamental Analisis Data

Durasi: 2.5 jam (13:00 - 15:30)
Notebook: `01_duckdb_intro.ipynb`

🎯 Tujuan Sesi

Setelah sesi ini, Anda bisa:

- ✅ Paham apa itu DuckDB dan use cases-nya
- ✅ Menulis SQL queries untuk analisis data
- ✅ Menggunakan aggregate functions dan GROUP BY
- ✅ Menguasai window functions (ROW_NUMBER, RANK, LAG, LEAD)
- ✅ Menulis CTEs (Common Table Expressions)
- ✅ Mengintegrasikan DuckDB dengan Pandas alur kerja
- ✅ Membandingkan performa DuckDB vs Pandas

📋 Agenda Sesi

Waktu	Topik	Durasi
13:00 - 13:15	Kick-off DuckDB & Setup Environment	15 min
13:15 - 13:45	DuckDB Fundamentals & Storage Concepts	30 min
13:45 - 14:15	SELECT, FILTER, ORDER BY	30 min
14:15 - 14:45	Aggregations & Window Functions	30 min
14:45 - 15:00	Integrasi DuckDB + Pandas & Wrap-up	15 min

🦆 Apa itu DuckDB?

Penjelasan Sederhana: DuckDB itu seperti Excel yang bisa dipakai dengan SQL, tapi lebih cepat untuk analisis data besar

Database untuk Analisis Data (Tidak Perlu Install Server)

Mudahnya: DuckDB bisa langsung dipakai tanpa setup ribet seperti database lain

Fitur Utama

- 📊 **OLAP** (Analytical) Database
- ⚡ **In-Memory** Processing
- 📂 **Columnar** Storage
- 🚀 **Fast** Query Execution
- 🔧 **Embedded** (No Server)
- 🐍 **Python** Native Support

"SQLite for Analytics" 🎯

Kapan Dipakai

- ✅ Analisis Data
- ✅ Pipeline ETL
- ✅ Alur Kerja Data Science
- ✅ Fast Prototyping
- ✅ Analitik Lokal
- ✅ Testing & Development

🤔 Why DuckDB?

Dibandingkan dengan Tools Lain

Feature	DuckDB	Pandas	PostgreSQL	SQLite
Setup	✅ Mudah	✅ Mudah	❌ Rumit	✅ Mudah
Speed	✅ Cepat	⚠️ Sedang	✅ Cepat	⚠️ Lambat (Analitik)
SQL Support	✅ Lengkap	❌ No	✅ Lengkap	⚠️ Terbatas
Analytics	✅ Optimal	✅ Baik	⚠️ Campuran	❌ Buruk
Memory	✅ Efisien	⚠️ Tinggi	✅ Efisien	✅ Rendah
Big Data	✅ Baik	❌ Terbatas	✅ Baik	❌ Buruk

VS OLAP vs OLTP

Penjelasan: OLAP (Online Analytical Processing) untuk analisis data (baca banyak), OLTP (Online Transaction Processing) untuk transaksi (tulis banyak) DuckDB cocok untuk OLAP (analisis)

OLTP (Transaction)

Contoh: PostgreSQL, MySQL

- ✅ Many small transactions
- ✅ Row-oriented
- ✅ INSERT/UPDATE heavy
- ✅ Normalized data
- ✅ Concurrent users
- ❌ Slow for analytics

OLAP (Analytical)

Contoh: DuckDB, ClickHouse

- ✅ Large scans
- ✅ Column-oriented
- ✅ SELECT heavy
- ✅ Denormalized data
- ✅ Batch processing
- ✅ **Cepat for analytics**

🛠️ Installation & Setup

```
# Install DuckDB
!pip install duckdb

# Import libraries
import duckdb
import pandas as pd

# Check version
print(f"DuckDB version: {duckdb.__version__}")
```

Versi terbaru: 1.4.1 LTS (7 Oktober 2025)

```
# Create connection
# In-memory (data hilang setelah session)
conn = duckdb.connect(':memory:')

# Persistent (save ke file)
conn = duckdb.connect('analytics.duckdb')

# Check connection
print(conn)
```

📁 Loading Data ke DuckDB

Berbagai Cara Load Data

1. Direct dari Parquet

```
# Query langsung tanpa load
result = duckdb.query("""
    SELECT * FROM
    'RUP-2025.parquet'
    LIMIT 10
""").df()
```

