Adaptive Server Anywhere 9 の重複インデックスの対処方法

このマニュアルでは、重複インデックス警告メッセージの履歴、重複インデックスが発生する理由、および重複インデックスの識別方法と削除方法について説明します。

SQL Anywhere 5 と Adaptive Server Anywhere 6 で作成されたデータベースには、各プライマリ・キーまたは一意性制約と参照外部キーに対して作成された結合インデックスがあります。多くの外部キーが同じプライマリ・キーまたは一意性制約を参照している場合、そのような結合インデックスは非常に大きくなる可能性があります。したがって、クエリ内の部分インデックス・スキャンに使用した場合、そのようなインデックスはパフォーマンスに影響を及ぼすことがあります。これは、プライマリ・キーと同じカラム、および外部キーと同じカラムに他の単純な（つまり、結合されていない）インデックスを作成することが推奨された理由でした。プライマリ・キー・カラムまたは外部キー・カラムに検索引数可能述部があるクエリで使用した場合は、単純なインデックスの方がはるかに適切に機能します。

Adaptive Server 7 以降のバージョンで作成されたデータベースは、データベース・スキーマで宣言されたプライマリ・キー、一意性制約、外部キーごとに独立したインデックスを持っているため、Adaptive Server Anywhere 7 で導入された変更によって、そのような重複インデックスを作成する必要性はなくなりました。これらのデータベースについては、結合インデックスは作成されません。何らかの理由により、データベースに重複インデックス（類似したカラム・シーケンス・プロパティを使用して同じカラムで作成されたインデックス）が含まれている場合は、更新操作とクエリ処理の両方のパフォーマンスが低下する可能性があります。パフォーマンスに対するフォーカスの一部として、Adaptive Server Anywhere 8 では新しいパフォーマンス警告メッセージ（重複インデックスの存在を識別したメッセージなど）がいくつか導入されました。ベース・テーブルの場合、インデックスがロードされるか、インデックスが追加または削除されると、インデックス・リストが解析され、重複インデックスにフラグが付けられます。サンプルの警告メッセージを次に示します。

Note: Duplicate index "idx1" for table "t" in database "db". (注意：データベース "db" のテーブル "t" に対する重複インデックス "idx1")

パフォーマンスへの影響

重複インデックスが存在すると、以下の理由によりパフォーマンスが低下することがあります。

- インデックス・カラムが変更されるたびに、Adaptive Server Anywhere はテーブルの値を変更し、インデックスも変更する必要があります。重複インデックスが存在する場合は、データが変更されるたびに各インデックスを個別に管理する必要があります。
- インデックス管理の量は、インデックスに影響する INSERT、UPDATE、DELETE 文の場合に増加します。

- オプティマイザの検索空間は、使用に適したインデックスを探すときに増加するため、一部の文については、最適なアクセス・プランを見つけることができない場合があります。* *

* *注意：Engineering Case #357981 の結果として行われた変更であるため、重複インデックスは、インデックス・ヒントで指定されない限り、最適化時に考慮されなくなっています。詳細については、以下の手順で参照してください。

1. サポート契約者様専用ページ（http://www9.ianywhere.jp/sail/index.html） にアクセスします。（ログインする際にユーザー名とパスワードが必要となりますが、SAIL 事務局より配信されております SAIL メールに記載されておりますので、そちらをご覧下さい。）

2. メニューの［修正情報検索］をクリックします。
3. 検索キーワードに 357981 と入力します。
4. ［Search］をクリックします。

一般的なインデックス重複ケース

インデックス重複の最も一般的なケースのリストを以下に示します。

- 一意として宣言されたカラムでプライマリ・キー・カラムが定義される
- プライマリ・キーを構成するカラムでインデックスが定義される
- 一意性制約で定義されているカラムでインデックスが定義される

Adaptive Server Anywhere は、プライマリ・キー・カラム、外部キー・カラム、および一意性制約を持つカラムに自動的にインデックスを付けます。一意性制約を（いずれにしても冗長である）プライマリ・キー定義に追加するか、自動的にインデックス付けされるカラムで明示的にインデックスを作成すると、重複インデックスが生じます。

可能な限り重複インデックスを削除することを強くお勧めします。

インデックスが重複しているかどうかを判断する方法

インデックスが重複しているかどうかを判断するために Adaptive Server Anywhere が使用する方法を以下に示します。この説明では、スキーマでインデックス idx1 と idx2 が定義されていると仮定します。この例では、データベース・サーバは、インデックス idx2 が一意であるため idx1 も一意であると推理します。

