パッケージシステムの導入を成功に導くインタビューの実践

武田薬品工業株式会社

■ 執筆者Ｐｒｏｆｉｌｅ ■

西　本　博　之

1987 年　武田薬品工業（株）入社.
　　生産管理システムの開発を担当.
　　製配販一貫管理システムの開発を担当.
1992 年　ＧＣＰ業務支援システムの開発・運用を担当.
　　現在，医薬開発本部開発推進部システムG所属.

■ 論文要旨 ■

パッケージシステム（DDWorks21：富士通製）を導入することにより，メインフレームで構築したＧＣＰ業務支援システム（COBOL：60 万ステップ程度）を 1999 年 12 月までにダウンサイジングすることができた．しかし，パッケージシステムに対して変に機能追加すると，元の方がまだ良かったという悲惨な局面が発生した.

パッケージシステムの導入を成功に導くには，無用なカスタマイズをしないことであり，やむなく機能追加する場合には，システム構造に合った的確なシステム化要件を作成することである．そのためには，ユーザに対して，システムの特性に応じた効果的なインタビューが実践できるかがポイントとなる.

ユーザ企業のシステム部門担当者の視点で，これらを実践するためのインタビューのポイント，システム設計のポイントについて，六つの事例をあげて考察した.

■ 論文目次 ■

■ 図表一覧 ■

１．はじめに

医薬品の開発では，医薬品としての適否を判断するための最終段階として，臨床試験（治験）が実施される．この段階では，初めて「ヒト」を対象にするということで，高度な科学性とともに厳格な倫理性が要求される．法規制としては，「医薬品の臨床試験の実施に関する基準（ＧＣＰ：Good Clinical Practice）」が定められている．また，1996 年５月に開催された日・米・ＥＵ三極医薬品承認審査ハーモナイゼーション国際会議（ＩＣＨ）で，ＩＣＨ－ＧＣＰが最終合意に達し，日・米・ＥＵ三極共通のＧＣＰ案が確定された．臨床試験におけるグローバリゼーションの幕開けである．このことを受け，日本におけるＧＣＰも 1997 年３月に改正され，それ以前のＧＣＰと区別して，新ＧＣＰと呼ばれている．

このようにシステムを取り巻く環境が大きく変化し，弊社においても，下記の課題を短期間に解決することが当時求められた．

①新ＧＣＰ（法規制の改訂）への対応
②西暦 2000 年問題への対応
③ダウンサイジングへの要求（ＧＵＩによる操作性の向上）
④メインフレーム（GS8400）で構築した業務システム（COBOL：60 万ステップ程度）の老朽化への対応

このため弊社システム部門では，Sigma21（**S**ystem of **I**ntegrated and **G**lobal **M**anagement for **A**pplication）と命名した中期計画を立案し，システム化を推進した．

２．ＧＣＰ業務支援システムの概要

Sigma21 を構成するシステム群の中で，臨床試験の手続き面を主としてサポートするシステムを弊社では，ＧＣＰ業務支援システムと呼んでいる．このＧＣＰ業務支援システムが取り扱う情報は下記のとおりである．（**図１**参照）

①厚生省へ臨床試験の開始を届け出る治験計画届の情報
②臨床試験を依頼する医療機関及び担当する医師を選定した情報
③各医療機関へ臨床試験を依頼した情報
④各医療機関からの臨床試験の実施可否の通知
⑤各医療機関との契約情報
⑥各医療機関に提供（交付）した薬剤の情報
⑦被験者が臨床試験へ参加することを同意した情報
⑧被験者への投薬を開始した情報
⑨被験者への投薬が終了した情報
⑩各医療機関から未使用の薬剤を回収した情報
⑪各医療機関からの臨床試験終了の通知
⑫厚生省へ臨床試験の終了を届け出る治験届の情報

