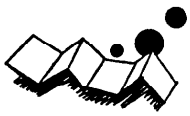
解　説



データベースの完全性と機密保護†

山崎晴明††　吉田　勇††

1. まえがき

近年，さまざまの企業で情報がデータベース化されてきている．これらの情報は発注，売上，在庫等の企業の基本的な日常活動に関するものばかりでなく，企業戦略に関する意志決定のサポートとする情報にまで及んでいる．このため，データベースの完全性維持と機密保護はますます重要となっている．本稿ではこれらの機能を実現するための方式について概説する[1]．

2. データベースの完全性

データベースシステムにおいて，データベースを本来あるべき完全な状態に保つことをデータベースの完全性 (integrity) という．完全性が破壊される要因としては，プログラムエラー，システムエラー，ハードウェア障害，オペレータの操作ミス等がある．データベース管理システム (DBMS) はこれらの障害が生起しても完全性を維持する機構を備えていなければならない．

2.1 データの正当性制御

データの正当性制御とは，たとえば年令が200才とか，給料が3万円以下といった，意味的に誤ったデータがデータベースに投入されるのを防止する制御である．これらの誤りをなくすには，システムに入力データの制約 (integrity constraints) を登録しておきシステムがチェックする方法がとられる．これらの制約には，以下のようなものがある[2]．

・ 給料が 30 万円以下である，というようなレコードに対する制約

・ 平均給料が 20 万円以下である，というようなレコードの集合に対する制約

・ 新しい給料が前の給料より多くなければならない，というような，更新前と更新後の値の関係による制約

・ 年令 40 才以上の従業員の給料は 20 万円以上である，というような条件付の制約

これらの制約をチェックするのは，一般的にはコマンド実行要求時であるが，トランザクションの終了時やトランザクション内のある時点の場合もある．コマンド実行要求時に制約違反が検出されたときには，その要求は拒否され，エラーコードが要求プログラムに返される．

上記のように，システムの行える入力データの意味的チェックには限界があり，本当に入力データが意味的に誤っているか否かの判断はアプリケーションプログラムやデータを入力する人間の責任に負うところが大きい．

2.2 同時実行制御

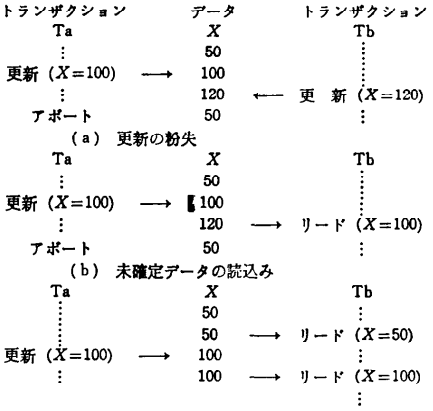
複数のユーザが同時にデータベースにアクセスするときには，個々のトランザクションとしては正しい処理を行っていたとしても，システム全体としては結果が一貫性を欠く (inconsistent) ことがある．ここでトランザクションとは，ユーザにとってのあるひとまとまりの処理単位であり，一般的にはひとつのトランザクション内で複数のアクセスコマンドが出される．トランザクション内で出された更新系コマンドの効果はトランザクションの終了時（たとえばコミットコマンドで明示される）に確定する．トランザクションは後に述べる回復処理の単位でもあり，何らかの原因でアボートすると，データベースの状態は当該トランザクション実行前の状態に復帰される．このようなトランザクションが同時実行され，同一データをアクセスすることにより一貫性が失われる可能性が生じるが，これには次の3つの形態がある (図-1 参照)．

(1) 更新の粉失 (lost update)

2つのトランザクション Ta, Tb が同じデータ X を更新するものとする．Ta, Tb の順で X を更新し，Ta がアボートすると，X は Ta の実行前の 50 に戻ってしまい，Tb の更新が粉失してしまう．

† Techniques for Database Integrity and Security by Haruaki YAMAZAKI and Isamu YOSHIDA (Systems Lab. OKI Electric Industry Company Ltd.).

