In [1]:
```python
import re
import pandas as pd
from datetime import datetime
from typing import Dict, List, Tuple, Optional
from dataclasses import dataclass
from enum import Enum

class ValidationSeverity(Enum):
    ERROR = "ERROR"
    WARNING = "WARNING"
    INFO = "INFO"

@dataclass
class ValidationResult:
    is_valid: bool
    errors: List[Dict]
    warnings: List[Dict]
    info: List[Dict]
    sanitized_data: Optional[Dict] = None

class TransactionValidator:
    def __init__(self):
        # Define required fields and their types
        self.required_fields = {
            'transaction_id': str,
            'transaction_date': str,
            'amount': (int, float),
            'merchant_name': str,
            'mcc': str,
            'card_number': str,  # Will be masked/tokenized
        }

        # PCI DSS compliant patterns
        self.card_number_pattern = r'^\d{13,19}$'
        self.valid_mcc_codes = set(['5411', '5812', '5541', '5311', '5999'])  # Exampl

    def validate_transaction(self, transaction: Dict) -> ValidationResult:
        """
        Validate a single transaction against all compliance requirements.
        Returns ValidationResult with details about any issues found.
        """
        errors = []
        warnings = []
        info = []
        sanitized_data = transaction.copy()

        # 1. Check for required fields and their types
        self._validate_required_fields(transaction, errors)

        # 2. Validate and mask PCI sensitive data
        self._validate_and_mask_sensitive_data(sanitized_data, errors, warnings)

        # 3. Validate business rules
        self._validate_business_rules(transaction, errors, warnings)

        # 4. Validate data format and ranges
        self._validate_data_formats(transaction, errors, warnings)
```

```
        # Determine overall validation status
        is_valid = len(errors) == 0

        return ValidationResult(
            is_valid=is_valid,
            errors=errors,
            warnings=warnings,
            info=info,
            sanitized_data=sanitized_data if is_valid else None
        )

    def _validate_required_fields(self, transaction: Dict, errors: List[Dict]):
        """Validate presence and types of required fields."""
        for field, expected_type in self.required_fields.items():
            if field not in transaction:
                errors.append({
                    'field': field,
                    'severity': ValidationSeverity.ERROR,
                    'message': f"Missing required field: {field}",
                    'requirement': 'COMPLIANCE-001'
                })
                continue

            if not isinstance(transaction[field], expected_type):
                errors.append({
                    'field': field,
                    'severity': ValidationSeverity.ERROR,
                    'message': f"Invalid type for {field}. Expected {expected_type}, g
                    'requirement': 'COMPLIANCE-002'
                })

    def _validate_and_mask_sensitive_data(self, transaction: Dict, errors: List[Dict],
        """Validate and mask PCI-sensitive data."""
        # Validate card number format
        if 'card_number' in transaction:
            card_num = transaction['card_number']
            if not re.match(self.card_number_pattern, card_num):
                errors.append({
                    'field': 'card_number',
                    'severity': ValidationSeverity.ERROR,
                    'message': "Invalid card number format",
                    'requirement': 'PCI-DSS-001'
                })
            # Mask card number - keep only last 4 digits
            transaction['card_number'] = f"****-****-****-{card_num[-4:]}"

        # Remove any CVV data if present
        if 'cvv' in transaction:
            errors.append({
                'field': 'cvv',
                'severity': ValidationSeverity.ERROR,
                'message': "CVV data should never be included in settlement data",
                'requirement': 'PCI-DSS-002'
            })
            del transaction['cvv']

    def _validate_business_rules(self, transaction: Dict, errors: List[Dict], warnings
        """Validate business rules and transaction logic."""
        # Validate MCC code
        if 'mcc' in transaction and transaction['mcc'] not in self.valid_mcc_codes:
```

```
            warnings.append({
                'field': 'mcc',
                'severity': ValidationSeverity.WARNING,
                'message': f"Unknown MCC code: {transaction['mcc']}",
                'requirement': 'BUSINESS-001'
            })

        # Validate transaction amount
        if 'amount' in transaction:
            amount = transaction['amount']
            if amount <= 0:
                errors.append({
                    'field': 'amount',
                    'severity': ValidationSeverity.ERROR,
                    'message': "Transaction amount must be positive",
                    'requirement': 'BUSINESS-002'
                })
            elif amount > 50000:  # Example threshold
                warnings.append({
                    'field': 'amount',
                    'severity': ValidationSeverity.WARNING,
                    'message': "Unusually large transaction amount",
                    'requirement': 'BUSINESS-003'
                })

    def _validate_data_formats(self, transaction: Dict, errors: List[Dict], warnings:
        """Validate data formats and ranges."""
        # Validate date format
        if 'transaction_date' in transaction:
            try:
                datetime.strptime(transaction['transaction_date'], '%Y-%m-%d')
            except ValueError:
                errors.append({
                    'field': 'transaction_date',
                    'severity': ValidationSeverity.ERROR,
                    'message': "Invalid date format. Expected YYYY-MM-DD",
                    'requirement': 'FORMAT-001'
                })

# Example usage
validator = TransactionValidator()

# Example transaction
sample_transaction = {
    'transaction_id': 'T123456',
    'transaction_date': '2024-01-09',
    'amount': 99.99,
    'merchant_name': 'ACME STORE',
    'mcc': '5411',
    'card_number': '4111111111111111'
}

# Validate transaction
result = validator.validate_transaction(sample_transaction)

# Check results
if result.is_valid:
    print("Transaction is valid. Sanitized data:", result.sanitized_data)
else:
    print("Validation failed:")
```

```
    print("Errors:", result.errors)
    print("Warnings:", result.warnings)
```

```
Transaction is valid. Sanitized data: {'transaction_id': 'T123456', 'transaction_dat
e': '2024-01-09', 'amount': 99.99, 'merchant_name': 'ACME STORE', 'mcc': '5411', 'car
d_number': '****-****-****-1111'}
```

In [ ]: