インデックス関連

カバリングインデックスの利用

●インデックスを張っているカラムの値だけを取得する場合、インデックファイルに列値もあるため、わざわざ、テーブルファイルにアクセスする必要はない。ただし、インデックスファイルには存在するが、テーブルファイルの当該タプルが更新又は削除等により不可視であるならば、インデックスファイルにある列値を返してはならない。

●そこで、I/O 負荷が小さい Visibility Map（VM）にアクセスすることで、ページ単位ではあるが（テーブルファイルへのアクセスもページ単位なので問題ない）、完全可視（1）か不可視存在（0）かを判別できるため、わざわざテーブルファイルにアクセスすることを回避することができる。これが、**インデックスオンリースキャン**である。ここで、可視とは、不要なレコードが 1 つも存在しない、かつ、どのトランザクションからも更新されていない状態である。同様に VM において、ページ単位で完全 FREEZE 処理済み（1）か未 FREEZE 処理（0）かを判別することもできる。

●VACUUM 等で VM は更新されるが、VACUUM も VM にアクセスして VACUUM 対象ページを探索しているため、持ちつ持たれつである。

●インデックスを張っているカラムではないカラムの列値を取得する場合には、結局、テーブルファイルへのアクセスが必要となる。

●ここでようやく本題であるが、インデックスを張っているカラムではないカラムの列値が頻繁に必要であるならば、当該カラムを CREATE INDEX の INCLUDE 句に指定することで、インデックスファイルにそのカラムの列値も一緒に登録されるため、**インデックスオンリースキャン**を利用できる。これが**カバリングインデックス**である。

テーブルデータのクラスタ化

●「CLUSTER indexname ON tablename」とあるように、インデックスを定義していないとクラスタ処理はできない。2 回目以降のクラスタは「CLUSTER tablename」だけで OK。

●CLUSTER コマンド実行時には、REINDEX コマンドも同時に実行される。どちらのコマンドもテーブルロックは ACCESS EXCLUSIVE なので、他からの競合は一切受けない。

●テーブルは空き領域を再利用できるが、インデックスはソート順に格納しておく必要があるため、空き領域があっても再利用されない。ただし、完全に空になった B-tree インデックスは再利用できるが、何かしら残っている可能性あるため、REINDEX したほうがいい。

●REINDEX コマンド実行時には、CLUSTER コマンドは同時に実行されないが、元のテーブルには、書き込みロックが掛かる。ただし読み込みは OK である。しかし、インデックス自体が ACCESS EXCLUSIVE のテーブルロックが掛かっているため、読み込みもブロックされる可能性がある。CREATE INDEX も REINDEX と挙動は同じ。

●VACUUM FULL 実行時も ACCESS EXCLUSIVE である。この時、インデックスも再定義される。

●VACUUM と ANALYZE のテーブルロックは、SHARE UPDATE EXCLUSIVE である。元のテーブルに対する DML の実行は可能である。なぜなら。SELECT 時のテーブルロックである ACCESS SHARE 及び INSERT、DELETE、UPDATE 時のテーブルロックである ROW EXCLUSIVE と競合しないためである。

REINDEX コマンド

●REINDEX INDEX で指定したインデックス

●REINDEX TABLE で指定したテーブルの全インデックス

●REINDEX DATABASE で DB 全てのインデックス。global 内の共有システムカタログのインデックスも含む。

●REINDEX SYSTEM で DB 全てシステムカタログ（主にインデックス破損時）。global 内の共有システムカタログのインデックスも含む。

開発者向けオプション

●開発者向け（スーパーユーザ等）のオプション。実運用での使用は想定外なので postgresql.conf からは除外。

①ignore_system_indexes (デフォルトは off)

システムテーブル（システムカタログ）の読み込み時に使用するシステムインデックスがあるが、当該インデックスに障害がある場合、当該インデックスを無視する（使用しない）ために設定（on）するパラメータ。その後、REINDEX SYSTEM コマンドでインデックスを復旧する流れとなる。

②zero_damaged_pages (デフォルトは off)

●ページヘッダに障害があるとエラーとなりトランザクションは中断。しかし zero_damaged_pages を on にすることで警告扱いとなり、障害のあるメモリ内のページをゼロで埋め処理を継続。

●上記の措置により障害のあったページ上にある全ての行の**データが破壊**。しかし、これによりエラーがなくなり正常なページに存在するテーブル内の行を取り出すことができる。

●ゼロ埋め処理されたページはディスクに書き込みを強要されないため、off にする前にテーブルを再作成することができる。

その他

●テーブル（デフォルトは 100%）やインデックス（デフォルトは 90%）の FILLFACTOR は、挿入だけなら 100%で問題ないが、更新（削除含む）が多ければ、空き領域を確保することで、更新前後の行を同じページ内に格納できる可能性が大きくなる。また、インデックスも同様に空き領域があれば、ページ分割の頻度も小さくなる。

●大量のデータを挿入する時は、INSERT 文よりも、COPY コマンドの方が断然早い。その後、INDEX を付ける。

●SELECT 文で ORDER BY 句を使用する場合、ORDER BY 句で指定されているカラムの順番で INDEX が定義されていれば、INDEX 順で処理するため、ソート処理は不要となる。LIMIT がある場合には、INDEX がなければ、ソートしてから、LIMIT を適用するが、INDEX があれば、LIMIT だけ抽出すれば済む。