2. Register Pandas DataFrame

```
# Load ke Pandas dulu
df = pd.read_parquet('RUP.parquet')

# Register ke DuckDB
conn.register('rup', df)

# Query
result = conn.query("SELECT * FROM rup")
```

📁 Loading Data ke DuckDB (lanjutan)

3. CREATE TABLE

```
# Create table dari file
conn.execute("""
    CREATE TABLE rup AS
    SELECT * FROM
    'RUP-2025.parquet'
""")
```

4. Load dari CSV

```
conn.execute("""
    CREATE TABLE data AS
    SELECT * FROM
    read_csv_auto('data.csv')
""")
```

🔍 Basic SQL Queries

SELECT, WHERE, LIMIT

```sql
-- Select all columns
SELECT * FROM rup LIMIT 10;

-- Select specific columns
SELECT nama_paket, pagu, metode_pengadaan
FROM rup
LIMIT 10;

-- WHERE condition
SELECT nama_paket, pagu
FROM rup
WHERE pagu > 1000000000
LIMIT 10;
```

🔍 Basic SQL Queries (lanjutan)

```sql
-- Multiple conditions
SELECT *
FROM rup
WHERE pagu > 1000000000
  AND metode_pengadaan = 'Tender'
LIMIT 100;
```

📊 Gabung Functions

COUNT, SUM, AVG, MIN, MAX

```sql
-- Count total records
SELECT COUNT(*) as total_paket
FROM rup;

-- Sum total pagu
SELECT SUM(pagu) as total_pagu
FROM rup;

-- Average, Min, Max
SELECT
    COUNT(*) as jumlah_paket,
    SUM(pagu) as total_pagu,
    AVG(pagu) as rata_rata_pagu,
    MIN(pagu) as pagu_min,
    MAX(pagu) as pagu_max
FROM rup;
```

🎯 GROUP BY Operations

```sql
-- Group by metode pengadaan
SELECT
    metode_pengadaan,
    COUNT(*) as jumlah_paket,
    SUM(pagu) as total_pagu,
    AVG(pagu) as rata_rata_pagu
FROM rup
GROUP BY metode_pengadaan
ORDER BY total_pagu DESC;

-- Group by multiple columns
SELECT
    metode_pengadaan,
    jenis_pengadaan,
    COUNT(*) as jumlah,
    SUM(pagu) as total
FROM rup
GROUP BY metode_pengadaan, jenis_pengadaan
ORDER BY total DESC;
```

🔢 HAVING Clause

Filter After Aggregation

```sql
-- Filter hasil GROUP BY
SELECT
    metode_pengadaan,
    COUNT(*) as jumlah_paket,
    SUM(pagu) as total_pagu
FROM rup
GROUP BY metode_pengadaan
HAVING COUNT(*) > 100  -- Filter after penggabungan
ORDER BY total_pagu DESC;
```

🔢 HAVING Clause (lanjutan)

```sql
-- Multiple HAVING conditions
SELECT
    nama_satker,
    COUNT(*) as jumlah_paket,
    SUM(pagu) as total_pagu
FROM rup
GROUP BY nama_satker
HAVING COUNT(*) > 10
   AND SUM(pagu) > 1000000000
ORDER BY total_pagu DESC
LIMIT 20;
```

🎨 DISTINCT & ORDER BY

```sql
-- Unique values
SELECT DISTINCT metode_pengadaan
FROM rup;

-- Count distinct
SELECT COUNT(DISTINCT metode_pengadaan) as jumlah_metode
FROM rup;

-- Order by multiple columns
SELECT nama_paket, pagu, metode_pengadaan
FROM rup
ORDER BY pagu DESC, nama_paket ASC
LIMIT 20;

-- Order with NULL handling
SELECT *
FROM rup
ORDER BY pagu DESC NULLS LAST;
```

💫 Window Functions: Konsep

Apa itu Window Functions?