†† 沖電気工業(株)総合システム研究所



(c) 再現性のない読込み

図-1 一貫性が失われる例

(2) 未確定データの読込み (dirty read)

Ta が X を更新し，Tb がその更新データを読み込んだ直後で Ta がアボートすると，Tb は実際には存在しないデータを読んでしまうことになる．

(3) 再現性のない読込み (unrepeatable read)

Tb が X を読み，次に Ta が X を更新した後再び Tb が X を読むと，ひとつのトランザクション内で読んだ同一データの値が異なってしまう．

J. N. Gray らは，上記3つの形態の一貫性のどこまでを保証するかにより，全く保証しない水準 0，(1)のみを保証する水準 1，(1)と(2)を保証する水準 2，(1)から(3)までのすべての一貫性を保証する水準3の4段階の一貫性の水準を定義している[3]．一貫性の水準を高めるとデータベースの同時使用度が低下し，スループットが落ちる．このため，差し支えない程度に一貫性の水準を低下させ同時使用度を高めることが一般的であり，アプリケーションに応じてユーザが水準を選択できることが望ましい．

2.2.1 ロック制御

一貫性を保証する最も普通の方法はロッキングである．ロックにはデータを更新するための排他ロックとデータを読むだけのための共用ロックの2種類設けるのが普通である．共用ロックを施したトランザクション間ではデータの同時アクセスが認められるが，排他ロックを施したトランザクションとの同時アクセスは認められない．ロック要求が認められないと，トランザクションは待たされることになる．したがって互いに他のトランザクションがロックしているデータに対して待ち状態にある場合にはデッドロックが発生する．

デッドロックの解決法には大別して2種類ある．ひとつはデッドロックを発生させないような方法を構じることであり，他のひとつはデッドロックが発生したときに検出して解除する方式である．防止方式は，ロック要求の出し方に制限を加えることにより実現される．たとえばそのトランザクションが使用するデータを全部一括してロックすればよい．しかしこの方式は処理を進めないと次にどのデータをアクセスしてよいかわからないような場合には厳しい制限である．

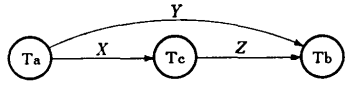
デッドロック発生の可能性が少ない場合には，検出方式の方がよい．文献 4)では効率よくデッドロックを検出するアルゴリズムを提案している．デッドロックが検出されたら，あるトランザクションをアボートすることにより解除する．どのトランザクションをアボートするのが最適であるかを見つける方法も開発されている[4]．

2.2.2 一貫性を保証するスケジュール

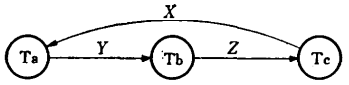
同時使用度を高めるためには，同一データに対する更新処理を含む複数のトランザクションでも並行して実行することが望まれる．しかし，このために前述したような一貫性が破壊されないようにするには，トランザクションの実行やロックのかけ方にある規則や制限を課す必要がある．

各トランザクションのすべてのリード，ライト操作を順序づけすることを“スケジュール”という．**図-2** に Ta, Tb, Tc の3つのトランザクションが与えられた場合のスケジュールの例を示す．ここではあるデータに対するリード，ライト操作は他の操作の割込

トランザクション:
Ta $X \leftarrow X+50$　$Y \leftarrow Y+50$
Tb $Y \leftarrow Y*10$　$Z \leftarrow Z*10$
Tc $Z \leftarrow Z*20$　$X \leftarrow X*20$
スケジュールと依存グラフ:
① Ta(X) Tb(Z) Tc(X) Ta(Y) Tb(Y) Tc(Z)



② Ta(Y) Tc(X) Tb(Z) Tb(Y) Ta(X) Tc(Z)



③ Tc(Z) Ta(Y) Tb(Y) Ta(X) Tc (X)Tb(Z)

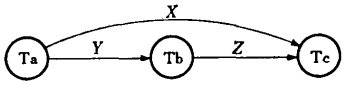


図-2 スケジュールと依存グラフの例

みを受けることなく，連続して行われると仮定して，たとえばTa(X) でデータXに対するトランザクションTa の処理を表わすものとする．

あるスケジュールが一貫性を保証するか否かを決めるのに“依存グラフ”(dependency graph) が使われる．グラフのノードはトランザクションを表わし，エッジはトランザクション間の依存関係を表わす．たとえばTa からTb へのエッジはTb がTa に依存する，すなわちTb のある操作がTa の操作の結果を読んでいる(この例ではTb が読むYの値は，Ta がライトした結果である) ことを表わす．グラフがサイクルを含まないならば，スケジュールは一貫性を保証している．たとえばスケジュール②はサイクルを含んでいる．この場合，データXについてはTc がTa に先行し，データYについてはTa がTb に先行し，データZについてはTb がTc に先行したことになり，結局全体的にみると3つのトランザクションの順序が定義できなくなってしまい，一貫性が保証されない．一方，スケジュール①あるいは③はサイクルを含んでいないのでトランザクションの順序が定義でき，この順序でシリアルに実行した結果と一致する．

