

# Complex - Documentazione Tecnica Approfondita

File di riferimento: `documentazione_tecnica_complex.md`

---

Copyright © 2025 TIM SPA - All Rights Reserved

## Autori:

- Daniele Speziale ([daniele.speziale@guest.telecomitalia.it](mailto:daniele.speziale@guest.telecomitalia.it)) - Lead Developer & System Architect
  - Marco Lucidi ([marco.lucidi@guest.telecomitalia.it](mailto:marco.lucidi@guest.telecomitalia.it)) - Senior Developer & Database Specialist
- 

## Indice

- [1. Panoramica del Sistema](#)
  - [2. Architettura Generale](#)
  - [3. Componenti Principali](#)
  - [4. Database Supportati](#)
  - [5. Configurazione](#)
  - [6. Query Processing](#)
  - [7. Sistema Export Excel/CSV](#)
  - [8. Sistema di Logging](#)
  - [9. Sicurezza](#)
  - [10. Deployment](#)
  - [11. Troubleshooting](#)
  - [12. API Reference](#)
- 

## Panoramica del Sistema

**Complex** è uno strumento avanzato per il trasferimento di dati tra database eterogenei sviluppato da TIM SPA. Il sistema supporta query complesse, template SQL, processing batch e trasferimenti multi-database con logging completo.

## Caratteristiche Principali

- Multi-Database:** Supporto per Oracle e SQL Server
- Query Avanzate:** SQL inline, file esterni, template con parametri
- Export Dati:** Generazione automatica Excel/CSV dai risultati query

- **Processing Sicuro:** Validazione query, timeout, gestione errori
- **Logging Completo:** Log strutturati con timestamp e livelli
- **Configurazione Flessibile:** JSON-based con validazione
- **Batch Processing:** Elaborazione a blocchi per grandi dataset

## Versione

- **Versione Corrente:** Enhanced Multi-Query Processor
  - **Linguaggio:** Python 3.13
  - **Database:** SQLite per storage locale
  - **Framework Web:** Flask con Blueprint
  - **Compatibilità:** Oracle 11g+, SQL Server 2016+
- 

## Architettura Generale

```
Complex/
├── app.py                # Entry point principale
├── config.json           # Configurazione sistema
├── Complex/
│   ├── __init__.py
│   ├── database_manager.py    # Gestione connessioni DB
│   └── enhanced_multi_query_processor.py # Processore query
├── queries/              # File SQL esterni
│   ├── DWH-IA-STD.sql
│   ├── IAM-PM-DB.sql
│   └── ...
├── logs/                 # Directory log
│   └── db_transfer_enhanced_*.log
└── job/                  # Modulo Timesheet (secondario)
    ├── app.py
    ├── models.py
    └── blueprints/
```

## Flusso di Elaborazione

### 1. Inizializzazione

- Caricamento `config.json`
- Validazione configurazione
- Setup logging con timestamp

### 2. Test Connessioni

- Verifica connettività database
- Autenticazione credenziali
- Validazione schemi

### 3. Processing Query

- Risoluzione SQL da diverse fonti
- Esecuzione batch sicura
- Trasferimento dati con schema mapping

### 4. Finalizzazione

- Verifica integrità dati
- Cleanup connessioni
- Report risultati

---

## Componenti Principali

### 1. DatabaseManager (`Complex/database_manager.py`)

Gestisce tutte le connessioni database con supporto per:

#### Funzionalità Oracle

```
python

# Connessione Oracle con Thick Mode
with db_manager.get_oracle_connection('oracle_dwh') as conn:
    cursor = conn.cursor()
    cursor.execute(sql_query)
    results = cursor.fetchall()
```

#### Funzionalità SQL Server

```
python

# Connessione SQL Server con Windows Auth
with db_manager.get_mssql_connection('mssql_dest') as conn:
    cursor = conn.cursor()
    cursor.execute(create_table_sql)
    conn.commit()
```

### Metodi Principali

- `get_oracle_connection(db_name)`: Context manager per Oracle

- `get_mssql_connection(db_name)`: Context manager per SQL Server
- `test_connection(db_name)`: Verifica connettività
- `ensure_schema_exists(db_name, schema)`: Creazione schemi

## 2. EnhancedMultiQueryProcessor (`enhanced_multi_query_processor.py`)

Processore avanzato per query multi-sorgente:

### Risoluzione Query

python

```
def resolve_sql_query(self, query_config):
    """
    Supporta:
    - SQL inline (string/array)
    - File SQL esterni (.sql)
    - Template con parametri {param_name}
    """
```

### Tipi SQL Supportati

#### 1. SQL Inline Array:

json

```
{
  "sql": [
    "SELECT sistema_id, nome_sistema",
    "FROM dbo.DWHMON_SOX_RISORSE",
    "WHERE classificazione_sox = 'CRITICO'"
  ]
}
```