Window functions melakukan kalkulasi pada sekelompok baris yang berhubungan dengan baris terkini, **tanpa collapse hasil seperti GROUP BY**.

```
-- GROUP BY: Collapse ke 1 row per group
SELECT metode_pengadaan, COUNT(*)
FROM rup
GROUP BY metode_pengadaan;

-- WINDOW: Tetap semua rows, tambah kolom baru
SELECT
    nama_paket,
    pagu,
    metode_pengadaan,
    ROW_NUMBER() OVER (PARTITION BY metode_pengadaan
                       ORDER BY pagu DESC) as rank_in_metode
FROM rup;
```

🏆 ROW_NUMBER, RANK, DENSE_RANK

```sql
SELECT
    nama_paket,
    pagu,
    metode_pengadaan,
    ROW_NUMBER() OVER (ORDER BY pagu DESC) as row_num,
    RANK() OVER (ORDER BY pagu DESC) as rank,
    DENSE_RANK() OVER (ORDER BY pagu DESC) as dense_rank
FROM rup
LIMIT 20;
```

Perbedaan:

- **ROW_NUMBER:** 1, 2, 3, 4, 5 (always unique)
- **RANK:** 1, 2, 2, 4, 5 (skip number jika tie)
- **DENSE_RANK:** 1, 2, 2, 3, 4 (no skip)

PARTITION BY

Window Function per Group

```sql
-- Top 5 paket per metode pengadaan
WITH ranked AS (
    SELECT
        nama_paket,
        pagu,
        metode_pengadaan,
        ROW_NUMBER() OVER (
            PARTITION BY metode_pengadaan
            ORDER BY pagu DESC
        ) as rank_in_metode
    FROM rup
)
SELECT *
FROM ranked
WHERE rank_in_metode <= 5
ORDER BY metode_pengadaan, rank_in_metode;
```

PARTITION BY = **GROUP BY** untuk window functions

📈 LAG & LEAD Functions

Akses Sebelumnya/Selanjutnya Row

```sql
-- Time series analysis dengan LAG/LEAD
SELECT
    tgl_pengumuman_paket::DATE as tanggal,
    COUNT(*) as jumlah_paket,
    LAG(COUNT(*), 1) OVER (ORDER BY tgl_pengumuman_paket::DATE) as prev_day,
    LEAD(COUNT(*), 1) OVER (ORDER BY tgl_pengumuman_paket::DATE) as next_day,
    COUNT(*) - LAG(COUNT(*), 1) OVER (ORDER BY tgl_pengumuman_paket::DATE) as delta
FROM rup
WHERE tgl_pengumuman_paket IS NOT NULL
GROUP BY tgl_pengumuman_paket::DATE
ORDER BY tanggal;
```

LAG: Sebelumnya row | **LEAD:** Selanjutnya row

Running Totals & Moving Averages

```sql
-- Running total (cumulative sum)
SELECT
    tgl_pengumuman_paket::DATE as tanggal,
    COUNT(*) as daily_count,
    SUM(COUNT(*)) OVER (ORDER BY tgl_pengumuman_paket::DATE) as running_total,
    AVG(COUNT(*)) OVER (
        ORDER BY tgl_pengumuman_paket::DATE
        ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
    ) as moving_avg_7day
FROM rup
WHERE tgl_pengumuman_paket IS NOT NULL
GROUP BY tanggal
ORDER BY tanggal;
```

ROWS BETWEEN: Define window frame

🔗 CTEs (Common Table Expressions)

WITH Clause untuk Query Kompleks

```
-- CTE untuk query yang lebih readable
WITH satker_summary AS (
    SELECT
        nama_satker,
        COUNT(*) as jumlah_paket,
        SUM(pagu) as total_pagu,
        AVG(pagu) as avg_pagu
    FROM rup
    GROUP BY nama_satker
),
top_satker AS (
    SELECT *
    FROM satker_summary
    WHERE jumlah_paket > 10
    ORDER BY total_pagu DESC
    LIMIT 20
)
```

🔗 CTEs (Common Table Expressions) (lanjutan)

```sql
SELECT *
FROM top_satker
WHERE avg_pagu > 1000000;
```

CTEs = Temporary named result sets

🎯 Multiple CTEs

```
-- Beberapa CTEs untuk analisis rumit
WITH metode_stats AS (
    SELECT
        metode_pengadaan,
        COUNT(*) as jumlah,
        SUM(pagu) as total_pagu
    FROM rup
    GROUP BY metode_pengadaan
),
jenis_stats AS (
    SELECT
        jenis_pengadaan,
        COUNT(*) as jumlah,
        SUM(pagu) as total_pagu
    FROM rup
    GROUP BY jenis_pengadaan
),
```

🎯 Multiple CTEs (lanjutan)

```
combined AS (
    SELECT 'Metode' as kategori, * FROM metode_stats
    UNION ALL
    SELECT 'Jenis' as kategori, * FROM jenis_stats
)
SELECT * FROM combined
ORDER BY total_pagu DESC;
```