システムは一貫性を保証するスケジュールとなるように，トランザクションの実行を制御しなければならない．このためにEswaran らは2フェーズロック方式を提案し，すべてのトランザクションがこの方式に従えば一貫性が保証されることを示した[5]．この方式は以下の2つからなる．

1) 各データはアクセスされる前にロックされなければならない．

2) 一旦ロック解除を行ったら，その後はロックをかけてはならない．

この方式に従うと，トランザクションはロックを順次かけていく成長フェーズ (growing phase) と，ロックが順次解除されていく縮退フェーズ (Shrinking phase) の2つのフェーズをもつことになる．

一般的なシステムでは，排他ロックはトランザクションの終了まで保持されるために，2フェーズロックの条件は満たすが，より強い条件となっている．この方法はトランザクション間の依存関係を防止することにより，障害発生時の回復処理を簡単にしている．というのは，あるトランザクションがアボートしたとき，そのトランザクションの更新結果を使っている他のトランザクションも次々にアボートさせなければならないという，いわゆるドミノ効果が発生しないからである．

2.2.3 ロック対象の大きさ

ロック対象の大きさ (granularity) をどの程度にするかは重要な問題である．ロック対象としては大きくはデータベース全体から，小さくはレコード内のフィールドまで様々の単位が想定される．一般にロック単位を小さくすれば同時使用度は向上するが，管理のためのオーバヘッドは多大となる．

ロック単位の選択はアプリケーションがデータベースをどのように使うかに大きく依存している．文献6)，7)ではシミュレーションの結果，最適なロック単位はトランザクションがアクセスするデータの量およびロック方法 (あらかじめロックをかけるのか，必要なときかけるのか) によっていろいろ変化することが示されている．このため，システムでロック単位の大きさを一意に規定してしまうのは好ましくなく，トランザクションの性質にあわせてロック単位を使いわけるのが有効となる．

Gray らは複数のロック単位を設け，それらを階層化する方式を提案している[8]．あるレベルの単位がロックされると，階層上のより下位のレベルのデータはすべて同じモードでロックされなければならない．ここで問題となるのは，あるデータを下位のロック単位でロックしたとき，その後にこのデータを含む，より上位のロック単位に対し，相反するモードでロック要求がある場合である．これは“意図ロック” (intention lock) を導入することにより解決している．すなわち，あるデータをロックしたい時には，まず上位のロック単位 (たとえばデータベース，エリア，ファイル) に対して，下位のロック単位(たとえばレコード)へのロックの意図を示す意図ロックを施さねばならない．意図ロックが施されたデータに対しては他のモードのロックを施すことはできないので，上記の問題は解決される．

Eswaran らはロックの対象データを述語(predicate)で指定する方法を提案している[5]．述語によるロックはファイル全体のロックより同時使用度が向上する．しかし反面，システムはどのレコードがロックされるべきかを決定しなければならない．また，別のトランザクションがレコードを追加したり，変更した結果，その述語を満たすようなレコードが新たに発生してしまう可能性がある (これらのレコードは“ファントム”と呼ばれる)．このため，システムはロック要求を比較して，互いに衝突を起こす可能性があるか否かを調

べる必要がある．述語として任意の形式を許すと，それらの衝突の可能性を調べることは困難であるから述語の形式は単純なものに制限する必要がある．現時点では述語によるロックをインプリメントしているシステムは存在しない．

2.3 障害回復処理

障害の種類はいろいろ考えられるが，その障害が与える範囲の程度に応じて，影響の少ない順から，トランザクション障害，システム障害，メディア障害の3つに大別できる．以下ではこれらの障害に対して障害回復に備えて通常行われる準備処理，および実際に障害が発生したときに行われる障害回復処理について述べる[9]．

（1） トランザクション障害

アプリケーションプログラムのエラーやデッドロックのために発生するひとつのトランザクションに閉じた障害である．この場合，トランザクション開始時点にデータベースの状態を戻さなければならない．

このための代表的な手法は更新前ログを使って後退復帰 (backward recovery) することである．更新前ログは更新系コマンドの対象となったデータの更新前の内容を保存したものである．トランザクション障害発生時には更新前ログの内容を順次時間的に逆方向に重ね書きしていくことにより，トランザクション開始前の状態にデータベースを戻すことができる．その後当該トランザクションを再実行することも可能である．

