SQL Server と InfoPath，Access による教務システム OBA 開発の技術と手法

兵庫県立西宮香風高等学校教諭　総務部システム管理課長
松本 吉生

1. システム開発に必要な技術と手法

和風建築の家を建てるには木組みや瓦葺の技術，軽量鉄骨のプレハブ住宅を建てるには鉄骨の強度計算や防錆塗装の技術，高層建築を立てるには鉄筋コンクリート構造の技術が必要であるように，作りたい建築物によって必要な技術が異なる。システム構築もこれと同じように，どのようなシステムを作りたいかによって必要な技術が異なる。またどんな技術が使えるかによってシステムを構築する手法が異なり，できたシステムの能力に違いが出る。本稿では SQL Server と InfoPath，Access による教務システム OBA 開発における技術と手法を解説する。

2. T-SQL とストアドプロシージャ

SQL Server と InfoPath，Access による教務システム OBA 開発において，InfoPath はデータを入出力するためだけに，Access は出力データを帳票に成型して印刷するためだけに利用し，データ処理のロジックはすべて SQL Server に実装する。SQL Server 内でデータ処理を行うには T-SQL という SQL を拡張した言語を用い，SQL Server 内部にプログラムを格納し実行する。これをストアドプロシージャという。ストアドプロシージャは SQL Server 内で実行されるプログラムで，このようなプログラムによる開発をサーバーサイド開発ともいう。データベースサーバーが T-SQL のような高機能のプログラム開発･実行環境を持たない時代には，データベースは単にデータを保存する箱として扱い，データ処理はクライアントアプリケーションがデータを呼び出して行い，データベースに書き込むという手法が使われた。この手法はクライアントサイド開発であり，データ処理はクライアントソフトウェアに実装された。この手法ではデータ処理にあたって常にネットワークを介してデータをクライアントに呼び出さなければならず，ネットワークに負荷がかかり，大量のデータ処理を行う場合に時間がかかる，場合によってはネットワークの障害などでデータが正常に処理されなかったりする問題があった。T-SQL によるストアドプロシージャではデータ処理が SQL Server 内だけで行われるので複雑な処理を高速かつ完全に行うことができる。

3. テーブル構造と正規化

システム構築においてデータベースのテーブル構造の決定は極めて重要である。どのようなテーブル構造をとるかによって，必要な処理がうまく処理できたり困難になったりする。場合によってはデータの整合性が失われてシステム運用が破綻する。データベースの設計については正規化の概念があり，データベース内で同じ情報が 2 回以上記録されないように，一意の情報を一か所に格納することが望ましいとされる。しかしこのことを機械的に適用すると，扱うデータが複雑になればなるほどテーブル間のリレーションが多くなり，あらゆる処理について複雑なリレーションを持たせなければならなくなってしまう。合理的であるはずの理論によって現実の処理が非合理的になるという本末転倒の現象がおきる。正規化の理論は押さえた上で，運用に対する合理的なテーブル設計が必要なのである。

たとえば授業の出欠記録をとる「出欠」テーブルを考えたとき，必要な情報をすべて正規化すると出欠記録を行うためには「生徒」，「性別コード」，「教員」，「講座」，「日程」，「時限」など数多くの基礎データを保存したテーブルとのリレーションが必要となる。これでは処理が複雑になるので，「出欠」テーブルに処理に必要なデータをすべて持たせることにする。つまり正規化の概念では「出欠」テーブルに「学籍番号」を持ち，「生徒名」は「生徒」テーブルに持ち，二つのテーブルを「学籍番号」でリレーションすることになるが，そうするのではなく「出欠」テーブルにも

「生徒名」を持たせ，「生徒」テーブルとのリレーションなしに生徒名を表示できるようにする。同様に「教員名」など他の項目についても基礎テーブルとのリレーションなしに必要な情報が表示できるように，すべて「出欠」テーブルに基礎テーブルと重複する形でデータを持たせる。

生徒名を「生徒」テーブルと「出欠」テーブルに重複して持たせると生徒名が変更になったときにデータの整合性が失われるのではないかと思われるが，後述するストアドプロシージャやトリガを使ってデータの整合性をとる手法がある。

4. SQL Server の基本的なデータ処理技術

SQL Server の最新バージョンは SQL Server 2008R2 である。筆者は SQL Server 2000 から開発をはじめ，現在は SQL Server 2005 を使ってシステムを開発・運用している。これら SQL Server の各バージョンはデータベースを運用するための様々な仕組みを有しており，バージョンによって違いがあるが，データ処理に関する基本的な次の機能を有している。

(1) ストアドプロシージャ

ストアドプロシージャは SQL Server 内に格納されるプログラムで，T-SQL で記述される。一連のデータ処理をバッチファイルのように記述し実行することができる。たとえば学期末に出欠や評価，講座基礎データから修得単位を求め，成績一覧表や通知表のデータを生成するといった処理を「学期末処理」といったストアドプロシージャにまとめておき，一括で実行することができる。ストアドプロシージャには引数も設定できる。