以下の状況が存在する場合、インデックスは重複であるとみなされます。

- idx1 と idx2 は同じカラムで定義されており、同じ順序を持ち、各カラムのプロパティ ASC/DESC がまったく同じである
- idx2 はプライマリ・キー・インデックスであるが、idx1 はそうではない
- idx2 は宣言された一意のインデックスであるが、idx1 はそうではない
- idx2 は外部キー・インデックスであるが、idx1 はそうではない
- idx2 は宣言されたクラスタード・インデックスであるが、idx1 はそうではない
- 上記のいずれでもない (この場合、Adaptive Server Anywhere は idx2 の複製となる idx1 をランダムに選択します)

たとえば、以下のインデックスが定義されていると仮定します。

```
CREATE INDEX idx1 ON T (T.A ASC, T.B DESC)
CREATE UNIQUE INDEX idx2 ON T (T.A ASC, T.B DESC)
```

この設計の場合、idx1 は idx2 の複製とみなされます。

重複したインデックスの削除方法

1. 以下のスクリプトをコピーして Interactive SQL に貼り付け、実行します。このスクリプトは、重複インデックスをリストしてから、インデックスを削除するストアド・プロシージャを作成します。

```
-- ここからコピーを開始する

call dbo.sa_make_object('procedure','sa_duplicate_indexes','dbo')
go

alter procedure dbo.sa_duplicate_indexes( )
result(
user_name char(128),
table_name char(128),
table_id unsigned int,
index_name char(128),
index_id unsigned int,
index_unique char(1),
index_cols varchar(1000),
index_seqs varchar(300),
dup_index_name char(128),
dup_index_id unsigned int,
dup_index_unique char(1),
dup_index_seqs varchar(300) )
```

```
begin

declare local temporary table simple_indexes(
user_name char(128) not null,
table_name char(128) not null,
table_id unsigned int not null,
index_name char(128) not null,
index_id unsigned int not null,
index_unique char(1) not null,
index_cols varchar(1000) not null,
index_seqs varchar(300) not null,
reverse_index_seqs varchar(300) not null
) on commit preserve rows;

declare local temporary table duplicate_indexes(
user_name char(128) not null,
table_name char(128) not null,
table_id unsigned int not null,
index_name char(128) not null,
index_id unsigned int not null,
index_unique char(1) not null,
index_cols varchar(1000) not null,
index_seqs varchar(300) not null,
dup_index_name char(128) not null,
dup_index_id unsigned int,
dup_index_unique char(1) not null,
dup_index_seqs varchar(300) not null,
) on commit preserve rows;

insert into simple_indexes
select u.user_name,
t.table_name,
t.table_id,
i.index_name,
i.index_id,
i."unique",
DT.cols,
DT.seqs,
DT.rev_seqs
from SYS.SYSINDEX i
join SYS.SYSTABLE t on (t.table_id=i.table_id)
```

```
join SYS.SYSUSERPERMS u on (u.user_id=t.creator),
lateral (select list(c.column_name order by ixc.sequence),
list(ixc."order" order by ixc.sequence),
list( (if ixc."order" ='A' then 'D' else 'A' endif) order by ixc.sequence)
from SYS.SYSIXCOL ixc
join SYS.SYSCOLUMN c
on (c.column_id=ixc.column_id and c.table_id=ixc.table_id)
where ixc.table_id=i.table_id
and ixc.index_id=i.index_id
) as DT( cols, seqs, rev_seqs)
where i.creator <> 0
;

// Find all-ascending or all-descending indexes which are duplicate of PKs

insert into duplicate_indexes (
user_name,
table_name,
table_id,
index_name,
index_id,
index_unique,
index_cols,
index_seqs,
dup_index_name,
dup_index_seqs,
dup_index_unique
) (
select S.user_name,
S.table_name,
S.table_id,
S.index_name,
S.index_id,
S.index_unique,
S.index_cols,
S.index_seqs,
'primary key',
tpk.seqs,
'P'
from simple_indexes S,
```

```
(select list(column_name order by column_id), list('A'), table_id
from SYS.SYSCOLUMN
where pkey='Y'
group by table_id ) tpk( clist, seqs, table_id)
where
tpk.table_id= S.table_id and
(index_seqs not like '%D%' or
index_seqs not like '%A%' )
and
index_cols = tpk.clist
)
;

// Find all-ascending or all-descending non-unique indexes which are duplicate of Fks

insert into duplicate_indexes (
user_name,
table_name,