もうひとつの，システムR[10]等で用いられている手法に新旧バージョン方式がある．これはファイルをあるデータ単位（ここではページと呼ぶことにする）に分割し，あるトランザクションがあるページに対して更新処理を行うときには，ページ全体をコピーして新バージョンを作り，以後当該トランザクションの，当該ページに対するアクセスはすべて新バージョンを対象に行う方式である．更新前の内容は旧バージョンとして保持されるので，トランザクション障害発生時には，この新バージョンを廃棄して旧バージョンを当該ページの内容とすることにより，更新前ログによる回復手順なしに瞬時にトランザクション実行前の状態に復帰できる．また，トランザクションが正常に終了した時には旧バージョンを廃棄し，新バージョンを当該ページの内容として確定することになる．

（2） システム障害

システムプログラムのエラーや電源断等により発生する，DBMS 全体にわたる障害である．この場合，システムを再立上げし，直前のシステムチェックポイントの状態に復帰しなければならない．この障害に対する回復手法は，トランザクション障害に対するものと同じ手法が使える．たとえば更新前ログによる後退復帰の場合には，ある特定のトランザクションに対するものだけでなく，直前のシステムチェックポイント以後に実行されたすべてのトランザクションに対して後退復帰を行えばよい．

（3） メディア障害

データベースを格納している二次記憶媒体の障害である．メディア障害に備えて定期的にバックアップダンプをとっておく必要がある．バックアップダンプはある時点におけるデータベースの内容のコピーである．この場合，データベース全体のコピーをとる場合と，変更処理の行われたファイルのみのコピーをとる場合とがある．メディア障害からの回復のためには，ダンプの他に更新後ログが必要である．更新後ログは更新系コマンドによる更新処理のたびごとに，データの更新後の内容を記録したものである．

メディア障害発生時には，まずバックアップダンプを用いてデータベースの状態をダンプ取得時に戻し，その後更新後ログを使ってダンプ取得後の更新処理を再実行することにより，データベースを障害発生前の状態に復帰させることができる．

バックアップダンプは通常は定期的に取得される．この時間間隔を短くすれば障害発生時の更新後ログによる回復処理が比較的短時間で行われる．しかし，バックアップダンプの取得はかなり時間を要するので，この取得間隔および取得の方法はコストや，ユーザにとってのサービス性等を十分考慮して決定しなければならない．

3. データベースと機密保護

データベースに格納された貴重な情報が盗難にあうとか，悪用されるといった被害からシステムを守ることは，最近のコンピュータ犯罪の例を引くまでもなくきわめて重要な課題である．本節では，こうしたデータの不正使用からシステムを保護するためのセキュリティ維持についてその方法を概説する．

なお本節で述べられるセキュリティ維持手法は，認可により使用者のアクセス権を制限する方法，データの暗号化により情報の機密を保護する方法の2つである．

3.1 使用権の認可とチェック

安全なデータベース管理システムは，利用者によるデータのアクセス要求に対し，その要求を利用者の権限に照し合わせて正当であるか否かチェックし，権限を持たない利用者によるデータの不当な読み出しや書込みを防止するための保護機構を備えていなくてはならない．そのためには，まず第一にデータベース管理システム自身を安全な環境で実行する必要がある．こうした実行環境を設定するのは通常はオペレーティングシステムである．D. Downs ら[11]は安全なオペレーティングシステムが備えるべき最小機能として次をあげている．

（1） データベース管理システム自体の変更を禁止する機能

（2） 一次記憶中にあるデータの保護機能

（3） データベース管理システム以外のプログラムがデータベースをアクセスすることを禁止する機能

（4） 正しい入出力操作の実行を保証する機能

なおこの機能とアプリケーションプログラムとの関係を図-3 に示す．

一方，データベースのセキュリティを維持するため，データベース管理システムは通常図-4 に示すような利用者のアクセス権の認可チェックを行う．

一般にユーザがデータベースをアクセスするときの方法として，問い合せ言語のように直接端末装置からコマンドを入力する場合と，アプリケーションプログラムを起動させることによってアクセスを行う場合とがある．図-4 中①は問い合せ言語を使用した場合であり，データベース管理システムは要求元ユーザがデータベースへの正当なアクセス権を有するか否かをチェックする．一方②は2つの意味を持つ．ひとつはアプリケーションプログラム作成時にプログラマに対して行われる資格チェックであり，他のひとつは端末ユーザがアプリケーションプログラムを実行させる際に行われるユーザとプログラムの組合せに対する資格チェックである．③は端末ユーザがアプリケーションプログラムを実行させる権限を持つか否かの資格チェックとなる．なお，この他にユーザ ID やパスワードを用いてユーザの端末装置の使用権をチェックするシステムもある．