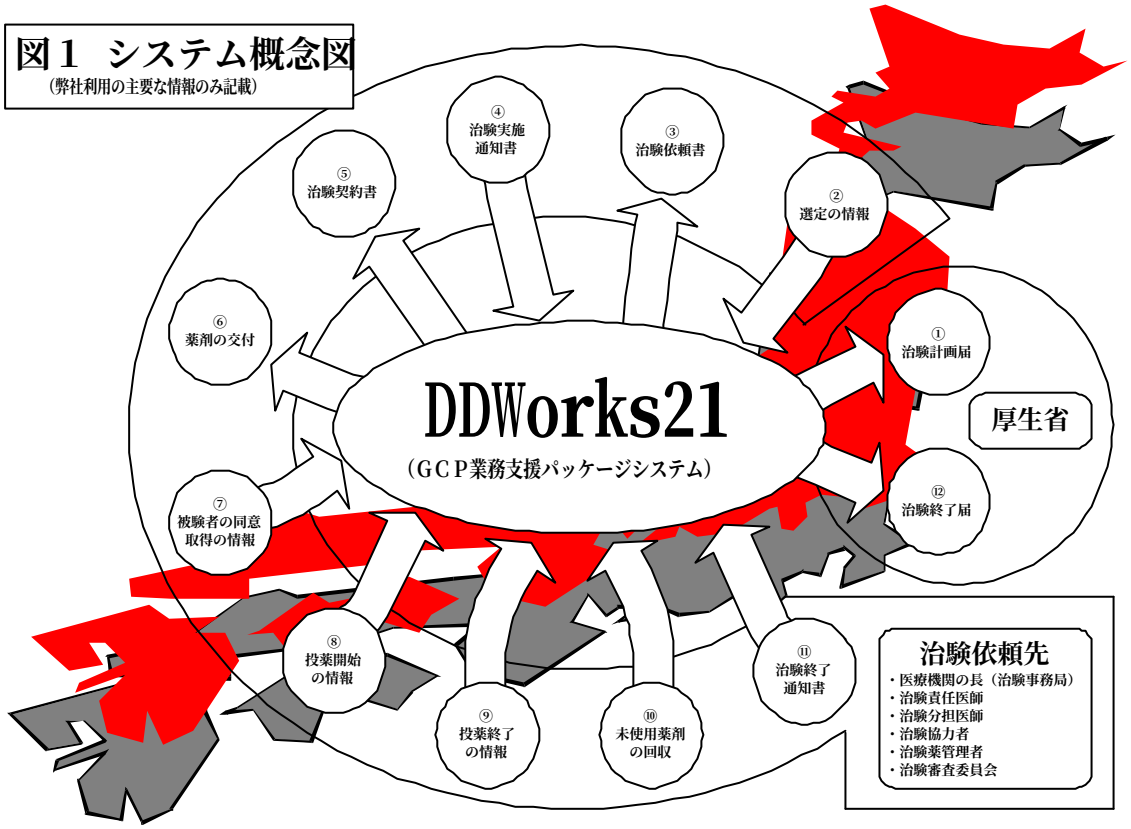
これらの情報を総合的にチェックして，ＧＣＰなどの法規制上，問題のない手順で臨床試験が実施されているか，それを証明するために必要な記録が適切な時期に作成（入手）されているか，契約書やその他の記録の内容に不備や抜けがないかなどの臨床試験の手続き面の品質管理，進捗管理を支援するのが主な機能となる．

3．パッケージシステム（DDWorks21）導入の経緯

Sigma21（中期計画）では，前述したような環境変化に対応するため，メインフレームで構築したすべてのシステムを西暦 2000 年までにダウンサイジングすることを基本方針とした．また，大規模なダウンサイジングを短期間で実現するために，パッケージシステムの導入を前提として，調査・検討を行った．