#### 2. File SQL Esterno:

json

```
{
  "sql_file": "DWH-IA-STD.sql"
}
```

#### 3. Template SQL:

json

```
{
  "sql_template": "queries/template.sql",
  "parameters": {
    "start_date": "2025-01-01",
    "schema_name": "DWH"
  }
}
```

## Validazione Sicurezza

- Blocco keyword pericolose (DROP, DELETE, TRUNCATE)
- Limite lunghezza query (100KB)
- Timeout configurabile per esecuzione

---

## Database Supportati

### Oracle Database

#### Configurazione

```
json
{
  "oracle_dwh": {
    "type": "oracle",
    "host": "10.50.41.9",
    "port": 1521,
    "service_name": "dwniam",
    "username": "dwh_user",
    "password": "secure_password"
  }
}
```

#### Caratteristiche Specifiche

- **Thick Mode:** Client Oracle nativo
- **Service Name:** Supporto TNS
- **Batch Fetch:** Recupero dati ottimizzato
- **Connection Pooling:** Gestione pool connessioni

### SQL Server

#### Configurazione

json

```
{
  "mssql_dest": {
    "type": "mssql",
    "server": "GRFWWW010VEN01.griffon.local",
    "port": 1433,
    "database": "DataWarehouse",
    "auth_type": "windows",
    "driver": "ODBC Driver 17 for SQL Server"
  }
}
```

## Autenticazione Supportata

- **Windows Authentication:** Trusted\_Connection=yes
  - **SQL Server Authentication:** Username/Password
  - **Schema Management:** Creazione automatica schemi
- 

## Configurazione

### Struttura config.json

json

```
{
  "databases": {
    "source_db": { /* configurazione database */,
    "dest_db": { /* configurazione database */
  },
  "queries": [
    {
      "name": "Login Standard DWH",
      "source_database": "oracle_dwh",
      "destination_database": "mssql_dest",
      "sql_file": "DWH-IA-STD.sql",
      "destination_table": "IA_STD",
      "destination_schema": "DWH",
      "enabled": true
    }
  ],
  "execution": {
    "batch_size": 1000,
    "timeout_seconds": 300,
    "drop_existing_tables": true,
    "log_level": "INFO",
    "log_directory": "logs",
    "query_directory": "queries"
  }
}
```

Parametri di Esecuzione

Parametro	Descrizione	Default	Range
batch_size	Righe per batch	1000	100-10000
timeout_seconds	Timeout query	300	30-3600
drop_existing_tables	Ricrea tabelle	true	true/false
log_level	Livello logging	INFO	DEBUG/INFO/WARNING/ERROR
log_directory	Directory log	logs	path
query_directory	Directory SQL	queries	path

Query Processing

Pipeline di Elaborazione

1. Fase 1: Risoluzione

```
python
```

```
# Risolve SQL da config/file/template  
resolved_sql = processor.resolve_sql_query(query_config)
```

## 2. Fase 2: Validazione

```
python
```

```
# Controlla sicurezza e sintassi  
processor.validate_sql_security(resolved_sql)
```

## 3. Fase 3: Esecuzione

```
python
```

```
# Esegue con timeout e batch  
df = processor.execute_query(query_config)
```

## 4. Fase 4: Trasferimento

```
python
```

```
# Scrive su database destinazione  
processor.write_to_destination(df, dest_config)
```

## Gestione Errori

```
python
```

```
try:  
    results = processor.execute_all_queries()  
except QueryTimeoutError as e:  
    logger.error(f"Timeout query: {e}")  
except DatabaseConnectionError as e:  
    logger.error(f"Errore connessione: {e}")  
except SecurityValidationError as e:  
    logger.error(f"Query non sicura: {e}")
```

## Monitoraggio Progress

```
2025-09-29 11:22:17,924 - INFO - OK: Query Login Standard DWH completata: 1021 righe -> [DWH].[IA_STD]  
2025-09-29 11:22:20,002 - INFO - SCHEMA: Schema [DWH] verificato/creato in [mssql_sviluppo_dest]  
2025-09-29 11:22:20,159 - INFO - DROP: Tabella [DWH].[IA_STD] eliminata  
2025-09-29 11:22:20,266 - INFO - CREATE: Tabella [DWH].[IA_STD] creata
```



# Sistema di Logging

## Configurazione Logging

```
python

# Setup automatico con timestamp
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
log_filename = f"db_transfer_enhanced_{timestamp}.log"

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler(log_filepath, encoding='utf-8'),
        logging.StreamHandler(sys.stdout)
    ]
)
```