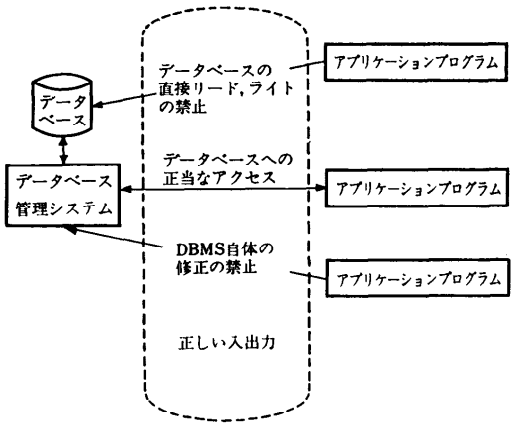


図-3 データベース管理システムの安全な実行環境

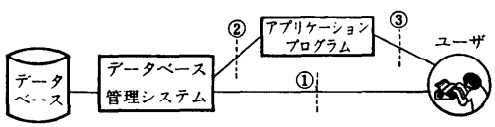


図-4 データベース管理システムのセキュリティチェック

データベース管理システムによる資格チェック機構の具体例として，システムR[1),12)]を考察してみる．システムRではデータベースのアクセス要求は SQL 言語で記述される．この言語は問い合せ言語であり，また PL/I や COBOL といったプログラム中に埋め込んで使用することもできる．

システムRでは，アプリケーションプログラムやリレーションの作成者，ビューの定義者は自身の作成したこれらのオブジェクトに関するすべてのアクセス権を持っている．さらに作成者はこの権利を他のユーザに認可 (grant) することもできる．この認可されたアクセス権はシステムカタログに登録される．たとえばリレーションの作成者がそのリレーションへのアクセスの権利を他のユーザと共有したいときには，次に示す Grant コマンドを出す[12)]．

Grant (privilege) on 〈リレーション〉
　to 〈ユーザ名〉 With grant option

ここで privilege には read, write, insert, delete, update, drop といったアクセス権の種別が指示される．また With grant option が指定されれば，アクセス権を認可されたものが，さらに他のユーザに対し，その権利を認可することも可能となる．認可されたアクセス権は後になって認可されたものによって取り消される (Revoke) こともあり得る．

たとえばユーザAがユーザXに対しあるリレーションの read, insert, update 権の認可を行い，次にユーザBが同じユーザXに対し read, update 権を認可したとする．その後AがXに対し insert, update 権を revoke したとすれば，XはAより認可された insert

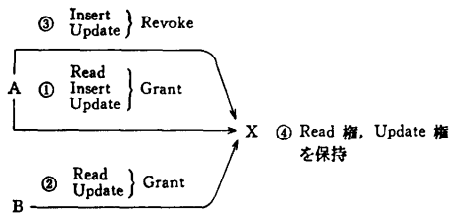


図-5 Grant, Revoke シーケンスの例

権は失うが，Aだけでなく Bからも認可されている read 権，update 権は継続して保持することになる(図-5)．

このようにして，Grant, Revoke コマンドは利用者のリレーションに対するアクセス権を動的に変化させることになり，問い合せに対するアクセス権チェックは実行時に行われることになる．

また，ビューリレーションに対しても同様のアクセス権が定義されるが，通常のリレーションと異なり，2つの制限がおかれる．ひとつはビューセマンティクスと呼ばれるもので，特定の操作は利用者のアクセス権とは無関係に禁止される（結合演算 (join) によって作られているビューリレーションに対する更新など)．他のひとつはビュー定義を行った者が持っているアクセス権に関するものであり，たとえば定義者があるリレーションに対して read 権のみ有するのであれば，そのリレーションの上に定義されたビューは read only となる．また，ビューがいくつかのリレーションにまたがるのであれば，それらのリレーション上のアクセス権の共通部分としてビューに対するアクセス権が定義される．

一方，SQL ステートメントを含んだアプリケーションプログラムはプレコンパイルされる．このアプリケーションプログラムをプレコンパイルする者をプログラマと呼ぶことにすれば，プログラマは当該プログラムに対する実行権を受け取る．この実行権はたとえプログラマがソースプログラム中の全 SQL ステートメントを実行する権利を持っていなくとも与えられる．また，プログラマは実行権を他のユーザに認可してもよい．この実行権のチェックはユーザが最初にアプリケーションプログラムを起動したとき行われる．一方，その後の内容に依存した制御は実行時に行われることになる．