その結果，ＧＣＰ業務支援のパッケージシステムとして，最終的には DDWorks21（富士通製：図１参照）が選択された．DDWorks21 を選択した主な理由は，テーブル構造，ソースオブジェクトなども含め，システムのすべてをオープンにできるという柔軟な姿勢であった．弊社の場合，組織・業務に合せたシステムがいくつか構築されており，パッケージシステムを的確に導入するためには，ユーザ側も，システム構造の詳細を把握しておく必要があったためである．

図１　システム概念図
（弊社利用の主要な情報のみ記載）



4．パッケージシステムを導入する際の問題

新ＧＣＰを遵守するための作業手順が各担当者の考え方により異なっていたため，新ＧＣＰの施行直後は，業務の標準化が難しく，パッケージシステムの仕様も流動的であった．その後は，厚生省からの通知・連絡などにより調整が成され，業務手順の見直しによるシステム改訂が発生することになった．

必要に迫られてパッケージシステムの機能を改訂することになったが，パッケージシステムに対して変に機能追加すると，元の方がまだ良かったという悲惨な結果となった．システムの構造に合わせたシステム化要件を作成しないで，安易に機能追加すると，この問題に直面することになる．

ユーザは，状況とか方法を具体的に述べず，ただ目標を言うだけである．自分の要求がもたらす二次的な影響のことは考えないのである．ユーザ要件を鵜呑みにして適切に処理しないで，システム構造と乖離したままのシステム化要件を作成すると，機能追加は表面的なものにとどまり，張子の虎のような機能をかぶせることになる．こうなると，新規開発に比べて明らかに機能が重くなり，思うようなレスポンスが出ず，ユーザから「元に戻してくれ．」ということになる．

このように機能追加によりレスポンスが極端に悪化する局面を分析すると，システムの特性に応じたインタビューが実践できておらず，ユーザ要件からシステム化要件を導き出すプロセスも適切でなく，結果としてシステム構造に合わないシステム化要件が作成されていた．

５．ケーススタディ

前述したように，パッケージシステムの導入を成功に導くポイントは，無用なカスタマイズをしないインタビューの実践と，システム構造に合った的確なシステム化要件を作成することである．このようなインタビューの実践は，ＩＴだけでなく，業務にも精通し，必要に応じてＢＰＲが推進できるユーザ企業のシステム部門担当者の役割である．以下に，ユーザ企業のシステム部門担当者として経験した教訓，反省点を紹介する．

項５．１では，無用なカスタマイズをしないインタビューの実践について述べる．項５．２では，やむなく機能追加する際に，システムの特性に応じたインタビューを実践し，システム構造に合ったシステム化要件を作成するポイントについて述べる．

５．１　無用なカスタマイズをしないインタビューの実践

５．１．１　システムではなく，ユーザの意識をカスタマイズすること

２次バグの発生などを考慮すると，パッケージシステムの導入においては，無用なカスタマイズはしないこと，プログラマに無駄な仕事をさせないことが最良の方法といえる．それには，システムではなく，ユーザの意識をカスタマイズすることである．

ユーザの頭の中をカスタマイズする方法として，ＮＬＰ神経言語学的プログラミングの「意識内容のリフレーミング」という臨床心理学の考え方を応用した．「意識内容のリフレーミング」には，意味のリフレーミング（意味を変えること）と，状況のリフレーミング（状況を変えること）の大きく二種類の方法があり，この方法を駆使して，相手の考え方を良い方向へ誘導する．

パッケージシステム導入の場合，システム機能が先に存在する点が新規のシステム開発とは異なり，システム機能について，「意味」や「状況」を問われる局面が増えることになる．この共通点が，「意識内容のリフレーミング」の応用を可能とした．

極端な例では，パッケージシステムを目の前にして，ユーザから「こんな機能は使えない．」と言われる．このときの返答のポイントは，「意味付け」と「状況設定」である．システム機能の説明は求められていないのである．この点に留意して理論武装しておき，ユーザの思考に合せた言葉で返答することが肝要である．