## Livelli di Log

Livello	Uso	Esempio
DEBUG	Dettagli SQL	SQL: SELECT * FROM...
INFO	Operazioni normali	OK: Connesso a Oracle
WARNING	Problemi non critici	WARNING: Schema esistente
ERROR	Errori gestibili	ERROR: Connessione fallita

## Log Strutturati

```
2025-09-29 11:22:00,744 - Complex.database_manager - INFO - CONNESSIONE Oracle [oracle_dwh]:
10.50.41.9:1521
2025-09-29 11:22:00,744 - Complex.database_manager - INFO - Service Name: dwniam
2025-09-29 11:22:03,596 - Complex.database_manager - INFO - OK: Connesso con Username/Password (Thick
Mode)
```

# Sicurezza

## Validazione Query

```
python
```

```
DANGEROUS_KEYWORDS = [  
    'DROP', 'DELETE', 'TRUNCATE', 'ALTER', 'CREATE',  
    'EXEC', 'EXECUTE', 'xp_', 'sp_password'  
]  
  
def validate_sql_security(self, sql: str):  
    """Valida che la query sia sicura per l'esecuzione"""  
    sql_upper = sql.upper()  
  
    dangerous_found = [kw for kw in DANGEROUS_KEYWORDS if kw in sql_upper]  
    if dangerous_found:  
        raise SecurityValidationError(f"Query contiene keyword pericolose: {dangerous_found}")
```

## Gestione Credenziali

- **Encryption:** Password in config.json (considerare vault esterni)
- **Windows Auth:** Preferita per SQL Server
- **Least Privilege:** Solo SELECT su database sorgente
- **Connection Timeout:** Timeout automatico connessioni

## Network Security

- **Firewall:** Porte database specifiche (1521 Oracle, 1433 SQL Server)
- **VPN:** Connessioni attraverso rete aziendale
- **SSL/TLS:** Crittografia connessioni quando disponibile

---

## Deployment

### Requisiti Sistema

Python 3.13+  
Oracle Instant Client 19c+  
ODBC Driver 17 for SQL Server  
Windows Server 2019+ / Linux RHEL 8+  
RAM: 8GB+ (16GB raccomandati)  
Storage: 100GB+ per logs e temp files

## Installazione Dipendenze

```
bash
```

```
# Dipendenze Python base
pip install pandas==2.1.0
pip install oracledb==1.4.0
pip install pyodbc==4.0.39
pip install flask==3.0.0
pip install openpyxl==3.1.0
pip install xlswriter==3.1.0

# Oracle Instant Client
# Scaricare da Oracle.com e configurare ORACLE_HOME

# SQL Server ODBC Driver
# Installare Microsoft ODBC Driver 17
```

## Setup Ambiente

```
bash

# 1. Clona repository
git clone <repo-url> Complex
cd Complex

# 2. Configura environment
cp config.json.template config.json
# Edita config.json con le tue credenziali

# 3. Test installazione
python app.py --test-connections

# 4. Prima esecuzione
python app.py
```

## Configurazione Produzione

```
json

{
  "execution": {
    "batch_size": 5000,
    "timeout_seconds": 1800,
    "log_level": "WARNING",
    "log_directory": "/var/log/complex",
    "query_directory": "/opt/complex/queries"
  }
}
```

---

# Troubleshooting

## Problemi Comuni

### 1. Errore Connessione Oracle

ERROR: ('08001', "ORA-12514: TNS:listener does not currently know of service...")

#### Soluzione:

- Verifica `service_name` in config.json
- Controlla connettività di rete: `telnet 10.50.41.9 1521`
- Verifica Oracle Instant Client installato

### 2. Errore SQL Server

ERROR: Provider TCP: Impossibile stabilire la connessione

#### Soluzione:

- Verifica server/porta in config.json
- Controlla Windows Authentication abilitata
- Verifica ODBC Driver 17 installato

### 3. Query Timeout

ERROR: Query timeout dopo 300 secondi

#### Soluzione:

- Aumenta `timeout_seconds` in config.json
- Ottimizza query SQL con indici
- Riduci `batch_size` per query complesse

### 4. Query Disabilitate

WARNING: Nessuna query abilitata da eseguire

**Causa:** Tutte le query hanno `"enabled": false` nel config.json

#### Soluzione:

json

```
{
  "queries": [
    {
      "name": "Login Standard DWH",
      "enabled": true, // ← Cambiare da false a true
      "source_database": "oracle_dwh",
      "destination_database": "mssql_sviluppo_dest",
      "sql_file": "DWH-IA-STD.sql"
    }
  ]
}
```

## 5. File SQL Mancanti

ERROR: Memory error durante processing

### Soluzione:

- Riduci `batch_size` (es. 500)
- Aumenta RAM sistema
- Usa query con LIMIT/ROWNUM per test