3.2 暗号化／復号化手法

前節で述べたような利用者のアクセス権チェックは通常はパスワードやユーザ ID の組み合せによって行われる．したがって，こうしたパスワード自身が盗まれてしまってはアクセス権チェックは無意味となる．盗聴や類推が困難なパスワードの構成法には様々なものがあるが[13]，たとえばデータベースへのアクセスを行う通信回線上に自己の端末を接続し，必要な情報を盗み出したり，本来のシステムの代りに自己のコンピュータを接続させてユーザにはあたかも本物のコンピュータと通信しているかのように見せかけ，パスワードを含む一切の情報を盗み出す（ピギーバックと呼ばれる）といった不正行為から，システムを守ることは非常に困難と思われる．

また，ハードウェアコストの低下傾向にあわせて，マイクロコンピュータにフロッピディスクを付加したような簡易システム上で実現したデータベースも今後広汎に普及してゆくと思われるが，このようなシステムでは前述のオペレーティングシステムによるサポート環境も十分ではなく，また，フロッピディスクのような入手の容易な媒体にデータが格納される可能性もあり，このような場合，媒体自体が盗まれてしまうといった危険が大きくなる．このような形での侵害に対しては，暗号化を適用した機密保護機構が特に注目されるものとなる．データベース管理システムへの暗号化法の適用には，媒体に収容されたデータの暗号化，伝送回線上の暗号化，システムの管理するパスワード表等，セキュリティ上のキーとなるデータの暗号化等，様々な形態が考えられる[12]．

暗号化システムの一般的な適用形態は図-6 のように表わすことができる．ここで平文テキストとは暗号化される前のテキストを表わし，キーは暗号化／復号化を実行するために必要となる固有のビット列である．最近注目されている暗号化法として，暗号化，復号化に同一キーを使用する DES (Data Encryption Standard) 方式と，異なるキーを使用する公開暗号システムとがある．ここではその概要を述べる．

(1) DES の概要[14),15)]

DES は IBM が提案し，NBS (National Bureau

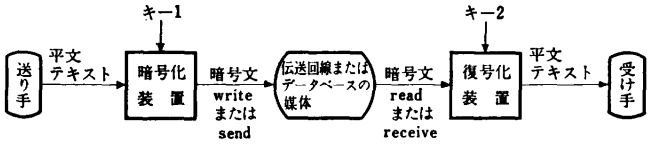


図-6 暗号化，復号化法のデータベースへの適用

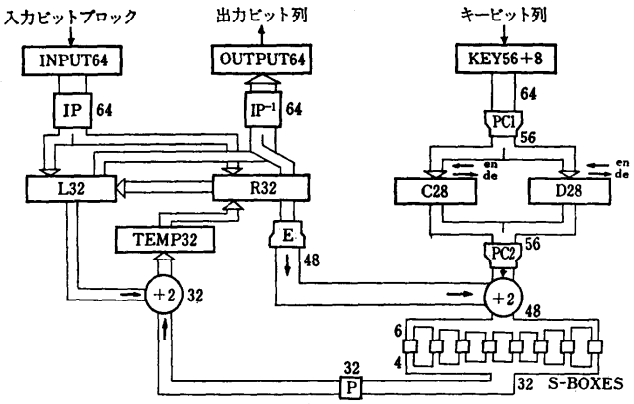


図-7 DES アルゴリズム概要

of Standards) によって，1977年米国の標準暗号規格として採用された．

この暗号化法は64ビットの入力ブロックと64ビットのキー（内8ビットはパリティ）とから64ビットの暗号化テキストを生成するもので，ほとんど同一のアルゴリズムにより復号化も可能としている点が大きな特徴である．図-7 に DES のブロックダイアグラムを示す[14]．このアルゴリズムはビット列の転置 (permutation；ブロック内のビットの配列を変える) および換字 (Substitution；記号またはビットを他の記号またはビットで置き換える) に基づき，次のように動作する．

① IP

64ビットの入力ブロックに対しては，まず初期転置 IP (Initial Permutation) が施される．IP 実行後は，32 ビットごとの2つのブロックに分割され各々 L32，R32 に入れられる．

② 内部ループ

次に L32 の内容は P から出力された32ビットと排他 OR 演算 (EOR) がとられ，TEMP 32 に入れられる．一方，R32 の内容は L32 にコピーされる．これは 16 回くり返される．