ユーザが提示した問題が「意味」の問題であれば，ユーザが無意味と考えているシステム機能をユーザの思考に合せた表現で意味付けする．「状況」の問題であれば，将来発生すると想定される状況をきちんと想像させるプレゼンテーションが重要となる．

そして最後に，様々な「状況」を想定して「業務」パターンに漏れがないか確認する．このとき，ユーザの「業務」と「役割」に不一致がないか，きちんと確認しておくことも重要である．不一致を放置したシステム機能は，その後，何らかの問題を生ずることになるからである．

こうした検討の後，結果的にカスタマイズすることになっても，パッケージシステムには無意味なシステム機能が無いことを，ユーザにきちんと理解させることは重要なプロセスである．

中には「意味」の問題でも「状況」の問題でも無い場合がある．ユーザが本当は使いたくないケースである．ＳＥから提案された内容が，その人がするかも知れない領域内にある場合には，考えることさえも拒むものである．ユーザが使いたくないと思っている場合は，大半がこの状態で思考が止まっている．このときの返答をユーザ要件とするのは早計である．ところが，その領域をはるかに超えた場面をＳＥが設定し，相手に考えさせることができれば，後はじっくり待つことである．心の中の反発心から解き放たれ，理性的に物事を考え，ユーザ自身から結論を変えてくるのである．このような対処方法を，専門的にはフューチャー・ページングという．ＳＥならば，プロジェクトで発言した内容が誰からも賛同を得ず，後日，同じ意見をユーザが自分の意見として発言していたという経験が一度や二度はあると思う．相手が拒否しているときは，実際は考えることさえも拒否しているケースが大半である．ユーザにきちんと伝えるべきことを伝え，その場で考えてもらい，時間を空けて後日，回答を得るプロセスは非常に有効であった．

ここで誤解がないよう補足するが，フューチャー・ページングは，単に時間を空けることではなく，その前にきちんとした説明なり説得（専門的にはアンカリング）があって，無意識の領域も含めて，相手に繰り返し考えさせることがポイントである．

５．１．２　操作マニュアルが長文化する無用なカスタマイズを阻止すること

ユーザが必要以上にＩＦ文を増やすカスタマイズを要求するときがある．ユーザは「このときはこう処理して，そのときはそう処理した方が便利だ．」と言う．これも大半は無用なカスタマイズである．このケースに対処するには，操作マニュアルを頭の中に描かせるのが良い方法である．ユーザが望む機能を具体的な文章にすれば，その機能が良い機能か，無用な機能か一目瞭然となり，旨くユーザを説得することができる．

ＩＦ文を増やすことにより，マニュアルに記述する字数が減るのであればそれは良いカスタマイズで，逆に，説明文が長くなるようであればそれは無用なカスタマイズといえる．

５．１．３　業務アプリケーション向きの明示的な機能を採用すること

目の肥えたユーザからのパーソナルなシステム向きの機能追加を阻止することも重要である．例えば，右クリックは，その機能が存在することを知らないユーザには使えない機能である．システム機能が明示的でないからである．業務アプリケーションシステムでは，目で見て分かる明示的な機能以外は避けるべきものと考える．ユーザに対しては，「明示的でないシステム機能は，パーソナルなシステムとは異なり，業務アプリケーションには不向きだ．」とはっきりと言う必要がある．また，システム化の方法として選択肢が与えられた場合に，明示的な機能を選択すべきことはシステム開発のセオリーと考える．明示的でない機能はユーザに余計な混乱を招き，新たなカスタマイズ要件を発生させるからである．

５．２　システム構造に合ったシステム化要件を作成するインタビューの実践

５．２．１　ユーザが必要とする一覧は小さなビューとして構築すること

ユーザは，システムから生成されるビューを見て，意思決定をし，アクションを起こす．このビューの設計時に，クライアントサーバシステムに合わない仕組み作りがなされていた．例えば，ユーザが見たい情報が，大きな一覧表として大量印刷されるケースである．このとき，本当にそのように大きな一覧表が意思決定のために必要なのか再考する必要がある．クライアントサーバシステムでは，ネットワーク上を大量のデータが伝送されることになり，レスポンスの面で問題となるからである．

