```
# database.py
# Database connection and operations module

import psycopg2
import psycopg2.pool
from contextlib import contextmanager
import logging
from datetime import datetime
from typing import List, Dict, Optional, Tuple

class DatabaseManager:
    def __init__(self, host: str = 'localhost', port: int = 5432,
                 database: str = 'network_analyzer', user: str = 'postgres',
                 password: str = 'password', min_conn: int = 1, max_conn: int = 20):
        """Initialize database connection pool"""
        self.connection_pool = psycopg2.pool.ThreadedConnectionPool(
            min_conn, max_conn,
            host=host,
            port=port,
            database=database,
            user=user,
            password=password
        )
        self.logger = logging.getLogger(__name__)

    @contextmanager
```

```
def get_connection(self):
    """Context manager for database connections"""
    connection = None
    try:
        connection = self.connection_pool.getconn()
        yield connection
    except Exception as e:
        if connection:
            connection.rollback()
        self.logger.error(f"Database error: {e}")
        raise
    finally:
        if connection:
            self.connection_pool.putconn(connection)


def insert_packet(self, packet_data: Dict) -> bool:
    """Insert packet data into database"""
    query = """
    INSERT INTO packets (source_ip, destination_ip, source_port, destination_port,
                         protocol, packet_size, tcp_flags, node_id, raw_data)
    VALUES (%(source_ip)s, %(destination_ip)s, %(source_port)s, %(destination_port)s,
            %(protocol)s, %(packet_size)s, %(tcp_flags)s, %(node_id)s, %(raw_data)s)
    """
    try:
        with self.get_connection() as conn:
            with conn.cursor() as cursor:
```

```
                cursor.execute(query, packet_data)
                conn.commit()
                return True
        except Exception as e:
            self.logger.error(f"Failed to insert packet: {e}")
            return False

    def insert_alert(self, alert_data: Dict) -> bool:
        """Insert alert data into database"""
        query = """
        INSERT INTO alerts (alert_type, source_ip, destination_ip, severity,
                            description, node_id, count)
        VALUES (%(alert_type)s, %(source_ip)s, %(destination_ip)s, %(severity)s,
                %(description)s, %(node_id)s, %(count)s)
        """
        try:
            with self.get_connection() as conn:
                with conn.cursor() as cursor:
                    cursor.execute(query, alert_data)
                    conn.commit()
                    return True
        except Exception as e:
            self.logger.error(f"Failed to insert alert: {e}")
            return False

    def get_packets(self, filters: Dict = None, limit: int = 1000) -> List[Dict]:
```

```
"""Retrieve packets with optional filters"""
base_query = """
SELECT id, timestamp, source_ip, destination_ip, source_port, destination_port,
       protocol, packet_size, tcp_flags, node_id
FROM packets
"""

conditions = []
params = {}

if filters:
    if 'source_ip' in filters:
        conditions.append("source_ip = %(source_ip)s")
        params['source_ip'] = filters['source_ip']

    if 'destination_ip' in filters:
        conditions.append("destination_ip = %(destination_ip)s")
        params['destination_ip'] = filters['destination_ip']

    if 'protocol' in filters:
        conditions.append("protocol = %(protocol)s")
        params['protocol'] = filters['protocol']

    if 'start_time' in filters:
        conditions.append("timestamp >= %(start_time)s")
        params['start_time'] = filters['start_time']
```

```
        if 'end_time' in filters:
            conditions.append("timestamp <= %(end_time)s")
            params['end_time'] = filters['end_time']

    if conditions:
        base_query += " WHERE " + " AND ".join(conditions)

    base_query += f" ORDER BY timestamp DESC LIMIT {limit}"

    try:
        with self.get_connection() as conn:
            with conn.cursor() as cursor:
                cursor.execute(base_query, params)
                columns = [desc[0] for desc in cursor.description]
                return [dict(zip(columns, row)) for row in cursor.fetchall()]
    except Exception as e:
        self.logger.error(f"Failed to retrieve packets: {e}")
        return []

def get_alerts(self, filters: Dict = None, limit: int = 1000) -> List[Dict]:
    """Retrieve alerts with optional filters"""
    base_query = """
    SELECT id, timestamp, alert_type, source_ip, destination_ip,
           severity, description, node_id, count, resolved
    FROM alerts
```

```
"""

conditions = []
params = {}

if filters:
    if 'alert_type' in filters:
        conditions.append("alert_type = %(alert_type)s")
        params['alert_type'] = filters['alert_type']

    if 'source_ip' in filters:
        conditions.append("source_ip = %(source_ip)s")
        params['source_ip'] = filters['source_ip']

    if 'severity' in filters:
        conditions.append("severity >= %(severity)s")
        params['severity'] = filters['severity']

    if 'resolved' in filters:
        conditions.append("resolved = %(resolved)s")
        params['resolved'] = filters['resolved']

    if 'start_time' in filters:
        conditions.append("timestamp >= %(start_time)s")
        params['start_time'] = filters['start_time']
```

```
        if 'end_time' in filters:
            conditions.append("timestamp <= %(end_time)s")
            params['end_time'] = filters['end_time']

    if conditions:
        base_query += " WHERE " + " AND ".join(conditions)

    base_query += f" ORDER BY timestamp DESC LIMIT {limit}"

    try:
        with self.get_connection() as conn:
            with conn.cursor() as cursor:
                cursor.execute(base_query, params)
                columns = [desc[0] for desc in cursor.description]
                return [dict(zip(columns, row)) for row in cursor.fetchall()]
    except Exception as e:
        self.logger.error(f"Failed to retrieve alerts: {e}")
        return []

def get_traffic_stats(self, time_range_hours: int = 24) -> Dict:
    """Get traffic statistics for the specified time range"""
    query = """
    SELECT
        protocol,
        COUNT(*) as packet_count,
        COUNT(DISTINCT source_ip) as unique_sources,
```

```
            COUNT(DISTINCT destination_ip) as unique_destinations
        FROM packets
        WHERE timestamp >= NOW() - INTERVAL '%s hours'
        GROUP BY protocol
        ORDER BY packet_count DESC
        """

        try:
            with self.get_connection() as conn:
                with conn.cursor() as cursor:
                    cursor.execute(query, (time_range_hours,))
                    columns = [desc[0] for desc in cursor.description]
                    return [dict(zip(columns, row)) for row in cursor.fetchall()]
        except Exception as e:
            self.logger.error(f"Failed to get traffic stats: {e}")
            return []

    def cleanup_old_data(self, days_to_keep: int = 30):
        """Remove old packets and resolved alerts"""
        try:
            with self.get_connection() as conn:
                with conn.cursor() as cursor:
                    # Remove old packets
                    cursor.execute(
                        "DELETE FROM packets WHERE timestamp < NOW() - INTERVAL '%s
days'",
                        (days_to_keep,)
```

```
                )
                packets_deleted = cursor.rowcount

                # Remove old resolved alerts
                cursor.execute(
                    "DELETE FROM alerts WHERE resolved = true AND timestamp < NOW() - INTERVAL '%s days'",
                    (days_to_keep,)
                )
                alerts_deleted = cursor.rowcount

                conn.commit()
                self.logger.info(f"Cleaned up {packets_deleted} old packets and {alerts_deleted} resolved alerts")

        except Exception as e:
            self.logger.error(f"Failed to cleanup old data: {e}")

    def close_connections(self):
        """Close all database connections"""
        if self.connection_pool:
            self.connection_pool.closeall()
```