table_id,
index_name,
index_id,
index_unique,
index_cols,
index_seqs,
dup_index_name,
dup_index_id,
dup_index_seqs,
dup_index_unique
) (

select S.user_name,
S.table_name,
S.table_id,
S.index_name,
S.index_id,
S.index_unique,
S.index_cols,
S.index_seqs,
tfk.role,
tfk.foreign_key_id,
tfk.seqs,
```

```
'F'
from simple_indexes S,
(select list(column_name order by primary_column_id),
list( 'A' ),
fk.foreign_key_id, fk.foreign_table_id, fk.role
from SYS.SYSFOREIGNKEY fk
join SYS.SYSFKCOL fkc
on (fk.foreign_table_id=fkc.foreign_table_id
and fk.foreign_key_id=fkc.foreign_key_id)
join SYS.SYSCOLUMN c
on (c.table_id=fkc.foreign_table_id
and c.column_id=fkc.foreign_column_id)
group by fk.foreign_key_id, fk.foreign_table_id, fk.role ) as tfk(clist, seqs, foreign_key_id,
foreign_table_id, role )
where
(index_seqs not like '%D%' or
index_seqs not like '%A%' )
and S.index_unique = 'N'
and
tfk.foreign_table_id=S.table_id
and
S.index_cols = tfk.clist

) ;

// Find duplicate indexes which are not keys and are not unique constraints

insert into duplicate_indexes (
select
S1.user_name,
S1.table_name,
S1.table_id,
(if S1.index_unique = 'U' then S2.index_name else S1.index_name endif) as index_name,
(if S1.index_unique = 'U' then S2.index_id else S1.index_id endif) as index_id,
(if S1.index_unique = 'U' then S2.index_unique else S1.index_unique endif) as index_unique,
S1.index_cols,
(if S1.index_unique = 'U' then S1.index_seqs else S2.index_seqs endif) as index_seqs,
(if S1.index_unique = 'U' then S1.index_name else S2.index_name endif) as dup_index_name,
(if S1.index_unique = 'U' then S1.index_id else S2.index_id endif) as dup_index_id,
(if S1.index_unique = 'U' then S1.index_unique else S2.index_unique endif) as dup_index_unique,
(if S1.index_unique = 'U' then S1.index_seqs else S2.index_seqs endif) as dup_index_seqs
```

```
from simple_indexes S1,
simple_indexes S2
where S1.index_id > S2.index_id
and S1.index_cols = S2.index_cols
and S1.table_id = S2.table_id
and ( S1.index_seqs = S2.index_seqs OR
S1.index_seqs = S2.reverse_index_seqs )
and ( S1.index_unique <> 'U' OR S2.index_unique <> 'U' )

);

select * from duplicate_indexes;
end
go

call dbo.sa_make_object('procedure','sa_drop_index','dbo')
go

alter procedure dbo.sa_drop_index(
in @user_name char(128),
in @table_name char(128),
in @index_name char(128),
in @index_unique char(1) )
begin
if (@index_unique = 'Y' OR @index_unique ='N') then
message 'The index ' || @user_name || '.' || @table_name || '.' || @index_name || '( ' ||
@index_unique || ' )' || ' is being dropped.' to client;
execute immediate
'drop index "' || @user_name || '"."' || @table_name || '"."' || @index_name || '"';
else
if (@index_unique = 'U' ) then
message 'Warning! Redundant unique constraint on ' || @user_name || '.' || @table_name || '.' ||
@index_name || ' should be dropped.' to client;
else
message 'Warning! Redundant index ' || @user_name || '.' || @table_name || '.' || @index_name || '( '
|| @index_unique || ' )' || ' is not dropped.' to client
end if
end if;
end
go
```

-- ここでコピーを終了する

2. 以下の文を実行して、重複インデックスをリストします。

```
SELECT * FROM sa_duplicate_indexes( ) AS DT
```

3. 以下のスクリプトをコピーして Interactive SQL に貼り付け、実行します。このスクリプトは重複インデックスを削除します。

```
begin
for l1 as c1 cursor for (select distinct user_name, table_name, index_name, index_unique from
sa_duplicate_indexes() )
do
call sa_drop_index(user_name, table_name, index_name, index_unique);
end for
end
```