

RDBサーバ i RDBにおける多目的なSQL最適化方式

6X-1

広島清美† 牛嶋一智‡ 森永智之‡ 山平耕作†

† (株) 日立製作所ソフトウェア事業部

‡ (株) 日立製作所中央研究所

1. はじめに

世の中の情報化が進み、情報産業の発展と共に、DBMSは、OLTP、OLAP など様々な目的で利用される。これらの場面では、DBMSには大量データから必要なデータを絞り込むような問合せ、及び大量データの結合、集約を伴い、アドホックでかつ複雑な問合せに対し、高速に処理することが必要である。そのため、SQLの最適化処理には、DBアクセスプランの探索において、極小解に陥ることなく、ミリ秒オーダーの短時間で最適解を見つけることが求められる。そのためのひとつの方法として、データの絞り込み率を優先、及びインデックスを優先することで、短時間に最適解に近い解を探索可能とし、RDBサーバ「HiRDB」に実装した多目的なSQL最適化方式について報告する。

2. SQL最適化処理に求められる課題

2.1 局所的なコスト比較による、コスト最小解の見逃し

HiRDBでは、ユーザによるSQLチューニングの負担を最小限にするため、SQLの処理時間見積もりを、DBアクセスプラン（以下、プラン）探索の目的関数とする、コストベースのSQL最適化方式[1]を採用している。コストベースのSQL最適化にて、プランの最適解を求める場合、スキャン方式、ジョイン順序、及びジョイン方式の組合せを、全探索すれば求まるが、最適解に至るまでの組合せ数が爆発し、現実的な時間で解を求めることはできない。そこで、あまり有効でないと思われるプランの探索を打ち切ること（枝刈り）が必要となる[2]。しかし、プランの枝刈りを行う場合、成長途中のプランの局所的なコストを比較して、局所的なコストの大きいものを枝刈りすると、極小解に陥ることが多く、OLAP等で用いられるスター型スキーマでの、複雑でアドホックな問合せに対し、有用なプランが得られないことが多い。

$$\begin{array}{ccccccc} \sigma_{T2}^{1.0} & - & \sigma_{T1}^{1.0} & - & \sigma_{T4}^{1.0} & - & \sigma_{T5}^{1.0} \\ (10) & & (1000) & & (50) & & (5) \\ & & | & & & & \\ & & T3 & & & & \\ & & (20) & & & & \end{array} \quad \begin{array}{l} (): \text{ScanCost} \\ \sigma = 0.1 \quad \sigma : \text{選択率} \end{array}$$

図 2.1 コストによるプラン枝刈りを失敗する例

図 2.1 に示す T1~T5 の 5 表のジョインでは、最初の 2 表のジョインを行うまでのコストは、T4・T5、T1・T2、T1・T3、T1・T4 の順に大きくなる。インデックスが定義されていないとすると、図 2.2 のジョイン順序でジョインすれば、T1・T3 のジョインを最初に行い、早期にデータを絞り込み、結果的にコスト最小のプランが得られる。にもかかわらず、T3 の Scan コスト

が T2 の Scan コストよりも大きいため、T1・T3 を最初にジョインするプランは枝刈りされ、図 2.2 のプランは得られない。枝刈りの結果、図 2.3 の極小解のプランに陥る。そこで、ある程度先を予想し、アクセスコスト最小以外の観点での、枝刈りを行うことで、有用なプランを残すことが必要である。

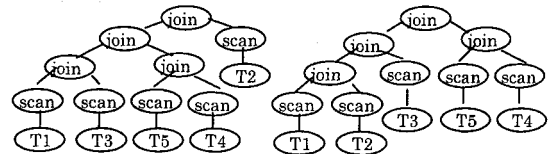


図 2.2 コスト最小プラン

図 2.3 コスト極小プラン

2.2 コスト計算誤差による、ユーザ意図の見逃し

コストベースの最適化では、複数キーでの絞り込みによるヒット率、変数を含む述語のヒット率、複数キーによる結合の結合率、他のキーで絞り込まれた場合に別のキーのキー値数、及びキーの分布を予想することは、現実的なシステムでは困難であり、コスト計算に誤差を生じる。このような事態では、ユーザの意図しないプランを SQL の最適化は、作成してしまう。そこで、ユーザの意図を汲み取ることが必要になる。

3. 多目的なSQL最適化

3.1 枝刈りせずに優先するプラン

(1) 絞り込み優先プラン

OLAP 等の分析業務では、大量データの多表ジョインを行うことが多い。その場合、局所的なコストでプランの枝刈りを行うと、実表スキャンコストが、ジョインコストよりも大きくなることがあり、表母体の大きさに、コストが左右され、それによりジョイン順序が決まってしまう。大表と複数の小表をジョインするようなスタースキーマ型の場合、ハッシュジョインを用いてジョインすることが一般的である。ハッシュジョインでは、最初の 2 表のジョインで、大表をジョインする必要があり、このときは最も絞り込み率の高い小表とジョインするのが、次段以降のジョインコストを小さくすることができる。しかし、最初の 2 表のジョインをコストにのみ頼って、決定すると、絞り込み率の高い小表よりも、より行数の少ない小表とジョインするほうが、コストが小さくなり、そちらを優先して、絞り込み率の高いジョインを枝刈りしてしまう。その結果、2 段目以降のジョインで、データの実合回数が増え、性能が悪くなる。

そこで、絞り込み率の高いジョインは、プランの成長とともに、コストが逆転する可能性が高いので、多少コストが高くても、枝刈りせずに残す必要がある。

(2) インデックス優先プラン

OLTP 業務など、ターンアラウンドタイムを小さくすることが求められる業務では、作業表およびハッシュ表を作成するプランは、使用しない方が多いことが多い。よって、イン