ユーザからは「すべてを確認できる一覧表を作成して欲しい．」と相談を受けるが，クライアントサーバシステムでは，最もシステム化に向かない要件と考える．実際，混雑時には他の機能にまで悪影響を与える帳票もある．レスポンスの問題を口にするとユーザは「システムとしての意味がない．」と言う．しかし，クライアントサーバシステムは大量の情報の中から，瞬時に必要な情報を抜き出すのが得意分野で，大量の情報を一覧にするのは不得手な分野だとユーザにきちんと認識させる必要がある．

ユーザ要件の確認段階から，この点を意識したヒアリングも必要である．ユーザ要件を一覧表ではなく，各々のユーザにフィットしたレスポンスの良い小さなビューとして，各画面の要所，要所に配置するようシステム化要件をまとめるべきであった．

DDWorks21 の中でも，見たい情報が日常利用している画面のすぐ側にコンパクトにまとめられている場合，ビューが小さいが故にレスポンスも良く，とても便利な機能として完成されている．

このような反省をふまえ，現在では，各々のユーザにフィットした小さなビューを弊社のシステム部門では**ＦＳＶ**（Fitting Small View）と命名し，ユーザには，ビューが小さいが故にレスポンスも良く，とても便利な機能として完成されることを約束している．レスポンスの良いシステムの実現（**ＦＳＶ**の実現）が，真のＧＣＰ業務支援システムの実現を意味すると考えている．（**図２**参照）

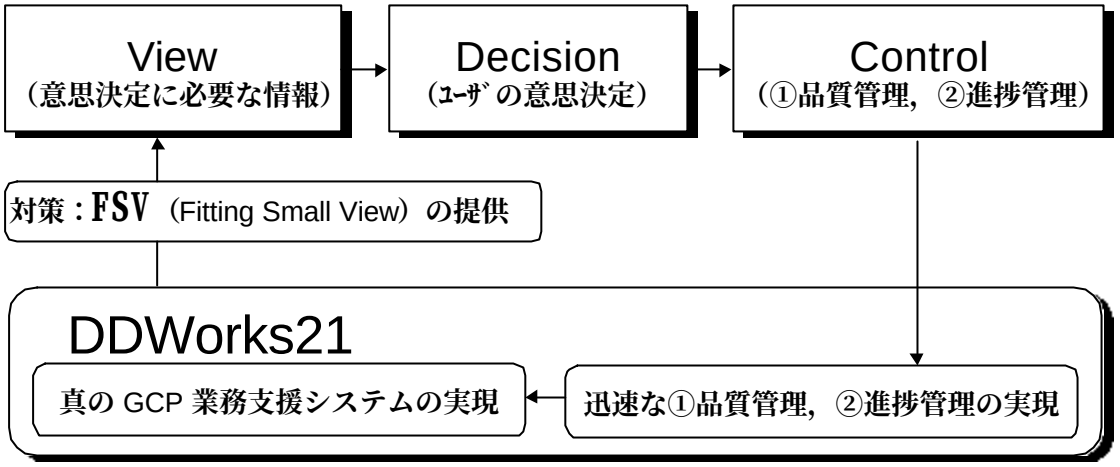


図２　ＦＳＶ（Fitting Small View）よる迅速な①品質管理，②進捗管理の実現

５．２．２　計算値をテーブル項目にする可能性を探ること

ＲＤＢの特性を考慮したレスポンスの良いシステム化要件の作成事例を紹介する．ＧＣＰ業務支援システムでは，治験届に関連するシステムチェックの必要性から，最新の届出回数，変更回数などを頻繁に算出する要件がある．この要件をそのままＳＱＬ文にすると，group by 句により最大回数を求めることになる．ところが，本当に group by 句が必要か考える必要がある．一般的には，group by 句は時間を要するために避けたい処理の一つである．DDWorks21 でも最初はこのようなＳＱＬがビューとして組み込まれていた．

