



ホーム | 製品 | サービス & ソリューション | サポート & ダウンロード | マイアカウント



DB2 Developer Domain > 製品別技術情報 > DB2いろはがるた >

DB2いろはがるた



第36回

「あ」- アクセスパス、SQL実行の計画書



執筆者

春野 さくら

「DB2いろはがるた」を執筆するために参上した、なぞの女性。日本の古典文学を愛する。

皆様、明けましておめでとうございます。このいろはカルタも昨年の3月に連載開始以来、早いものもので、もう「あ」まで進んで参りました。本年も、少しおつきあいくださいますようお願いいたします。さて、さっそくですが、新年のいろは初めはアクセスパスのお話しです。

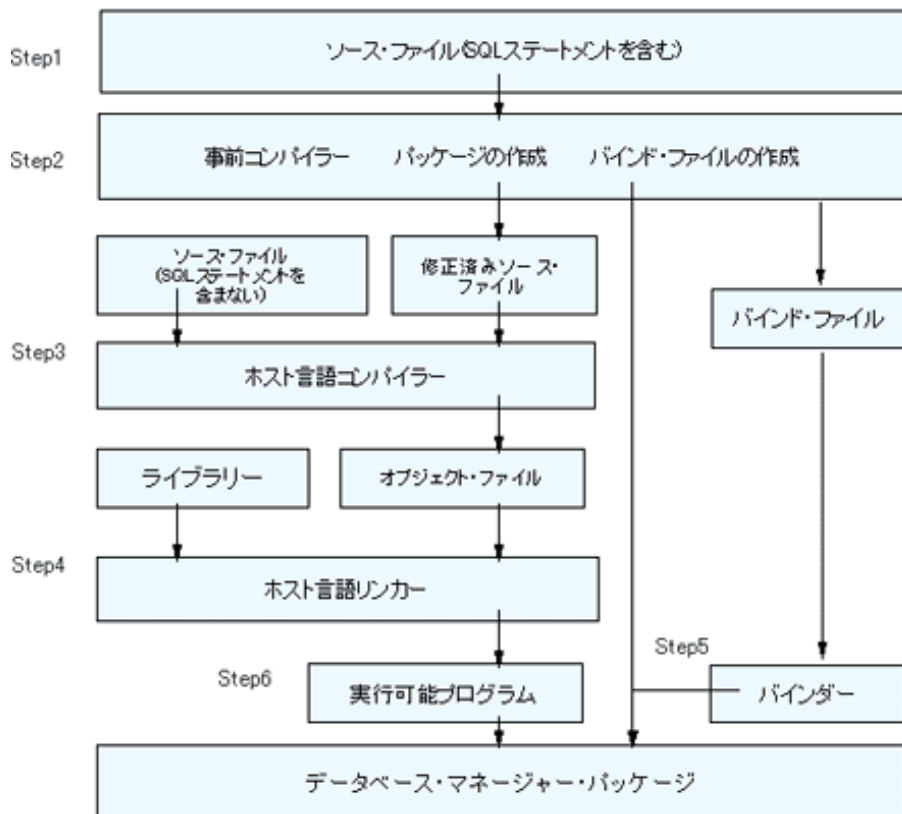


DBMSのオプティマイザーの代表的なタイプとして、ルールベースとコストベースの2種類があります。ルールベースとはオプティマイザーがソースコードのSQL文のWhere文節や、索引の有無にもとづいて、アクセスパスを作成する方法です。それに対してコストベースとはオプティマイザーがDBMSの中にある統計情報に基づいてアクセスパスを作成する方法で、通常SQL文の書き方には左右されません。DB2 UDBはそもそもの製品の最初のバージョンからコストベース・オプティマイザーを採用していました。(ここはDB2の開発者が業界1だといって、ものすごく自信を持っているところです。)DB2ではOracle製品と違って、ルールベース・オプティマイザーは提供していません。そこで当然、Oracleでいうヒント関数というものもありません。

それではDB2ではアクセスパスはいつ作成されるのでしょうか？DB2 UDBではSQLコンパイラーがアクセスパスを作成する処理のことをバインドと呼びます。そのバインド処理には2つのタイミングがあります。というのは、DB2には埋め込みSQLプログラミングの方式に静的と動的といわれるふたとおりの処理の方法があるからです。DML文の種類や、列名、表名などがSQL文の実行時より前に既に決定されており、あらかじめ実行前にアクセスパスを作ることができるSQL文を静的SQL文と呼びます。それに対して対話式プロ