## Log Debugging

bash

*# Abilita debug completo*

```
{
  "execution": {
    "log_level": "DEBUG"
  }
}
```

*# Analizza log specifico*

`tail -f logs/db_transfer_enhanced_20250929_112200.log | grep ERROR`

## Test Performance

python

```
# Test di performance query
```

```
import time
start_time = time.time()
results = processor.execute_query(query_config)
execution_time = time.time() - start_time
print(f"Query completata in {execution_time:.2f} secondi")
```

---

## API Reference

### DatabaseManager

#### Metodi Principali

```
python
```

```
class DatabaseManager:
    def __init__(self, config: Dict[str, Any])

    @contextmanager
    def get_oracle_connection(self, db_name: str)

    @contextmanager
    def get_mssql_connection(self, db_name: str)

    def test_connection(self, db_name: str) -> bool

    def test_all_connections(self) -> Dict[str, bool]

    def ensure_schema_exists(self, db_name: str, schema_name: str)
```

### EnhancedMultiQueryProcessor

#### Metodi Principali

```
python
```

```

class EnhancedMultiQueryProcessor:
    def __init__(self, config: Dict[str, Any])

    def resolve_sql_query(self, query_config: Dict[str, Any]) -> str

    def execute_query(self, query_config: Dict[str, Any]) -> Tuple[pd.DataFrame, str, str]

    def execute_all_queries(self) -> Dict[str, Any]

    def generate_sample_query_files(self)

    def validate_sql_security(self, sql: str)

    def write_dataframe_to_destination(self, df: pd.DataFrame, dest_db: str, table_name: str)

```

## ExcelExporter

### Metodi Principali

```

python

class ExcelExporter:
    def __init__(self, config: Dict[str, Any])

    def export_query_result(self, df: pd.DataFrame, query_name: str) -> Dict[str, str]

    def export_to_excel(self, df: pd.DataFrame, filename: str) -> str

    def export_to_csv(self, df: pd.DataFrame, filename: str) -> str

    def export_with_metadata(self, df: pd.DataFrame, query_config: Dict[str, Any]) -> str

    def export_multiple_queries(self, results: Dict[str, pd.DataFrame]) -> str

    def ensure_output_directory(self)

    def cleanup_old_exports(self, days_to_keep: int = 30)

```

## Configurazione Query

### Formati Supportati

```

python

```

*# SQL Inline*

```
{  
  "name": "Query Name",  
  "sql": "SELECT * FROM table"  
}
```

*# SQL Array (multiriga)*

```
{  
  "name": "Query Name",  
  "sql": [  
    "SELECT col1, col2",  
    "FROM table1 t1",  
    "JOIN table2 t2 ON t1.id = t2.id"  
  ]  
}
```

*# File SQL Esterno*

```
{  
  "name": "Query Name",  
  "sql_file": "query.sql"  
}
```

*# Template SQL*

```
{  
  "name": "Query Name",  
  "sql_template": "template.sql",  
  "parameters": {  
    "param1": "value1",  
    "param2": "value2"  
  }  
}
```

---

## Note Finali

### Roadmap Futuri Sviluppi

1. **PostgreSQL Support:** Aggiunta driver PostgreSQL
2. **Export Avanzati:** PowerBI integration, PDF reports
3. **REST API:** Interfaccia web per configurazione
4. **Scheduler:** Esecuzione automatica query con export
5. **Monitoring:** Dashboard tempo reale
6. **Cloud Support:** Azure SQL, Oracle Cloud



## Contatti

- **Lead Developer:** Daniele Speziale ([daniele.speziale@guest.telecomitalia.it](mailto:daniele.speziale@guest.telecomitalia.it))
- **Senior Developer:** Marco Lucidi ([marco.lucidi@guest.telecomitalia.it](mailto:marco.lucidi@guest.telecomitalia.it))
- **Azienda:** TIM SPA
- **Versione:** Enhanced Multi-Query Processor
- **Data:** 2025

## Licenza

**Copyright © 2025 TIM SPA - All Rights Reserved**

Questo software e la relativa documentazione sono proprietà esclusiva di TIM SPA.  
È vietata la riproduzione, distribuzione o utilizzo non autorizzato.

## Developed by:

- Daniele Speziale ([daniele.speziale@guest.telecomitalia.it](mailto:daniele.speziale@guest.telecomitalia.it)) - System Architecture & Core Development
- Marco Lucidi ([marco.lucidi@guest.telecomitalia.it](mailto:marco.lucidi@guest.telecomitalia.it)) - Database Integration & Query Processing

---

*Documentazione generata il: 30 Settembre 2025 Versione documentazione: 1.0*