③ キーの扱い

64 ビットのキー列は，内8ビットのパリティを除き，実質 56 ビットとして与えられている．このビット列は，まず PC1 で転置されその結果は2つの 28 ビットブロックに分割されて，それぞれ C28，D28 に格納される．C28，D28 の内容は，暗号化時には1ビットもしくは2ビット左へ，復号化時には1もしくは2ビット右へ巡回シフトされる．これが暗号化と復号化におけるアルゴリズムの唯一の相異点である．

次にシフトされた内容は PC2 によって 48 ビットに縮少され，E から出力される 48 ビットと EOR がとられ，S-BOX への入力となる．ここで E は R32 の内容(32 ビット) を 48 ビットに拡張する．

S-BOXES は8個からなっており，その各々は6ビットの入力を4ビットに換字するため，S-BOXES の出力は合計 32 ビットから成り，P によって転置された後 L32 の内容と EOR がとられて，TEMP 32 への入力となる．

④ IP^{-1}

16 回のくり返し計算を経て最終的に L32，R32 に格納されたビット列は，IP の逆転置 IP^{-1} を行って 64 ビットの暗号文として出力される．このとき R32 と L32 の内容が左右入れかわっているため，復号化に全く同様のアルゴリズムを適用することが可能となる．

（2） 公開キー暗号方式の概要

DES の最大の弱点は，暗号化と復号化時に同一のキー値を用いるため，受信者は何らかの手段でそのキーの値を知らされていなくてはならず，このときキー値の漏洩が起きる危険があるということであった．これに対し，1967 年に W. Diffie と M. E. Hellman により提案された公開キー暗号系 (Public-key crypto system)[16] は，図-6 におけるキー1とキー2とに異なるキー値を持たせ，キー1の方を公開するものである．ここでこの2つのキー値はある特定のキー対生成関数によって作られる．このときキー2は公開されたキー1と生成関数とから推定可能であってはならない．メッセージは公開されたキー1（パブリックキー）により暗号化され，解読はキー2（シークレットキー）を用いて行われる（図-8）．

この方法は DES 等他の暗号法に比べ，キー値の極秘伝送を必要としない，メッセージの改ざん防止や認証に大きな威力を発揮するといった利点を持っている．

キー対生成法の例としては，有名な Rivest, Shamir Adleman 法 (RSA 法) と呼ばれるものがある[14]．これは，p, q を十分大きな素数としたとき，$r=p\cdot q$ および，$(p-1)\cdot(q-1)$ と素な整数 s をとり，(s, r) を公開キーとして公表するものである．

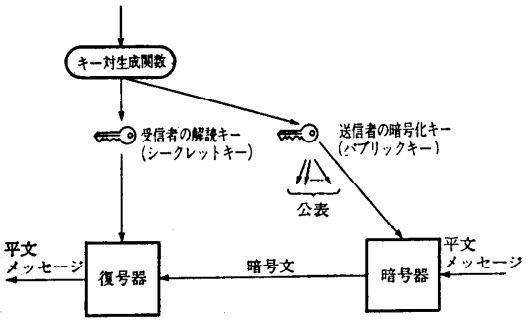


図-8 公開キー暗号方式

暗号文Cは，メッセージMを整数と考えたとき，

$$C \equiv M^s \bmod r$$

で与えられる．次に，Xを$(p-1)$，$(q-1)$の最小公倍数としたとき，$st \equiv 1 \bmod X$なるtを選ぶ（このようなtが存在することは，sが$(p-1)(q-1)$と素な整数であることから導かれる）．このtがシークレットキーとなる．復号化は：

$$D \equiv C^t \bmod r$$

により行われる．$D=M$となることの証明は次のようである．

$$D \equiv C^t \equiv (M^s)^t \bmod r.$$

ゆえに，$M^{st} \equiv M \bmod r$を示せばよい．

まず$M^{st}-M$がPの倍数であることを証明する．

$M^{st}-M=M(M^{st-1}-1)$であるからMがPの倍数であれば，証明は明らかである．一方MがPの倍数でないとき；

$st-1 \equiv 0 \bmod X$で，かつXは$(p-1)$の倍数だから，$st-1$は$(p-1)$の倍数となる．したがってFermatの定理*により$M^{st-1}-1 \equiv 0 \bmod p$となり，$M^{st-1}-1$が$p$の倍数であることが示される．

同様にして，$M(M^{st-1}-1)$はqの倍数ともなる．pとqは素数であるから，$M(M^{st-1}-1)$は$r=p \cdot q$の倍数となる．

ゆえに$M^{st}-M \equiv 0 \bmod r$.