🔄 Subqueries

```sql
-- Subquery di WHERE
SELECT nama_paket, pagu
FROM rup
WHERE pagu > (SELECT AVG(pagu) FROM rup);

-- Subquery di FROM (inline view)
SELECT
    metode,
    avg_pagu,
    RANK() OVER (ORDER BY avg_pagu DESC) as rank
FROM (
    SELECT
        metode_pengadaan as metode,
        AVG(pagu) as avg_pagu
    FROM rup
    GROUP BY metode_pengadaan
) subquery;
```

CTEs > Subqueries agar mudah dibaca! 👍

Date/Time Functions

```sql
-- Extract date parts
SELECT
    tgl_pengumuman_paket,
    YEAR(tgl_pengumuman_paket) as tahun,
    MONTH(tgl_pengumuman_paket) as bulan,
    QUARTER(tgl_pengumuman_paket) as kuartal,
    DAYOFWEEK(tgl_pengumuman_paket) as hari_dalam_minggu,
    DAYNAME(tgl_pengumuman_paket) as nama_hari
FROM rup
WHERE tgl_pengumuman_paket IS NOT NULL
LIMIT 10;

-- Date arithmetic
SELECT
    tgl_pengumuman_paket,
    tgl_pengumuman_paket + INTERVAL 30 DAY as plus_30_hari,
    DATE_DIFF('day', tgl_awal_kontrak, tgl_akhir_kontrak) as durasi_hari
FROM rup
LIMIT 10;
```

📝 String Functions

```sql
-- String pengolahan
SELECT
    nama_paket,
    UPPER(nama_paket) as upper_case,
    LOWER(nama_paket) as lower_case,
    LENGTH(nama_paket) as panjang,
    SUBSTRING(nama_paket, 1, 50) as first_50_chars,
    REPLACE(nama_paket, 'Pengadaan', 'Procurement') as replaced
FROM rup
LIMIT 10;

-- String search
SELECT *
FROM rup
WHERE nama_paket LIKE '%Konstruksi%'
   OR nama_paket ILIKE '%konstruksi%'  -- tidak case sensitive
LIMIT 20;

-- Ekspresi reguler
SELECT * FROM rup
WHERE REGEXP_MATCHES(nama_paket, '(?i)jalan|jembatan')
LIMIT 20;
```

🔢 CASE WHEN (Conditional Logic)

```sql
-- Categorize pagu
SELECT
    nama_paket,
    pagu,
    CASE
        WHEN pagu < 100000000 THEN 'Small'
        WHEN pagu < 1000000000 THEN 'Sedang'
        WHEN pagu < 10000000000 THEN 'Large'
        ELSE 'Very Large'
    END as kategori_pagu,
    metode_pengadaan
FROM rup
LIMIT 20;
```

🔢 CASE WHEN (Conditional Logic) (lanjutan)

```sql
-- Use in penggabungan
SELECT
    CASE
        WHEN pagu < 100000000 THEN 'Small'
        WHEN pagu < 1000000000 THEN 'Sedang'
        ELSE 'Large'
    END as kategori,
    COUNT(*) as jumlah
FROM rup
GROUP BY kategori;
```

Penggabungan dengan Pandas

Execute Query → DataFrame

```
import duckdb
import pandas as pd

# Load data
df = pd.read_parquet('RUP-2025.parquet')

# Register DataFrame
conn = duckdb.connect(':memory:')
conn.register('rup', df)
```

Penggabungan dengan Pandas (lanjutan)

```
# Query dan dapat hasil sebagai DataFrame
result_df = conn.execute("""
    SELECT
        metode_pengadaan,
        COUNT(*) as jumlah,
        SUM(pagu) as total_pagu
    FROM rup
    GROUP BY metode_pengadaan
    ORDER BY total_pagu DESC
""").df()

print(result_df)
```

🔄 Query Pandas DataFrame Langsung

```
# Cara 1: Register dulu
conn.register('rup_df', df)
result = conn.execute("SELECT * FROM rup_df WHERE pagu > 1000000").df()

# Cara 2: Query langsung (auto register)
result = duckdb.query("""
    SELECT
        metode_pengadaan,
        AVG(pagu) as avg_pagu
    FROM df
    GROUP BY metode_pengadaan
""").df()

# Cara 3: Method relation
result = conn.from_df(df).filter("pagu > 1000000").df()
```