デクスを使用したネストループジョインを優先した方がよい。また、インデクス定義は、ユーザの明確な意図であるため、それを利用することは、ユーザの意図を汲み取るための有効な手段である。よって、インデクスの利用を優先したプランを作成する必要がある。

3.2 HiRDBのSQL最適化処理

(1) プラン枝刈りのための3種類のキュー

HiRDBでは、コストによる枝刈り、絞込み率による枝刈り、インデクスの有効利用による枝刈りの3種類の枝刈りのために、3個のキューを準備し、それぞれのキューで、各枝刈りを行ったプランを保持する。

一つ目のキューは、コスト最小キューである。これは、コストのみによる枝刈りを行い、一定数のプランを残し枝刈りするためのキューである。

二つ目は、絞込み率を優先するキューである。単純に表の絞り込み率だけを見て、これの大きい順にプランを優先すると、小表同士のジョインの無い完全スタースキーマ型のジョインでは問題が無いが、小表同士のジョインのある、変形スタースキーマ型では、最初に小表同士をジョインし、次に、大表と最も絞り込み率の高い小表、または、小表同士のジョインから導出された小表とのジョインが優先されなくなる。そのため、実際には絞込み率ではなく、ジョイン結果導出される中間表の行数が少ないプランを優先する。よって、このキューには、中間行数の多いプランを枝刈りし、中間行数の少ない一定個のプランを残す。

三つ目は、インデクス優先キューである。これは、インデクスを優先して使用するプランを残すキューである。前述したとおり、ジョインSQL最適化においては、インデクス優先ということは、ネストループジョインを優先して残すということである。このキューでは、ネストループジョインを連続して実行できる数を枝刈りの目的関数としている。

(2) HiRDBのプラン生成手順

HiRDBでは、投入されたSQLに対し、各表、問合せ指定をノードとし、2表以上間の探索条件をノード間のエッジとする、グラフを作成する。これを最適化グラフと呼んでいる。最適化グラフのエッジを一つ一つ解決していき、徐々にアクセスプランを作成していく。作成するアクセスプランを、実行木と呼んでいる。RDBMSのアクセスプランは、2項間の演算が規定できれば、十分であるため、実行木は、2分木の構造で表現する。部分的に変換した実行木を、一つのノードに置き換え、全エッジが解決されるまで繰り返す。

最適化グラフから、実行木に変換するときに、コスト最小、インデクス優先、中間行数最小の、三つの観点で枝刈りを行い、それぞれの観面向けのキューに実行木を残すことで多目的の最適化を実現している。三つのキューで、それぞれのプランを成長させていき、最終プランが得られた時点で、最終的にはコストが最小のプランを実行する。もっとも、コストベースの最適化では、複数キーの複合したキー値数、ヒット

率、結合比などの精度には、現実的なシステムでは限界があるので、HiRDBには、最終的にはコストで負けたプランを実行するオプションも残してある。

図3.1に、最適化グラフから実行木への変換過程を示す。

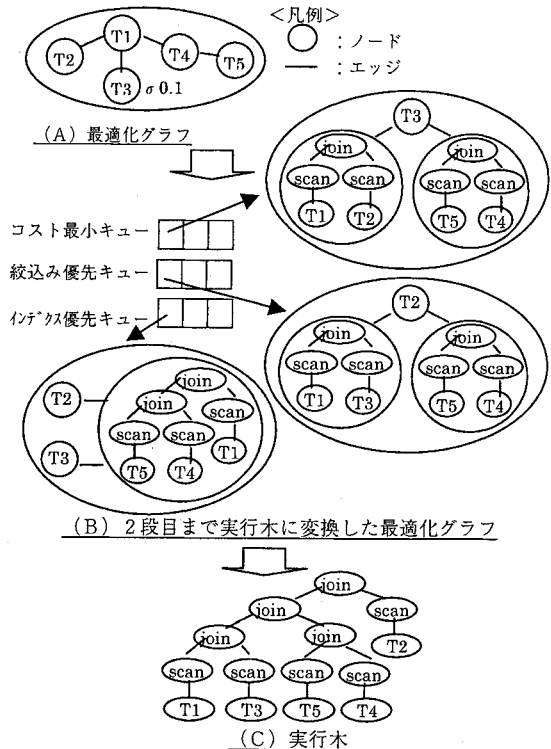


図 3.1 最適化グラフから実行木への変換

4. 効果

図2.1のスキーマにおいて、各表の行数を、 $T1=9,000,000$ 、 $T2=100$ 、 $T3=1000$ 、 $T4=30,000$ 、 $T5=100$ 、 $T3$ の選択率 0.2 の場合、HiRDBでは、絞込み優先の枝刈りを行うことにより、最適解(図2.2のプラン)を得ることができ、コストによる枝刈りのみを行った時の極小解(図2.3のプラン)に比べ、約1.5倍の性能を得ることができた。

5. おわりに

本報告では、OLTP、OLAPをはじめ、多様な使用目的に適したRDBMSとして、ユーザの定義したインデクスを有効に活用しつつ、スタースキーマ型、及びその変形型の複雑な多表ジョインのSQLに対して、有効なSQL最適化方式を示した。

6. 参考文献

- [1] P. Griffiths Selinger, et al: "Access Path Selection in a Relational Database Management System", In Proc. Inter. Conf. Management of Data(ACM)(Boston, Mass., May 1979), 23-34.
- [2] Surajit Chaudhuri, "An Overview of Query Optimization in Relational Systems", In PODS98, 1998.