グラムのように実行時まで、SQL文の中の列名や表明がわからないようなものを動的SQL文と呼びます。静的SQL文は実行前にPrepまたはBindコマンドを実行することにより、バインド処理が行われ、アクセス・プランがシステム・カタログの中に保存されます。アクセス・プランはセクションとも呼ばれ1つのSQL文に対して一つ作成されます。実際にシステム・カタログに保管されるのはソース・プログラム・モジュール一個に対して一つのパッケージが作成され、その中に複数のセクションが含まれることになります。これらのパッケージはSQL文が実際に実行される時に使われ、DB2がシステム・カタログ表に保管されている情報を使用して、データにアクセスする方法と、照会の結果を提供します。

### 埋め込み SQL のプログラミング開発手順



組み込み動的SQL文が入っているプログラミング・モジュールは関連したパッケージとセクションを持っていますが、セクションは動的に準備されるSQL文を保管される場所として使用されます。組み込み動的SQL文の場合、アクセス・プランはセクションには保管されません。というのは、文が実行されるまでは、アクセス・プランが作成されないからです。

バインド・ファイルを作成し、別のステップでパッケージにバインドすることを据え置きバインドと呼んでいます。その場合には、次の2ステップの処理が必要です。

バインド・ファイルを作成する

```
connect to db2cert
prep prog1.sqc bindfile
```

このコマンドによって、ソース・モジュールのパッケージの作成に必要なデータが全てprog1.bndという名前のファイルに保管されます。

バインド・ファイルをデータベースにバインドする

```
connect to db2cert
bind prog1.bnd
```

このバインド処理によって、バインド・ファイルの中のSQL文が検査され、現行データベース統計を使用して、最良のデータ・アクセスプランが判別されま

す。この時点で、DB2のオプティマイザーによってアクセス・プランが選択されることとなります。このアクセスプランはシステム・カタログ表に保管されませんが、前にもいったようにアクセス・プランが作成されるのは静的SQL文の場合だけです。組み込み動的SQL文の場合には、パッケージとセクション番号が割り当てられますが、文が実行されるまでは、アクセスプランは作成されません。

バインド・ファイルが作成されると、パッケージにタイム・スタンプが保管されます。このタイム・スタンプは整合性トークンと呼ばれます。これはアプリケーションが正しいSQL文を実行することを確認するために使われます。実行可能アプリケーションはパッケージ名とセクション番号に基づいてSQL文の実行を試みます。必要なパッケージとセクションが見つからない場合は、アプリケーションは次のようなエラー・メッセージを戻します。

```
SQL0805N パッケージ「pkgschema.pkgname」が見つかりませんでした。
```

必要なパッケージとセクションがシステム・カタログに存在する場合は、タイム・スタンプが検査されます。実行可能アプリケーションのタイム・スタンプがシステム・カタログ表に保管されているタイム・スタンプと一致していない場合は次のメッセージが戻されます。例えばバインド処理の後で、誰かがこっそりとソース・コードを変更したような場合に起こります。

```
SQL0818N タイムスタンプの矛盾が起きました。
```

こうなったら、このままではそのアプリケーションは実行できません。元にもどってバインド処理をやり直すこととなります。時間というのは重要なものです。みなさん待ち合わせには遅刻しないでゆきましょう！（アクセスパスとは何も関係ない話ですみません、最近どうも遅刻してしまうことがよくあり、ひんしゅくをかっています。）

ご参考までに、バインド・ファイルとパッケージのタイムスタンプが一致しているかどうかを検証する方法を説明します。バインド・ファイルの中のタイムスタンプを調べるにはdb2bfdと呼ばれるツールを使います。そうするとその出力の中に暗号化されたタイムスタンプが見つかります。一方システム・カタログの中のパッケージのタイムスタンプを調べるためには、次のSQL文を実行します。

```
select pkgschema, pkgname, unique_id from syscat.packages
where pkgname='prog1'
```

この結果の表示の中のunique\_idが先ほどのタイムスタンプと一致すれば、しめたものです。何も問題ありません。SQL文は正常に実行できることとなります。めでたし、めでたしです。

今回は新年早々なんか、こちゃこちゃとした話しになってしまいました。何か夢がたりないようで、反省！まあ、なにはともあれ、本年もよろしゅうお願いいたします。

[↑ 上に戻る](#)