演習6-1　ビットマップインデックスとハッシュインデックス

ビットマップインデックス

　検索に用いられるカラムに対して、その値とレコードとのビットマップを使って検索を行う。

＊長所

・B-treeインデックスがカーディナリティが高い場合に有効だったのに対して、ビットマップインデックスは低い場合に有効。

・WHERE句にＯＲを用いた場合にもインデックスが利用されて検索を高速で実行できる。（AND検索もできるがB-treeのほうが有効）

＊短所

・レコードの追加，削除，更新に時間がかかるため、更新の多いOLTP（オンライン・トランザクション処理）系のシステムには向かない。

ハッシュインデックス

＊長所

・ハッシュ関数を用いて、検索に使用するキーとレコードを直接結び付けているため、データ数に関わらず、目的の要素がある場所を一度の計算で導くことができる。(B-treeはノードを辿っていくので直接、とはいかない)

・なので、等値検索する場合は非常に高速で検索できる。

＊短所

・等値検索以外の場合にはインデックスが利用できない。（使いどころが限定的）

演習6-2 インデックスの再編

SQL Server

・インデックスごとに再構築する

ALTER INDEX｛インデックス名｝ ON｛テーブル名｝ REBUILD;

・table1のインデックスをまとめて再構築する

ALTER INDEX ALL ON dbo.table1 REBUILD;

Oracle Database

・インデックスの再作成

alter index 〈ユーザー名〉.〈インデックス名〉 rebuild;

・インデックスの格納先表領域を変更する

alter index 〈ユーザー名〉.〈インデックス名〉 rebuild tablespace 〈変更先の表領域名〉;

MySQL

・テーブルを最適化する

OPTIMIZE TABLE {テーブル名};

~~ほかのDBMSのようなインデックスの再構築をする構文が見つけられませんでした…~~

~~※代わりに~~

~~・一度インデックスを削除して~~

~~ALTER TABLE [テーブル名] DROP INDEX [インデックス名];~~

~~・改めてインデックス作る~~

~~ALTER TABLE [テーブル名] ADD INDEX [インデックス名]([カラム名]);~~