たとえば，$s=3$，$r=55$，$p=5$，$q=11$としメッセージが$M=4$で与えられたとすれば，

$$C=4^3 \equiv 9 \bmod 55.$$

$(5-1)$，$(11-1)$の最小公倍数$X=20$だから，$st \equiv 1 \bmod 20$なるtは7である．したがって，復号化は$D=C^7=9^7 \equiv 4 \bmod 55$となる．

* Fermatの定理

pを素数，aをpの倍数ではない整数としたとき$a^{p-1} \equiv 1 \bmod p$が成り立つ．

この方法は，十分大きな素数p，qの積が与えられたとき，それから元の2つの素数を推定することは，非常に困難であるという事実に基づくものである．

4. 今後の展望

データベースシステムは，今後ますます利用形態が多様化し，またハードウェア，ソフトウェア技術の進展が大きな影響を与え続ける分野であると思われる．たとえばネットワークとデータベースとを結んだ分散データベースシステムにおいては完全性の維持やセキュリティの管理を分散して行う必要が生じる．また，データベースの知識化による知識ベースシステムが最近注目されているが，このようなシステムにおける完全性には従来の技術の範疇にはない全く新しい対応が必要となってくる．一方，オフィスオートメーションの進展に伴い新たにテキストやイメージデータを収容したデータベースも注目を浴びつつあるが，このような情報に対する機密保護の概念や機構もまた異なったものとなることが予想される．

さらに，ハードウェアコストの低下に伴い，データベースマシンというアプローチも盛んに行われているが，完全性やセキュリティ維持の技法をどのように効率よくハードウェア化するかという研究が重要である．このようなハードウェア環境でのセキュリティ維持は，たとえばHsiaoのセキュリティフィルタの構想[17]にみられるように，従来のソフトウェアベースのセキュリティ維持手法よりも，一層安全なシステムの実行環境を与えてくれるものとして期待される．

参考文献

1) Fernandez, E. B., et al: Database Security and Integrity, Addison-Wesley (1981).
2) Date, C. J.: An Introduction to Database Systems (Second Ed.), Addison-Wesley (1977).
3) Gray, J. N., et al.: Granularity of Locks and Degrees of Consistency in a Shared Data Base, Proc. IFIP TC-2 Working Conference on Modelling in Data Base Management Systems, pp. 365-394 (1976).
4) Gray, J. N.: Notes on Data Base Operating Systems, Operating Systems- An Advanced Course, Springer-Verlag, pp. 393-481 (1978).
5) Eswaran. K. P., et al.: The Notions of Consistency and Predicate Locks in a Database System, Comm. ACM. Vol. 19, No. 11, pp. 624-633 (1976).
6) Ries. D. R., et al.: Effects of Locking Granularity in a Database Management System,

ACM TODS Vol. 2, No. 3, pp. 233-246 (1977).
7) Ries, D. R., et al.: Locking Granularity Revisited, ACM TODS. Vol. 4, No. 2, pp. 210-227 (1979).
8) Gray. J. N., et al.: Granularity of Locks in a Shared Data Base, Proc. 1st Int'l Conf. on VLDB, pp. 428-451 (1975).
9) Verhofstad. J. S. M.: Recovery Techniques for Database Systems, ACM Computing Surveys, Vol. 10, No. 2, pp. 167-195 (1978).
10) Gray. J. N., et al.: The Recovery Manager of the System R Data Base Manager, ACM Computing Surveys, Vol. 13, No. 4, pp. 223-242 (1981).
11) Downs, D. and Popek, G. J.: A Kernel Design for Secure Database Management System, Proc. 3rd Int'l Conf. on VLDB, pp. 507-514 (1977).
12) Griffiths, P., et al.: An Authorization Mechanism for a Relational Database System, ACM TODS, Vol. 1, No. 3, pp. 242-255 (1976).
13) 小林侔史：OA 革命の落し穴—増加するコンピュータ犯罪，日経コンピュータ，No. 15, pp. 144-150 (1982).
15) Davies, D. W. and Bell, D. A.: The Protection of data by Cryptography, National physical Laboratory, NPL report com. 98 (1978).
15) 土居範久：米国の暗号化規格 DES, BIT Vol. 13, No. 2 (1981).
16) Diffie, W. and Hellman, M. E: New Directions in Cryptography, IEEE trans. on Informations Theory, IT 22 (1976).
17) Banerjee, J., et al.: Concepts and Capabilities of a Database Computer, ACM TODS, Vol. 3, No. 4, pp. 347-384 (1978).

(昭和 57 年 6 月 25 日受付)