次の例は実行時に引数の記述を含めて「テスト」テーブルの「テスト内容」フィールドにデータを書き込む「テストストアドプロシージャ」という名前のストアドプロシージャを生成する T-SQL 文である。きわめて単純であるがストアドプロシージャの基本形を示している。

```
create procedure テストストアドプロシージャ
@ P_ 引数 as varchar(50)
as
insert into テスト(テスト内容)
values (' テスト書き込み：引数は「' + @ P_ 引数 + '」でした。')
```

Fig. 1　ストアドプロシージャのサンプル

ストアドプロシージャの実行は以下のように execute コマンドを使う。引数はストアドプロシージャの名前の後に加える。テキストはシングルクオーテーションの引用符で囲む。

```
execute テストストアドプロシージャ ' これは引数です '
```

Fig. 2　ストアドプロシージャの実行例

テスト内容
テスト書き込み：引数は「これは引数です」でした。

Fig. 3　ストアドプロシージャの実行結果

(2) トリガ

トリガはテーブルにデータが追加，変更，削除されたときに自動的に実行されるプログラムで，テーブルに対して設定される。トリガを生成する次のストアドプロシージャを実行すると「生徒」テーブルに「生徒のテストトリガ」の名前のトリガが設定され，「生徒」テーブルに新しいデータが追加されたときに「テスト」テーブルの「テスト内容」フィールドにデータが追加されたことが記録される。

```
create trigger 生徒のテストトリガ on 生徒 after insert
as
insert into テスト(テスト内容)
values ('「生徒」テーブルにデータが追加されました。')
```

Fig. 4　トリガを生成するストアドプロシージャの例

「生徒のテストトリガ」が設定された「生徒」テーブルに新しいデータを追加すると，自動的にトリガが実行され，「テスト」テーブルの「テスト内容」フィールドに記録がされる。

テスト内容
「生徒」テーブルにデータが追加されました。

Fig. 5　トリガの実行結果

(3) ジョブ

ストアドプロシージャは execute コマンドで実行され，トリガはテーブルに対するデータの追加，変更，削除をイベントとして実行される。これに対して一定の処理をスケジュールにより実行するものがジョブである。ジョブには SQL 文やストアドプロシージャを設定することができる。

ジョブを実行するのはSQL Server自身ではなく「SQL Server エージェント」というサービスであり，SQL Server Management Studioのオブジェクトエクスプローラから管理できる。

Fig. 6　SQL Server エージェントの管理

ジョブの作成と管理はSQL Serverエージェントの「ジョブ」フォルダで行う。ジョブで設定する基本項目はステップとスケジュールである。

ステップには実行させるSQL文やストアドプロシージャの実行文を記述する。複数の実行文を1つのステップにまとめて設定することや複数のステップに分けて処理することができる。

スケジュールではジョブを実行するタイミングを設定する。実行する日付と時間を指定し，一回だけの実行なのか定期的な実行なのか，定期的な実行ならば毎日決まった時間に実行させることや繰り返しの単位を細かく指定することができる。

次に示すジョブステップの例では，1つのステップに1つのSQL文と2つのストアドプロシージャの3つの文をまとめて記述している。

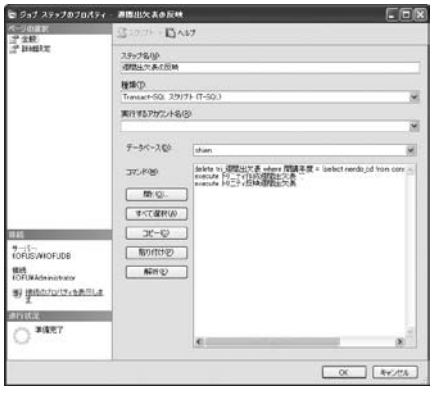

Fig. 7　ジョブステップの例

次に示すジョブスケジュールの例では，このジョブを毎日3:00に実行する設定にしている。サーバーに負荷のかかる処理は，通常業務のない深夜に実行することが望ましいからだ。

Fig. 8　ジョブスケジュールの例

5. SQL Serverの技術とテーブル設計

既に述べたように，システム構築ではテーブル設計が重要なポイントになる。このとき前述したSQL Serverの技術を前提としてテーブル設計を行う。筆者の経験では以下の点が設計の目安になる。

(1) 一意のデータと重複して生成するデータ

生徒についての氏名や生年月日のように，ただ1つの一意のデータでよいものは学籍番号をキーにして1つのテーブルに格納する。所属部活動のように複数のデータが生成するものは別テーブルで管理する。