代案としては，group by 句で求める計算値をテーブル項目として追加する方法である．group by 句で毎回計算する代わりに，該当の更新画面で計算値を算出しておく方が処理が軽く，システム全体のレスポンスも確実に保証される．

データベース正規化のルールには，計算値の取り扱いについてのルールは無いが，ＳＱＬが簡素化される点を考慮すれば，計算値も可能な限りテーブル設計時に組み込む考え方もある．但し，計算値を削除する際の処理ロジックがルール化（削除しないなどのルール）できなければ実現は不可能である．したがって，ＳＱＬの group by 句が必要になった場合，計算値をテーブル項目にして，削除ロジックがルール化できないかをユーザに確認する．これがレスポンスの良いシステム化要件を作成するためのインタビューと考える．

また，コーディングし易くするための共用テーブルのようなビューの利用も問題である．ビューにはインデックスも張れず，チューニングする術がなく，ビューについて画期的なレスポンス対策が実現するまでは，ＳＱＬ実行の中間段階で多用するのはレスポンスを悪化させる以外の何ものでもないと考える．このようなビューの作成は禁止すべきと考える．

５．２．３　外部（参照）キーの重要性を認識すること

レスポンスの良いシステム改訂を実現するためには，テーブル構造まで含めた対応が必要である．前述したようにレスポンスの問題はテーブル構造に起因する場合が多く，ジョインすることの多いテーブル間に，１項目で連結できる外部（参照）キーを追加するだけで，ＳＱＬが簡素化され，レスポンスが大幅に改善されることがある．（**図３**参照）

良いテーブル構造は，プログラマの仕事を楽にする．ＳＱＬが複雑になり過ぎて思うようなレスポンスが出ない場合には，外部キーの追加が必要である．

また，帳票追加などの要件により，こういった外部（参照）キーの追加が必要になるケースは，テーブル設計時にユーザが必要とするビューを見抜けなかったデータベース設計時のミスともいえる．データベース設計時に，ＥＲ図を見て，他に必要な外部（参照）キーがないか，充分に確認する必要がある．ユーザには，「この情報と，あの情報をくっつけた，こんなビューは必要ないか？」という具合に，ＥＲ図を頭に描きながら新たな外部（参照）キーを探索するインタビューも効果的と考える．

外部キーの重要性は，DDWorks21 のテーブル定義を見れば即座に理解できる．２００を超すテーブルの１つ１つに，同じ名前の外部キーがいくつも配置されている．例外はあるが，基本的には，各テーブルのユニークキー（１項目）が，そのまま関連のあるテーブルすべてに外部キーとして組み込まれている．このため，任意に存在する２個の関連テーブルは，Where 文にたった１個の AND 条件を記述することでジョインすることが可能である．

これは，各テーブルのユニークキーの値を，システムが自動生成（自動付番）する仕組みの上に成り立っている．ユニークキーの値が，業務とは独立・不変の値になっているため，すべての外部キーを一度に更新する処理は存在せず，レスポンスの心配も不要である．レスポンスの向上及びコーディングの省力化に，テーブル構造が大きく貢献している良い例だと考える．