DuckDB bisa query Pandas DataFrame secara native! 🎉

⚡ Kecepatan Comparison

```
import time

# Pandas approach
start = time.time()
pandas_result = df.groupby('metode_pengadaan')['pagu'].agg(['count', 'sum', 'mean'])
pandas_time = time.time() - start
```

⚡ Kecepatan Comparison (lanjutan)

```
# DuckDB approach
start = time.time()
duckdb_result = conn.execute("""
    SELECT
        metode_pengadaan,
        COUNT(*) as count,
        SUM(pagu) as sum,
        AVG(pagu) as mean
    FROM df
    GROUP BY metode_pengadaan
""").df()
duckdb_time = time.time() - start

print(f"Pandas: {pandas_time:.4f}s")
print(f"DuckDB: {duckdb_time:.4f}s")
print(f"Speedup: {pandas_time/duckdb_time:.2f}x")
```

DuckDB lebih cepat! ⚡

💾 Export Results

```
# Export ke various formats
conn.execute("""
    COPY (
        SELECT * FROM rup WHERE pagu > 1000000000
    ) TO 'high_value_packages.csv' (HEADER, DELIMITER ',');
""")

# Export ke Parquet
conn.execute("""
    COPY (
        SELECT * FROM rup
    ) TO 'rup_export.parquet' (FORMAT PARQUET);
""")

# Export ke Excel via Pandas
result_df = conn.execute("SELECT * FROM rup").df()
result_df.to_excel('rup_report.xlsx', index=False)
```

🎯 Contoh Dunia Nyata

Example 1: Top Satker Analysis

```sql
WITH satker_metrics AS (
    SELECT
        nama_satker,
        COUNT(*) as total_paket,
        SUM(pagu) as total_pagu,
        AVG(pagu) as avg_pagu,
        MIN(pagu) as min_pagu,
        MAX(pagu) as max_pagu
    FROM rup
    GROUP BY nama_satker
),
ranked_satker AS (
    SELECT
        *,
        ROW_NUMBER() OVER (ORDER BY total_pagu DESC) as rank_by_pagu,
        ROW_NUMBER() OVER (ORDER BY total_paket DESC) as rank_by_count
    FROM satker_metrics
)
```

Example 1: Top Satker Analysis (lanjutan)

```sql
SELECT * FROM ranked_satker
WHERE rank_by_pagu <= 20 OR rank_by_count <= 20
ORDER BY total_pagu DESC;
```

🎯 Contoh Dunia Nyata

Example 2: Monthly Trend Analysis

```sql
WITH monthly_stats AS (
    SELECT
        DATE_TRUNC('month', tgl_pengumuman_paket) as bulan,
        COUNT(*) as jumlah_paket,
        SUM(pagu) as total_pagu
    FROM rup
    WHERE tgl_pengumuman_paket IS NOT NULL
    GROUP BY bulan
)
SELECT
    bulan,
    jumlah_paket,
    total_pagu,
    LAG(jumlah_paket) OVER (ORDER BY bulan) as prev_month_count,
    jumlah_paket - LAG(jumlah_paket) OVER (ORDER BY bulan) as delta_count,
    (jumlah_paket - LAG(jumlah_paket) OVER (ORDER BY bulan))::FLOAT /
        NULLIF(LAG(jumlah_paket) OVER (ORDER BY bulan), 0) * 100 as pct_change
FROM monthly_stats
ORDER BY bulan;
```

🎯 Contoh Dunia Nyata

Example 3: Pivot Analysis

```sql
-- Pivot: Metode vs Jenis
SELECT
    metode_pengadaan,
    SUM(CASE WHEN jenis_pengadaan = 'Barang' THEN 1 ELSE 0 END) as barang,
    SUM(CASE WHEN jenis_pengadaan = 'Jasa Konsultansi' THEN 1 ELSE 0 END) as konsultansi,
    SUM(CASE WHEN jenis_pengadaan = 'Pekerjaan Konstruksi' THEN 1 ELSE 0 END) as konstruksi,
    COUNT(*) as total
FROM rup
GROUP BY metode_pengadaan
ORDER BY total DESC;

-- Alternative: PIVOT (jika supported)
PIVOT rup
ON jenis_pengadaan
USING COUNT(*)
GROUP BY metode_pengadaan;
```

💡 Praktik Terbaik

Tips Menulis SQL yang Baik

1. **Use CTEs untuk query yang rumit** 📝
    - Lebih readable daripada nested subqueries
2. **Index pada kolom yang sering difilter** ⚡

```
CREATE INDEX idx_pagu ON rup(pagu);
```

3. **EXPLAIN untuk analyze query plan** 🔍

```
EXPLAIN SELECT * FROM rup WHERE pagu > 1000000;
```

💡 Praktik Terbaik (lanjutan)

4. **Limit results saat development** 🎯
    - Gunakan LIMIT untuk testing
5. **Gunakan meaningful aliases** 🏷️

```
SELECT
    COUNT(*) as total_paket,   -- Baik
    SUM(pagu) as tp              -- Bad
FROM rup;
```

6. **Format SQL agar mudah dibaca** ✨
    - Indentation, line breaks, uppercase keywords

💡 Praktik Terbaik (lanjutan)

7. **Avoid SELECT *** ⚠️
    - Tentukan kolom yang dibutuhkan saja
8. **Beri komentar pada query rumit** 💬

```
-- Calculate running total per satker
SELECT ... ;
```

🎯 Latihan Praktis

Latihan untuk Anda

1. **Basic Aggregation**
    - Top 10 satker by total pagu
    - Count paket per metode pengadaan
    - Average pagu per jenis pengadaan
2. **Window Functions**
    - Rank paket by pagu
    - Top 3 paket per satker
    - Running total per month

🎯 Latihan Praktis (lanjutan)

3. **Time Series**
    - Monthly penggabungan
    - Week-over-week growth
    - Identify trends
4. **Analisis Rumit**
    - Pivot table: Metode vs Jenis
    - CTEs untuk multi-step analysis
    - Combine menyaring dan penggabungans

🎯 Latihan Praktis (lanjutan)

5. **Kecepatan Test**
    - Compare Pandas vs DuckDB
    - Benchmark complex queries
    - Optimize slow queries
6. **Data Quality**
    - Find duplicate records
    - Identify outliers
    - Validate data integrity

📚 Poin Penting

Yang Harus Anda Ingat

- ✅ DuckDB = SQL untuk data analysis yang cepat
- ✅ Perfect untuk analytical workloads (OLAP)
- ✅ Window functions untuk ranking & running calculations
- ✅ CTEs membuat complex queries lebih readable
- ✅ Native integration dengan Pandas
- ✅ Lebih cepat dari Pandas untuk penggabungan
- ✅ Query file Parquet secara langsung
- ✅ No server needed - embedded database

DuckDB + Pandas = Powerful combo! 💪

🔗 Resources

Dokumentasi & Learning

- **DuckDB Docs:** https://duckdb.org/docs/
- **SQL Reference:** https://duckdb.org/docs/sql/introduction
- **DuckDB Python API:** https://duckdb.org/docs/api/python/overview
- **Window Functions Guide:** https://duckdb.org/docs/sql/window_functions

🔗 Resources (lanjutan)

Sesi Selanjutnya

Sesi 3: Visualisasi Data

- Matplotlib & Seaborn untuk static plots
- Plotly untuk interactive tampilans
- Data storytelling prinsip

BREAK sampai 15:45 ☕

🎉 Selesai Sesi 2!

Excellent Work! 👏

Coffee Break sampai 15:45

Pertanyaan?? 🙋

📞 Referensi Cepat

Pola Umum

```sql
-- Aggregation
SELECT col, COUNT(*), SUM(val) FROM table GROUP BY col;

-- Window fungsi (blok kode yang bisa dipanggil)
SELECT col, ROW_NUMBER() OVER (ORDER BY val DESC) FROM table;

-- CTE
WITH cte AS (SELECT ...) SELECT * FROM cte;

-- Time series
SELECT DATE_TRUNC('month', date_col), COUNT(*) GROUP BY 1;

-- Top N per group
SELECT * FROM (
    SELECT *, ROW_NUMBER() OVER (PARTITION BY grp ORDER BY val) rn
) WHERE rn <= 5;
```

Sampai jumpa di Sesi 3! 🚀

📞 Tetap Terhubung

Resources & Support

- 👤 **Nama:** [Kurnia Ramadhan,ST.,M.Eng]
- 📧 **Email:** [kurnia@ramadhan.me]

We're here to support your journey! 🤝