(2) 変更の履歴を記録する

生徒についての氏名データは一意のものであるが，場合によっては在学中に氏名が変更になる場合がある。このときテーブルの氏名データを書き換えるだけでは，履歴を残すことができない。氏名変更の履歴を残す手法として，氏名変更のデータを「氏名変更」テーブルのような別テーブルに記録することとし，トリガによって追加データを「生徒」テーブルに間接的に反映させる。こうすることで「氏名変更」テーブルに変更履歴が蓄積される。

(3) 削除権限の取り扱い

データの削除は慎重に行わなければいけない。できれば一般ユーザーには削除権を与えないことが望ましい。テーブルの削除権の設定はテーブルのプロパティで行う。(Fig. 9)

削除を必要とする処理には「削除」フィールドを作り，実際にデータを削除するのではなくフィールドのデータによって削除扱いになる処理を行う。

(4) 集計する必要があるデータをコード化する

データのコード化のメリットとして全体のデータ

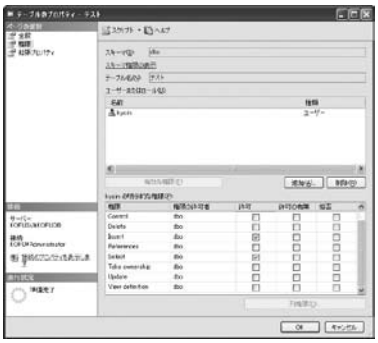

Fig. 9　選択と挿入の権限のみを与えた例

量を減らすことがある。繰り返しの多い項目について文字列よりコードで記録する方が一般にデータ量を減らすことができるだろう。しかし今日的なサーバーの記憶容量を考えると，多少のデータ量を減らすことよりも別テーブルを作りリレーションによって複雑化することを避けた方がよい場合もある。

保存する記録のどの項目をコード化すればよいかについては，集計する必要があればコード化し，その必要がなければ単純に文字列で記述すればよい。たとえば生徒の健康診断の情報について，疾病ごとに何らかの集計が必要ならばコード化を考え，集計の必要がない項目は自由に文字列で記述することにする。

(5) データのライフサイクルに注目する

処理するデータが永続的に一意のもの，不定期に追加されるもの，年度単位で生成するもの，日々更新されるものなどライフサイクルに注目してテーブル構造を考える。

生徒の生年月日のように永続的に一意で変更のないものは単純に1テーブルの1フィールドで管理する。住所データのように転居によって不定期に更新されるものは別テーブルで管理するかトリガを使った履歴記録の手法をとる。クラス出席番号のような年度単位で管理するデータは年度データをフィールドに持たせて管理する。出欠情報のように日々更新されるものは日付データをフィールドに持たせて管理する。

(6) スナップショットの必要性を検討する

生徒の出欠情報を日々処理している場合，リアルタイムで出欠状況を把握したい。このときは直接データを集計するクエリを作成して表示させることになる。しかし特定のタイミングでスナップショットを必要とする場合がある。

たとえば出欠情報をもとにして通知表を作成する場合などは直接の集計では運用上望ましくない。仮に学期末に通知表を印刷して生徒に配布した後，出欠記録の間違いがわかったとしよう。通知表が出欠データを直接に集計している場合，出欠記録を訂正した時点で通知表の出欠データが更新されるが，このとき既に手渡した通知表の記載がどうであったかが追跡できなくなる。したがって通知表のように，学校から保護者や生徒に手渡される帳票データについては必ずスナップショットをとり，渡したデータの記録を参照できるようにしておくべきである。この場合は通知表データをストアドプロシージャによって別テーブルにスナップショットを保存し，そこから帳票を生成するシステムにしておく。

(7) データの入力者をテーブルに記録する

多くの職員がデータベースを利用する場合，個々の処理を誰が行ったかを記録することを考えたい。そのためにはテーブルに「処理者」フィールドを用意し，職員番号などの ID が自動的に記録されるようにする。T-SQL には実行者の ID を得る関数があり，トリガと組み合わせて利用する。次のトリガの例は「テスト」テーブルにデータを挿入したユーザーの ID を自動的に記録するサンプルである。

```
create trigger テストのテストトリガ on テスト after insert
as
update テスト
set 処理者 = suser_sname()
where テスト . 管理番号 = (select 管理番号 from inserted)
```

Fig. 10　処理者の ID を記録するトリガの例

6. まとめ

システムの構築の技術は日進月歩であり，その方向性は高度な処理を簡単に実装できる方向にすすんでいる。プログラミング技法などの習熟に必要な時間は少なくてすみ，かつ多くの技術情報が Web などから入手できる時代となった。必要なのは仕事内容そのものの理解であり，それをシステムに落とし込む論理的思考力である。どの学校にもそれぞれの現場のニーズに応じた処理の形があるはずであり，校務の合理化のためにもデータベースを活用したシステム構築をぜひ試していただきたい。