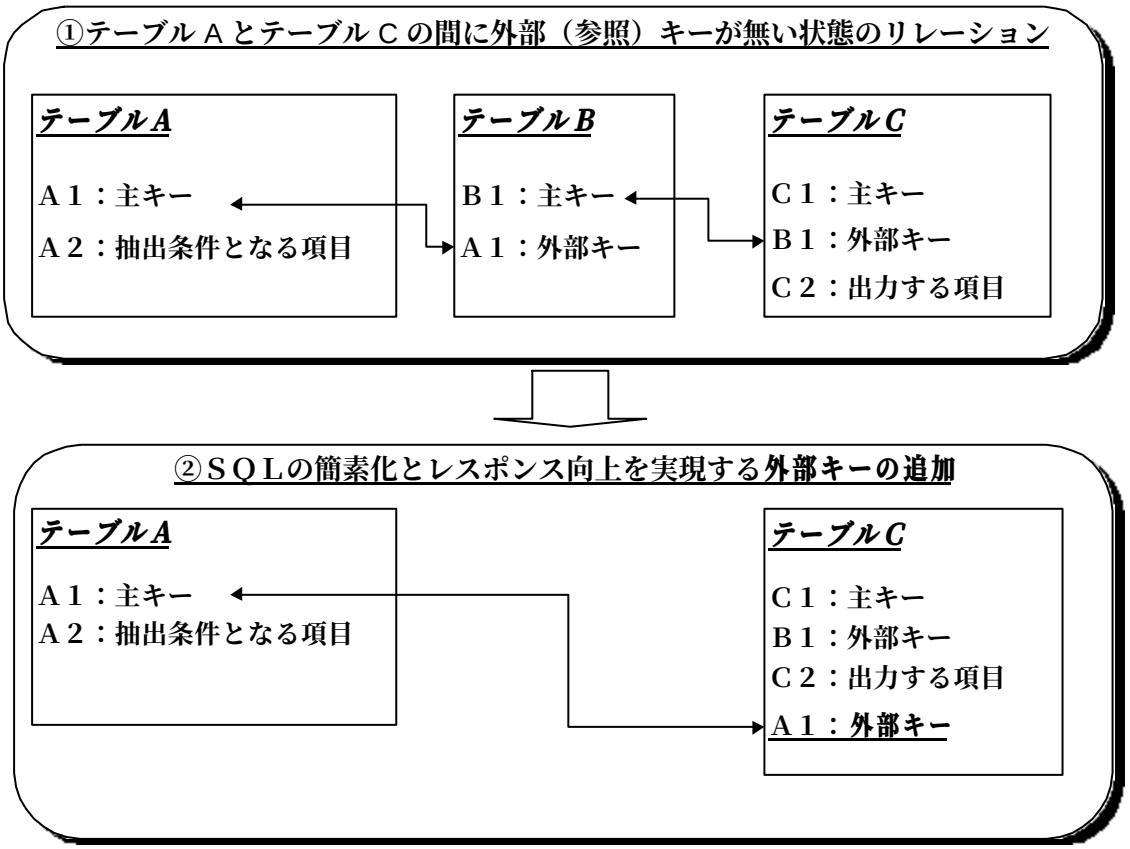


図３　ＳＱＬを簡素化する外部キーの追加

６．まとめ

パッケージシステム（DDWorks21：富士通製）の導入により，メインフレームのすべての機能（COBOL：60 万ステップ程度）を 1999 年 12 月までにダウンサイジングすることができた．パッケージシステムの導入の際には，テーブル構造やソースオブジェクトまでオープンにしていただけたことは貴重で，そのときのウォークスルーの経験から様々な教訓を得ることができた．

パッケージシステムの導入を成功に導くポイントは，無用なカスタマイズをしないことであり，やむなく機能追加する場合には，システム構造に合った的確なシステム化要件を作成することである．そのためには，システムの特性を良く理解して，的確なインタビューを実践することである．

従って，開発元のＳＥとユーザ企業のシステム部門担当者が旨く融合した良好な関係でのシステム作りが，より良いシステムの実現に結びつく一番の成功要因と考える．この経験を活かし，システムの特性に応じたインタビューの実践に努め，パッケージシステムをより良いものにしていきたいと考える．

参考文献

（１）R．バンドラー，Ｊ．グリンダー：ＮＬＰ神経言語学的プログラミング　リフレーミング　心理的枠組の変換をもたらすもの，星和書店，1988 年 4 月 8 日